Advanced Review

Large-scale data mining using genetics-based machine learning

Jaume Bacardit[1]* and Xavier Llorà[2]

In the last decade, genetics-based machine learning methods have shown their competence in large-scale data mining tasks because of the scalability capacity that these techniques have demonstrated. This capacity goes beyond the innate massive parallelism of evolutionary computation methods by the proposal of a variety of mechanisms specifically tailored for machine learning tasks, including knowledge representations that exploit regularities in the datasets, hardware accelerations or data-intensive computing methods, among others. This paper reviews different classes of methods that alone or (in many cases) combined accelerate genetics-based machine learning methods.

How to cite this article:
WIREs Data Mining Knowl Discov 2013, 3: 37–61 doi: 10.1002/widm.1078

INTRODUCTION

We are living in the petabyte era. We have larger and larger data to analyze, process, and transform into useful answers for the domain experts. Robust data mining tools, able to cope with petascale volumes and/or high dimensionality producing human-understandable solutions are crucial to numerous aspects of science and society. Genetics-based machine learning (GBML) techniques[1,2] are perfect candidates for this task. Recent advances in representations,[3–7] learning mechanisms,[7–13] and theoretical modeling[14–18] along with experimental comparisons against other machine learning techniques realized across benchmark datasets[2,19–21] have shown the potential of evolutionary computation (EC) techniques in herding large-scale data analysis.

If GBML techniques aspire to be a relevant player in this context, they need to have the capacity of processing vast amounts of data. They need to process these data within reasonable time. Moreover, massive computation cycles are getting cheaper, allowing researchers to have access to unprecedented computational resources on the edge of petascale computing. Several topics are interlaced in these two requirements: (1) having the proper learning paradigms and knowledge representations, (2) understanding them and knowing when they are suitable for the problem at hand, (3) using efficiency enhancement techniques, and (4) transforming and visualizing the produced solutions to give back as much insight as possible to the domain experts.

This review paper aims at shedding light on the above-mentioned questions, following a road map that starts by describing what large scale means, and why large is a challenge and opportunity for GBML methods. Afterwards, we show how to tackle this opportunity through the use/combination of multiple facets: parallel models, representations able to cope with large dimensionality spaces, advanced exploration mechanism or hardware implementations, among others. Each class of components that contributes to large-scale data mining tasks is described in detail. Furthermore, we show a few examples of how we can visualize the results of GBML systems. Then, we describe how we can model the scalability of the components of GBML systems via a better engineering that will make embracing large datasets routine. Next, we illustrate how all these ideas fit by reviewing a case study of GBML applied to a large-scale problem. Finally, we summarize the broad variety of mechanisms described throughout the manuscript, indicating how each of them contributes to the functioning of the GBML methods and discussing potential risks/benefits of combining them.

*Correspondence to: jaume.bacardit@nottingham.ac.uk

[1]Interdisciplinary Computing and Complex Systems (ICOS) Research Group, School of Computer Science, University of Nottingham, Nottingham, UK

[2]Google Inc., 1600 Amphitheatre Pkwy Mountain View, CA 94043

DOI: 10.1002/widm.1078

WHAT IS LARGE SCALE?

This section aims at describing the broad spectrum of problems that can potentially be included in the spectrum of 'large scale', not in terms of their respective domains, but in terms of the characteristics of the problem. The most relevant of these dimensions of large scale is, of course, the number of records. For instance, GenBank[22] (the genetic sequences database of the US National Institute of Health) as of October 2010 contains more than 125 million gene sequences (82 million in 2008) and more than 118 billion nucleotides (85 billion in 2008). The Large Hadron Collider is forecasted to generate up to 700 megabytes of data per second.[23] Next-generation sequencing technologies can sequence up to 1 billion base pairs in a single day.[24]

Moreover, large records not only come from the natural sciences. The Netflix prize[25] has probably been one of the most famous data mining challenges in the latest years (and a predecessor of what later in time has been known as *Crowdsourcing*). Netflix is a video rental company which in 2006 released an anonymized version of their movie ratings database of over an 100 million ratings from 480,000 customers and 18,000 movies. They challenged the public to provide a recommendation algorithm that was 10% better than their own proprietary method. In 2007, the best participant method was 8.43% better, 9.44% in 2008 and, finally, in 2009 a team managed to break the 10% barrier.[26]

However, large sets of records are not the only facet of the term of large scale. Another challenging aspect is the number of variables of the problem. As an example, the data resulting from microarray analysis[27] (one of the most widespread technologies of molecular biology research) usually generates records with tens of thousands of variables. Microarrays generally have an added difficulty: Generating each record is very expensive, thus datasets tend to have very few samples which creates a very high imbalance between records and variables. In this scenario, many machine learning methods tend to overfit the data. Thus, very sophisticated statistical validations mechanisms are employed to assure that the results of the data analysis process are statistically sound.[28] Other natural sciences domains, such as protein structure prediction (PSP),[29] routinely generate datasets with hundreds (if not more) of variables, in this case coupled with a number of records at least in the hundred thousand range.

The (large) number of classes is another source of difficulty for machine learning methods. There are well-known information retrieval/text mining datasets with an extremely high number of classes,[30] datasets where the classes are not just a set of labels, but they have a hierarchical structure[31] or datasets where the frequency of certain classes is very low, that is, that present class imbalance. This phenomenon has been studied for long in the data mining community,[32] and probably on its own cannot be considered as one of the facets of difficulty in large-scale data mining. However, as datasets grow and tend to become more heterogeneous, the class imbalance problem becomes a recurrent issue, and it cannot be ignored. The GBML field has also seen some specific and principled work toward tackling the class imbalance problem.[16]

Given the different facets that large-scale datasets can take, it is necessary to study their characteristics in a principled and quantitative way. Complexity metrics[33] have been proposed that assess the datasets in many different ways such as their sparseness, shape of its decision frontier, dimensionality, and so on. Furthermore, these metrics have been applied to study the domains of competence of GBML methods.[34,35] However, their widespread application to large-scale domains is still limited, given that the calculation of most of these metrics does not scale well for large numbers of instances or attributes.

When does a dataset becomes large scale? This is a very difficult question to address, because it depends on the type of learning and the resources available at each organization for performing data analysis. The simplest definition, albeit vague, is that a dataset becomes large scale when its size starts becoming an issue to take into account explicitly in the data mining process.[36] If an algorithm just needs to perform a single pass through a training set it may easily process hundreds of millions of records, but if the training set needs to be used again and again the algorithm may start struggling with just a few tens of thousands of instances. The same situation happens with the number of attributes. Depending on the number of parameters involved in the knowledge representation, the tractability frontier can be moved around.

THE ADVANTAGES AND CHALLENGES OF GBML FOR LARGE-SCALE DATA MINING

As said in *Introduction*, GBML methods are perfect candidates for large-scale data mining for many reasons, but the main of them comes from their main working principle: evolution. Evolutionary algorithms are innately parallel because they are population based, and this parallelization capacity has been exploited for many years to improve their

efficiency, in what is known as parallel genetic algorithms[37] or, in a more general perspective, parallel metaheuristics.[38] The different stages of an evolutionary algorithm require more of less amount of synchronization points, going from no synchronization and hence maximum potential parallelism (the evaluation of the population) to some other stages that are highly sequential (e.g., the selection algorithm). Considering these constraints, several paradigms of parallelism exist with different strategies for the distribution of the computation effort of the evolutionary algorithm that attempt to boost the efficiency of the algorithm in different ways, such as the master–slave paradigm (just parallelizing the evaluation of the individuals), the island model (with distributed populations and carefully defined policies for the migration of individuals between them) or cellular evolutionary algorithms, where the population has a specific topology that defines the neighbors of each individual and hence, the potential channels of communication. The choice of paradigm depends on many different factors, and theoretical models exist[37] to inform this decision. The section of the paper on parallel models will describe several specific examples of the application of these paradigms to GBML.

However, as the focus of this paper is not just large-scale problems, but *large-scale data mining* problems, we have to be aware of some challenges as well, and the main one of them is dealing with large volumes of data. Generally, EC methods are purely computational, that is, a very expensive fitness function needs to be run millions (or more) of times, but this function does not need to access large volumes of information. Our context is totally different, and without dealing carefully with data management, all the potential parallelism of EC can be in jeopardy.

The choices and strategies for data management totally depend on the chosen hardware platform. The strategy used for experiments run in a single-CPU or a traditional (small) cluster (e.g., HDF5[39]) probably will not scale to data-intensive computing environments with hundreds or thousands of computing nodes and other alternative methods (for instance, Hadoop Distributed File System[40] or MongoDB[41]) would be needed, and neither of them would be suitable for general-purpose graphics processing units (GPGPUs). The aim of this paper is to focus on the data mining methods and not on the data management itself, thus we will not discuss these options in detail with the exception of the GPGPU case, as handling the data and mining it are so intimately bound that they cannot be described in separate. What we will discuss in depth in several sections is what to do once the data has been parsed and loaded.

KALEIDOSCOPIC LARGE-SCALE DATA MINING

The aim of this section is to present the four different types of techniques that have been used to adapt GBML systems for large-scale data mining tasks. As any taxonomy probably we are not going to be able to cover every possible scenario but we believe that this classification covers most published works of GBML for large-scale datasets. Moreover, these four categories are not mutually exclusive. There are many case where the actual method is a mix of a few of these types of scaling up techniques, as we will show later in the manuscript. We would also like to clarify that the techniques included in this classification are (mostly) specifically tailored for GBML methods. There are many efficiency enhancement techniques for EC methods reported in the literature,[42] which we will not cover in this paper.

The four types of methods are

1. *Software solutions*. In this category, we include methods that modify the data mining methods to improve their efficiency without special/parallel hardware.
2. *Hardware acceleration techniques*. This category includes methods that employ customized hardware such as vectorial instructions of modern standard processors [MultiMedia eXtension (MMX), Streaming SIMD Extensions (SSE), etc.] or specialized computing hardware such as GPGPUs.
3. *Parallelization models*. This category includes all classic methods (in opposition to the next one) to use multiple computing nodes to perform a single data mining experiment.
4. *Data-intensive computing*. This category includes parallel scenarios that go beyond traditional high-performance computing clusters, where the number of computing cores go beyond tens of thousands, they are heterogeneous, distributed, and sometimes fail. This scenario has become very popular in the last few years thanks to the appearance of actors such as Google with their MapReduce data analysis methodology[43] and its open-source implementation, Hadoop.[44]

The following sections describe each of these four categories of scaling up methods.

SOFTWARE SOLUTIONS

This section covers methods that modify its inner workings to improve their efficiency without relying on parallel implementations nor special hardware. Because of the broad diversity of such mechanisms, this section will be split in four different categories of methods:

1. *Windowing mechanisms*, where a subset of examples from the training set is used for fitness computations, and how the subset/s are created and how often are they changed varies across methods.
2. *Exploiting regularities in the data*, to precompute classifications, or avoid irrelevant computational effort
3. *Hybrid methods* that substitute or extend the traditional exploration operators of EC methods with smarter alternatives.
4. *Fitness surrogates* that generate a *cheaper* estimation of the fitness function

Windowing Mechanisms

By windowing algorithms we understand methods that instead of evaluating candidate solutions on the whole training set they use only a subset of examples. This type of mechanisms has been used for many years across the whole machine learning community[45–48] as well as in the specific context of GBML. Alex Freitas[49] proposed a taxonomy for this kind of methods depending on how often the system changes the chosen subset of examples:

- *Individual-wise*, changing the subset of the training examples used for each fitness computation.
- *Run-wise*, selecting a static subset of examples for the whole evolutionary process.
- *Generation-wise*, changing the subset of the training examples used at each generation of the evolutionary process.

For a genetic algorithm (GA) using a flat population is quite unfair to change the fitness function among individuals in the same generation. When the population has specific neighborhood relations (e.g., a cellular GA) this policy has shown some successful results.[50] Selecting a static subset of examples before the learning process is what is known as prototype selection, and there are many successful GBML methods to perform such process.[51,52] In the rest of this subsection, we will focus on the third option, namely the generationwise methods.

How can this generation-wise training examples subset be selected? Most systems reported in the literature perform a pure random sampling process to select the subset. One example of this approach, applied to attribute selection (but able to be extrapolated to rule induction), is Ref 53. Other approaches refine the random sampling in several ways. Difficult instances (missclassified) can be used more frequently than correctly classified instances,[54] or 'aged' instances (instances not used for some time) can have more probability of being selected.[54]

Also, Ref 53 discusses the issue of choosing the final solution of the learning process from all the individuals in the final population. If the sample used is small, maybe the best individual of the last iteration is not representative enough of the whole training set. The authors propose two approaches: the first one is to use the whole set in the last iteration. The other approach is to perform a kind of voting process among all individuals in the population.

ILAS (Incremental Learning with Alternating Strata)[55] is a windowing method that was originally proposed for the GAssist Pittsburgh GBML method[56] but that later in time has been also adapted to other learning systems.[3] This mechanism first divides the training set into a set of strata. Each strata has the same class distribution as the overall training set in an effort of making it a good representative of all the overall dataset. Afterward, each iteration of the GA will use each of these strata in a round-robin fashion. When using such a mechanism the practitioner would expect to have speedups equivalent to the number of strata, but the reality is that many times, in the specific case of GAssist (with variable-length individuals), the obtained speedups were better than the number of strata. The reason was that with moderate number of strata the windowing mechanisms managed to introduce generalization pressure that created more compact (i.e., short) individuals which were faster to evaluate[56] and also obtained in most cases better test accuracy. ILAS has been successfully applied in many real world problems, both small[20,56] and large.[3,8,57–59]

RSS-DSS (random subset selection–dynamic subset selection)[60] is a much more complex windowing method applied on top of a steady-state genetic programming classifier. The windowing process is split into two hierarchically organized stages. The first stage, RSS is similar to ILAS, it partitions the whole training set into a series of B blocks. The authors determine the number of blocks so each of them can fit in memory. Within each block, a second windowing process called DSS is performed. DSS dynamically samples with replacement examples from

If $Leu_{-2} \in [-0.51, 7] \wedge Glu \in [0.19, 8] \wedge Asp_{+1} \in [-5.01, 2.67] \wedge Met_{+1} \in [-3.98, 10] \wedge Pro_{+2} \in [-7, -4.02] \wedge Pro_{+3} \in [-7, -1.89] \wedge Trp_{+3} \in [-8, 13] \wedge Glu_{+4} \in [0.70, 5.52] \wedge Lys_{+4} \in [-0.43, 4.94]$ **then predict** $alpha$

FIGURE 1 | Rule generated from a Bioinformatics dataset with 300 attributes.

its corresponding RSS block given two constraints: the age of an examples (the last time it was used) and its difficulty (based on training error). The examples within a DSS sample are used for a fix number of GP iterations which is set in proportion to the population size, and the number of DSS samples generated per block is variable (but with an upper limit). Blocks that are easier to learn (based on the fitness of the best individual) use less number of DSS samples. Finally, the blocks are picked at random with uniform probability, instead of the round-robin method of ILAS. This method was evaluated on a half a million examples dataset, the popular KDDCUP'99 intrusion detection dataset held at the UCI repository.[61]

Windowing mechanisms in GBML have not been applied only to classification tasks, but also for other types of problem such as prototype selection[51] or subgroup discovery.[62]

A crucial issue about most windowing mechanism is how much we can increase the number of strata without constraining the success of the learning process. Intuitively, we can think that the number of strata can be increased while each of the stratum is still a good representative of the overall training set. Initial steps toward answering this question have been done in the particular context of ILAS. In Ref 55, a simple theoretical model for the success of ILAS was proposed, based on the assumption that the number of rules in the solution is known and that each rule covers a similar number of examples. Under those assumptions, a closed form equation can be estimated for the probability of success of the stratification, defined as the probability that each strata had at least one instance covered by each rule in the solution.

$$P(\text{success}/s) = \mathrm{e}^{-rs \cdot \mathrm{e}^{-\frac{pD}{s}}} \quad (1)$$

Where s is the number of strata, r the number of rules, D is the training set size, and p is the probability that a random problem instance represents a particular rule.

Exploiting Regularities in the Data

This subsection covers methods that manage to alleviate a portion of the run-time of the system by precomputing some parts of the evaluation process or by skipping computations in part of the data that is identified to be irrelevant. These methods exploit the fact that in most problems the data does not uniformly cover the whole search space or that the individual subparts of a solution (e.g., rule set, decision tree, and so on) do not consider all the variables of the problem at the same time.

Some methods[63] are able to precompute the instances that match a rule for problems with discrete/discretized attributes by aggregating together examples that present the same value for the attributes of the problem. The result of this process is a tree structure that is able to retrieve very efficiently all the instances of the training set that belong to a specific value of a specific attribute. Afterward, the match process of a given rule is just the intersection of the sets generated for each attribute of the problem. The retrieval process will be fast because the tree is rather sparse as the search space is not uniformly distributed. Other methods[64] use a very similar approach but for the dual problem: given an instance, identify the set of rules that match it. The latter approach is more challenging, as a population of rules changes more frequently than a training set (unless we are in a *stream data mining* context), thus it is not only to efficiently construct the tree, but also to update it.

A second family of methods exploits another regularity in the data, which is that not all attributes are equally important. To exemplify this issue, Figure 1 shows a rule has been generated from a Bioinformatics dataset[3] with 300 attributes. Only 9 attributes out of the 300 were present in this rule, hence the rest were irrelevant. In most systems, the match process for such a rule would look like shown in Figure 2. This algorithm would essentially waste 291 out of the 300 iterations of the *ForEach* loop because the attributes are not relevant. Thus, what can be done to alleviate this difficulty.

Different alternatives have been proposed in the literature. Some methods[65] opt for reordering the attributes in the problem from specific to general based on statistics extracted from the current population of rules. The rationale for this decision is that the most specific attributes are the most likely of stopping the match function to return *false*. Hence, if these attributes are evaluated first it is quite probable that the match algorithm performs just a few iterations of the *ForEach* loop in most situations. Still, in the cases


```
Function match (instance x, rule r)
Foreach attribute att in the domain
        If att is relevant in rule r and (x.att < r.att.lower or x.att > r.att.upper)
                Return false
        EndIf
EndFor
Return true
```

FIGURE 2 | Pseudocode of the match algorithm for an hyperrectangle rule representation.

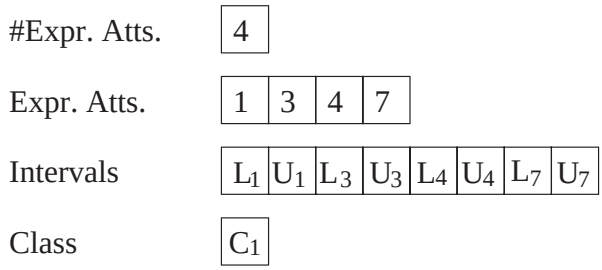

FIGURE 3 | Example of a rule in the attribute list knowledge representation with four expressed attributes: 1, 3, 4, and 7. l_n = lower bound of attribute n, u_n= upper bound of attribute n, $c1$ = Class 1 of the domain. © Memetic Computing by Springer-Verlag Berlin/Heidelberg. Reproduced with permission of Springer-Verlag Berlin/Heidelberg in the format Journal via Copyright Clearance Center.

where a whole rule matches, the irrelevant attributes need to be evaluated.

Therefore, could we simply get rid of these irrelevant attributes? This is the aim of the attribute list knowledge representation (ALKR).[3] In this representation, each rule only holds information for the attributes of the problem that it considers to be relevant. The way of achieving this consists in (1) updating the representation and (2) adding two new exploration operators. The representation for a rule is exemplified in Figure 3. The most important element of the representation (and the one that gives its name) is the vector that contains the attributes considered to be relevant for this rule. The rule will only hold intervals for these attributes and the crossover and mutation and match function will only use them and no other attribute. A key question remains, which is how to identify which are the correct attributes for that rule. This is the job of two extra operators, called *generalize* and *specialize*, which are in charge of updating the list by either removing or adding randomly chosen attributes, respectively. As the list of attributes is variable-length, it may be subject to the bloat effect.[66] To control the bloat, it is necessary to use the representation within a system that employs a fitness function that rewards compact (i.e., using few attributes) rules, such as the BioHEL system,[3] which will be described in a bit more detail in *Real Examples*. ALKR uses a 'virtual one-point crossover operator' that has been adapted to cope with individuals that may contain different sets of attributes. This is done by simply considering that the missing attributes were there momentarily and applying a simple one-point crossover to the *virtual* complete rules. The match function remains unchanged, but it iterates over the list of attributes of the rule, not over all the attributes of the domain, thus the number of wasted iterations of the match function should be kept to a minimum.

The representation was evaluate in both small and large datasets[3] showing not only that it was more efficient than traditional representations but that it was also learning better, obtaining higher test accuracy in most datasets. The reason for this is that given that each rule only contains relevant attributes, the exploration operators are more focused and hence the learning process becomes more efficient.

Hybrid Methods

Large-scale datasets very often present an extremely vast search space. In these cases, the main challenge becomes to explore this vast space efficiently using smart/directed exploration mechanisms, either in combination or replacing the traditional operators of EC methods. This trend is not specific to GBML, it has affected the overall EC field. The two most popular classes of these methods probably are estimation of distribution algorithms (EDAs)[67] and memetic algorithms (MA).[68] The former methods estimate a model of the structure of the problem and then generate offspring according to this model, whereas the latter perform a combination of local search (LS) and global search.

There are many examples of the application of both classes of advance exploration mechanisms to GBML methods. Several flavors of EDA have been employed hybridized within both Pittsburgh and Michigan learning classifier system (LCS) methods. For instance, the eXtended Classifier System (XCS) Michigan LCS was integrated[9] with two

different types of EDA: the extended compact genetic algorithm (eCGA)[69] and the Bayesian Optimization Algorithm.[70] These two methods derive global structural information from the best rules in the population, which later is used to inform the crossover operator when generating new offspring by preventing crossover from breaking good building blocks (BBs) in the individuals it is crossing.

The compact classifier system (CCS)[71] is a recent integration of EDAs within the framework of a Pittsburgh LCS, using the compact genetic algorithm (CGA)[72] CGA is run iteratively to generate different rules. Different perturbations of the initial solution of CGA are needed to generate different rules, and the individuals in CCS store a set of such perturbations. The objective of CCS is to determine the minimum set of rules that creates a maximally general solution. Later in time, Llorà et al.[73] proposed $\chi eCCS$, an extension of CCS. $\chi eCCS$ evolves a population of rules using the model building and recombination mechanisms of eCGA. A niching method using restricted tournament selection[74] was employed to guarantee that the population learns (all at once) all the rules needed to solve the domain. Experiments showed that this method scales quadratically in relation to the problem size for the Multiplexer family of problems. EDAs have also been used for other machine learning tasks such as feature or prototype selection.[75,76]

Memetic-like exploration methods have also been employed in GBML systems. One such example was the Memetic Pittsburgh Learning Classifier System (MPLCS).[8] This method extracted statistics from the evaluation process of the GAssist[56] Pittsburgh GBML system such as which parts of which rules were performing well or not, or what part of the input space was covered by each rule. Then GAssist was extended with different types of LS operators that were performing directed search based on these statistics. Two kinds of LS operators were proposed: (1) rulewise operators that were editing individual rules to either specialize them to remove misclassifications or generalize them to add more positive matches, and (2) a rule setwise operator that given a set of N parents (where generally $N > 2$), it constructed a single offspring by taking the rules of all the parents and choosing the best subset of them and their correct order (as GAssist generates ordered rule sets) with the objective of generating a rule set with the highest possible training accuracy and as few rules as possible. This type of rule setwise exploration operators has also been used in other contexts such as multistep problems[77] or genetic fuzzy systems[78] but always using a traditional crossover framework of two parents–two offsprings. Moreover, memetic operators have also been employed in Michigan LCS to explore not just the predicate of a rule but also its performance parameters.[11,79] Also, memetic search has been used for machine learning tasks other than classification, such as in prototype selection.[51]

Fitness Surrogates

Fitness surrogates have been extensively used in the EC context for efficiency enhancement purposes. These methods construct a *cheap* estimation of the fitness function, and use it instead of the original, exact, fitness formula whenever possible. Generally, the success of these methods comes when the proper regime of combination between the surrogate and the original fitness formula is identified. Theory exists to determine the optimal regime of combination of both fitness functions based on the bias, variance, and cost of the original fitness and the surrogate.[42,80]

Recent works have applied the principle of fitness surrogates to GBML.[81,82] Both works use a methodology recently proposed[83] for the construction of fitness surrogates based on the model-building process of EDAs.[67] EDAs, as mentioned above, generate models of the structure of the problem to bias the exploration process. Some EDAs estimate the BBs of the problem. Each block consisting in a set of problem's attributes that interact between them. Afterward, the fitness contribution of each BB is estimated from the individuals in the population using regression, and the surrogate function for a given individual is constructed as the addition of the estimated fitness contributions of the BBs it contains. In Ref 81, two different model building methods were evaluated for GBML purposes: eCGA,[69] which generates set of non-overlapped BBs; and DSMGA,[84] which generates a list of (possibly) overlapped BBs. The latter method was able to generate more reliable fitness surrogates than the former for problems where the same variable was involved in more than one BB, which is a very usual situation in the context of machine learning.[18] The scalability of this methodology in relation to the size of the BBs was the focus of Ref 82, where fitness surrogates for various classes of hierarchical decomposable problems with varying sizes were evaluated.

HARDWARE ACCELERATION TECHNIQUES

In this section, we will include GBML methods that use specialized hardware and will include two parts: techniques using vectorial instructions and techniques using GPGPUs.

Usage of Vectorial Instructions to Boost the Matching Process

One of the main characteristics of the early supercomputers such as the Cray-1 was the use of vectorial processing units, where the same operation was applied in parallel and in high efficiency to a whole array of variables, in what is known as single-instruction multiple-data (SIMD) paradigm. Despite its efficiency, this paradigm was also complex to implement, and its use faded in time in favor of simpler architectures that could run at faster clock speeds. SIMD reappeared in the mid 1990s, not in the high-performance computing market but rather in the consumer market, when Intel introduced the MMX instruction set for their Pentium processors, which allowed to reuse each of the registers in the Floating Point Unit of the processor to pack several integers within (e.g., 2×32 bit integers, 4×16 bit integers or 8×8 bit integers) and perform SIMD operations on them. Later in time, this capacity was expanded in types of operations and sizes of the registers with Intel's own SSE (Streaming SIMD Extensions) while other hardware vendors proposed their own alternative instruction sets (e.g., PowerPC's Altivec or AMD's 3DNow!).

As shown in *Software Solutions*, the process of matching a rule with a given instance generally consists in iteratively checking if each attribute in the problem matches, and stopping the process when any of such tests fail. Using vectorial instructions this loop can be unrolled, and several attribute tests can be performed in parallel. An example of this technique was presented in Ref 85 for problems with binary attributes. That approach was based on the use of vectorial instructions coupled with a specific encoding for rules and instances. Two bits were employed to represent both the condition associated to a certain attribute in the rule and the value of the attribute in the instance as follows:

Predicate	Encoding	Value	Encoding
Att is 0	10	0	10
Att is 1	01	1	01
Att is irrelevant	00/11	–	–

Given this encoding, checking (for a given attribute) if a condition matches an instance is as simple as computing the logical *and* operation between the encodings for condition and value and then comparing the result with the original instance attribute value. If both values are *equal*, then the condition has been matched. However, a rule contains multiple conditions that need to be check. Current CPUs provide logical operations for 32-bit unsigned integers. This has two advantages: (1) allowing packing up to 16 conditions per integer—not wasting any memory and (2) performing the matching process described at the unsigned integer level will provide the parallel matching of 16 conditions at once. The use of the 128-bit vectorial instructions, which can perform the same operation in parallel to four 32-bit words, allows the extension of the exact same mechanism to match up to 64 conditions at once. Figure 4 presents a sample implementation of both the 16 conditions CPU matching and the 64 conditions SSE matching. This vectorial instructions-based matching algorithm was later in time extended to deal with continuous attributes.[86]

```
#define RECODE_BLOCKS (2*NUM_CONDITIONS/\
                       (8*sizeof(unsigned int))+1)
#define RULE unsigned int*

int isRuleMatchedBinary ( RULE rule, INSTANCE ins ) {
     register int i,iFlag;

     for ( i=0, iFlag=1 ;
           i<=RECODE_BLOCKS && iFlag ;
           i++)
        if ( (rule[i]&ins[i]) != ins[i] )
            iFlag = 0;

     return iFlag;
}
```

```
#include <emmintrin.h>
#define RECODE_BLOCKS (4*(2*NUM_CONDITIONS/\
                       (8*4*sizeof(unsigned int))+1))
#define RULE unsigned int*

int isRuleMatchedSSE ( RULE rule, INSTANCE ins ) {
     register int iFlag = 1;

     // Matching using SSE2 instruction set
     register int i,iMax,tmp;
     __m128i vir,vii;

     for ( i=0, iMax=RECODE_BLOCKS/4  ;
           i<iMax && iFlag ;
           i++) {
        tmp = i*4;
        vir = _mm_load_si128((__m128i*)&rule[tmp]);
        vii = _mm_load_si128((__m128i*)&ins[tmp]);
        vir = _mm_and_si128(vir,vii);
        vii = _mm_cmpeq_epi32(vir,vii);
        iFlag &= (-1 == _mm_movemask_epi8(vii));
     }

     return iFlag;
}
```

FIGURE 4 | Parallel match algorithm using binary encoding (top) and Streaming SIMD Extensions (SSE) vectorial operations (bottom). Reprinted with permission from Ref 85. Copyright 2006 Association for Computing Machinery, Inc.

High-Performance Fitness Computation Using GPGPUs

GPGPUs are a relatively new technology but one that has created a huge impact in all fields of scientific

computation because it has enabled researchers to access very large amounts of raw computational power for a small fraction of the price than an equivalent amount of power costs in a traditional high-performance computer cluster. This is achieved by exploiting with a different angle a hardware originally designed for the video games industry. Within a GPGPU, a very high number of threads, all executing the same code, can be run in parallel, each of them operating on different data. Thus, it changes the classic SIMD paradigm into a single-program multiple data. However, the communication between threads and from the threads to the main memory of the GPGPU is far from trivial and in many cases it requires a significant change of the programming model to find the perfect equilibrium between usage of raw CPU power and access to memory.

Probably the most popular of the GPGPUs platforms is *CUDA* (computer unified device architecture),[87] a parallel computing architecture developed by NVIDIA. Threads in CUDA are organized in geometrical lattices at two different levels. First, threads are aggregated in blocks, and then blocks are aggregated to compose the overall data distribution. Threads within a block can communicate between themselves at high speed through the usage of *shared memory*. Threads can also access other types of memory. *Global memory* is the main memory of the GPU. It can be accessed by all threads, but it is slow, specially if all threads are trying to read/write at the same time unless this access is coalesced (consecutive threads in a block accessing consecutive position in memory). *Constant memory* can hold data that does not change throughout the whole execution of the program and can be accessed at high speed, but it is very limited in size. *Local memory* is used to store data specific to each thread. This memory is not used explicitly by the programmer but rather by the CUDA compiler. *Texture memory* is a read-only (from the thread's point of view) type of memory that is optimised for memory access patterns following a two-dimensional spatial locality. The proper use of each type of memory is crucial for obtaining good performance out of CUDA.

GPGPUs have been widely used in recent years to boost the performance of all kinds of EC methods[88,89] as well as in the specific context of GBML.[90–91] Moreover, GPGPUs have been integrated into many different machine learning methods.[92–95] In this paper, we will describe one specific example for illustrative purposes: The CUDA-based fitness computation[96] of the BioHEL[3] GBML system.

BioHEL is a rule-based GBML method that applies the iterative rule learning (IRL) paradigm.[97] Its fitness computations are the result of matching each rule in its population against a training set. Moreover, BioHEL also contains other efficiency enhancement methods such as the ILAS windowing scheme[55] or the ALKR rule representation,[3] described in the previous sections. The design of the GPGPU-based fitness evaluation was made so it could be integrated with these other methods and their respective speedups could become cumulative.

Specifically, the most costly part of BioHEL fitness formula is computing three metrics:

1. The number of instances in the training set that match the condition of the rule.
2. The number of instances in the training set that match the class of the rule.
3. The number of instances in the training set that match the class and the condition at the same time.

In most situations, it is not really necessary to know which *specific* instances were matched by each rule (unless memetic operators, such as those described earlier in this paper, are used). This is very important in this situation, as extracting data from a GPGPU is very costly (much more costly than sending data there). Thus, if the specific matches are not needed, the GPGPU can count the matches (doing what is generally known as *reducing the data*) itself and return back a data structure proportional to the number of rules in the population, instead of being proportional to rules $\times$ instances.

The GPGPU-based solution worked in two stages (called *kernels*):

1. Kernel 1: Match and initial reduction. Each thread in the GPGPU is in charge of performing an individual match operation involving one rule and one instance. Afterward, it computes three bits corresponding to the three metrics described above. Then the reduction (counting) stage starts. This process can be performed at high speed if the threads are able to communicate between themselves through the shared memory.[98] The drawback is that shared memory is limited to the threads that belong to the same block (up to 1024 threads in the latest versions of CUDA). Thus, to maximize the power of this initial reduction it is necessary to design the grid (how the blocks in an experiment are distributed) so all the threads of a block operate on the same rule. At the end of the kernel the rule's performance counts are saved to the device's global

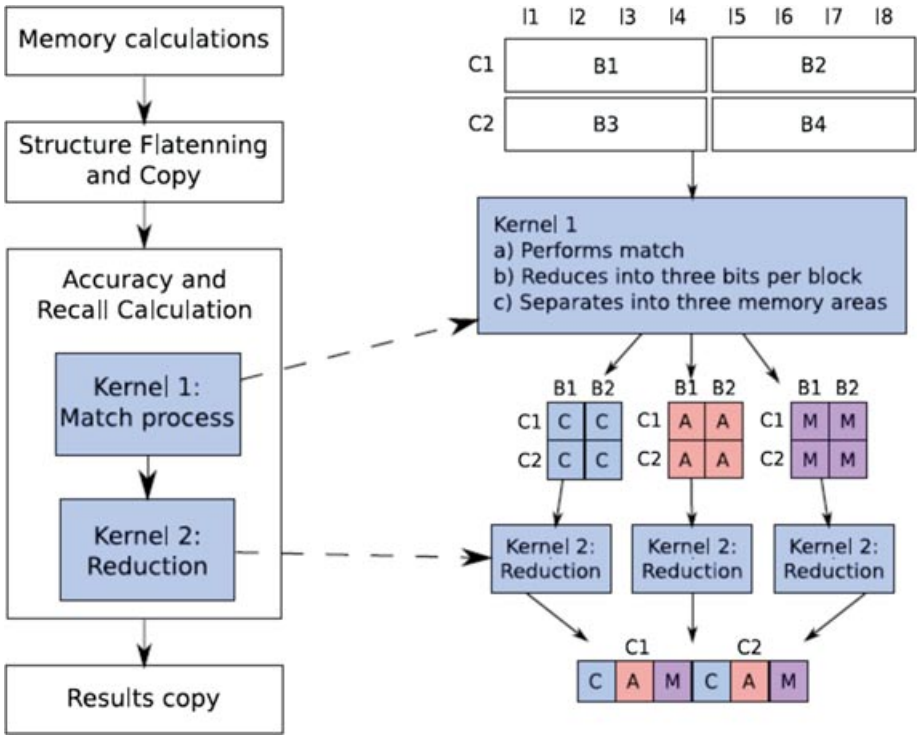

FIGURE 5 | Diagram of the computer unified device architecture based fitness computation pipeline of the BioHEL genetics-based machine learning system.

memory. This produces a large impact in the kernel's performance. However, this impact has been greatly reduced thanks to the initial reduction.

2. Kernel 2: Full reduction. For most problems (those with more than 1024 examples), it is necessary to perform a second reduction stage (or, for problems with more than 2^{20} examples a third stage, and so on) to get the final values for these three metrics. This is the aim of the second kernel. This kernel will be applied iteratively to reduce the data for each of the three metrics, one at a time, as this is more efficient than a single kernel reducing all three metrics at once.

Moreover, the training set had to be flattened (into simple data structures) and uploaded to the device's global memory before the kernels, and the final result of the reduction downloaded from the device into the computer's memory so BioHEL could continue its normal working cycle. The overall process is represented in Figure 5.

The speedups that can be obtained in CUDA greatly depend on being able to maintain occupied at most times all the computational resources (threads) in the GPU. This is a challenge as many factors can contribute to stall the processors in the GPU. We have already mentioned the most important of these factors: the memory access. If all threads try to write to main memory at once there is a bottleneck, hence the need to use efficiently the shared memory and the reduction methods to minimize the volume of global memory writes. Moreover, to efficiently read from the main memory consecutive threads (in CUDA's grid) should be reading consecutive positions of memory in what is called *coalesced* memory access. Another important aspect is the flow of the code being run in the threads. If an algorithm is essentially serial, it will be difficult to obtain any kind of speedup. This issue especially affects LS methods, such as the memetic operators mentioned above in *Hybrid Methods*. Moreover, even if an algorithm can be parallelized, if the code has many branches (e.g., ifs, loops), parts of the GPU will be stalled because blocks are splitted in warps, and all threads within each warp are synchronized so all run the same instruction at once in a true SIMD fashion. Hence, if different threads within a warp take different branches of an algorithm some of them will be stalled waiting for the rest. Finally, all threads within a block share a single register bank; so if a kernel needs to use a large number of registers, it can create a constraint in the total number of threads being run at once. In summary, while the potential

benefits of using CUDA are enormous, not all GBML methods are suitable to be adapted to this architecture because of their patterns of memory access or flow of code.

The consequence of these constrains is that we should only expect to obtain good performance out of GPGPUs when the code run in the GPU kernels is relatively simple. In Ref 96, we showed how we can successfully combine the ALKR representation with a GPGPU matching, but the key to that success was to limit the degree of parallelism of the kernels: Each kernel was performing the match process of a single example from the training set (or the current window) with a single rule in the population. That is, iterating over the attributes expressed in that rule. Other methods[90] implemented kernels where each thread was only matching *a single attribute* of a rule. This strategy would have not been possible to implement in combination with ALKR because each rule may have a different set of expressed attributes, and this would create a very sparse grid structure where many threads (those associated to attributes not present in the rule) would not be used, hence reducing the speedup of the system.

PARALLELIZATION MODELS

The following section contains a description of parallel implementations of GBML methods. It will be split into two parts: coarse-grained and fine-grained methods.

Coarse-Grained Parallel Models

By coarse-grained methods, we understand parallel implementations presenting a moderately low amounts of communication between them, generally correlated with a quite high computational load per node. Included in this category are methods with varying degrees of communication. The most extreme case are methods that present no communication at all between nodes. Most GBML (and in general EC) methods can exploit this option given that they are stochastic methods and their evaluation requires running them multiple times with different random seeds. This can be treated as a, rather obvious, example of coarse-grained parallelism. However, there are methods that explicitly exploit the act of running the algorithm multiple times, for instance, for ensemble learning.[99] In Ref 100 it was shown how by simply running the GAssist Pittsburgh GBML method multiple times, producing a rule set for each run, and then aggregating these rule sets using a majority vote, the ensemble managed to obtain higher test accuracy than the average of the single runs for a set of 25 real-world classification datasets. Later in time this same ensemble mechanism has been applied to a broad variety of large-scale datasets[58,59,86,101–104] with equally successful results. More traditional types of ensemble learning, such as Bagging or Boosting have also been used in GBML, specifically using Genetic Programming.[105,106]

Moreover, ensembles created using GBML and other bio-inspired methods have also been applied to nonstandard types of classification, such as ordinal classification,[100] where the classes of the problem have an intrinsic order between them and a multiclass problem can be decomposed in a series of two-class problems by joining together the instances of classes that are adjacent in this intrinsic order, and then generate predictions for the full problem by chaining several of these subpredictors. A different class of classification problem but that can be solved with a quite similar ensemble architecture are hierarchical classification problems,[107] where different predictor models are generated for different layers of the class hierarchy, and the final prediction is done, similarly to the above example, by chaining several of these predictors from top layer to bottom layer.

If we increase the level of communication, we find several examples of GBML methods that apply some of the classic paradigms of parallel GAs[37] such as the master–slave model[8,86] or the islands model.[108,109]

The performance of the master–slave model, in which the GA cycle is managed by a single (master) node, and the slave nodes only perform fitness computations for the individuals held at the master, relies on two assumptions to achieve good performance: (1) most of the computation time is concentrated in fitness computations. This has been shown to be the case for most GBML systems. In large-scale datasets (hundreds of variables or a few hundred thousand records), fitness computations easily pass 99% of the CPU time.[85] (2) The communication costs of sending the population to the slaves and receiving back the fitness values is low. This may not be always the case in GBML, as these approaches (especially those following the Michigan[110] or the IRL[97] approaches) tend to use rather large populations, forcing us to send rules to the evaluation slaves and collect the resulting fitness. This scheme also increments the sequential parts that cannot be parallelized, posing a thread to the overall speedup of the parallel implementation as a result of Ambdhal's law.[111]

The NAX system[86] presents an interesting variant of the master–slave model to minimize such communication cost. NAX is a parallel GBML system

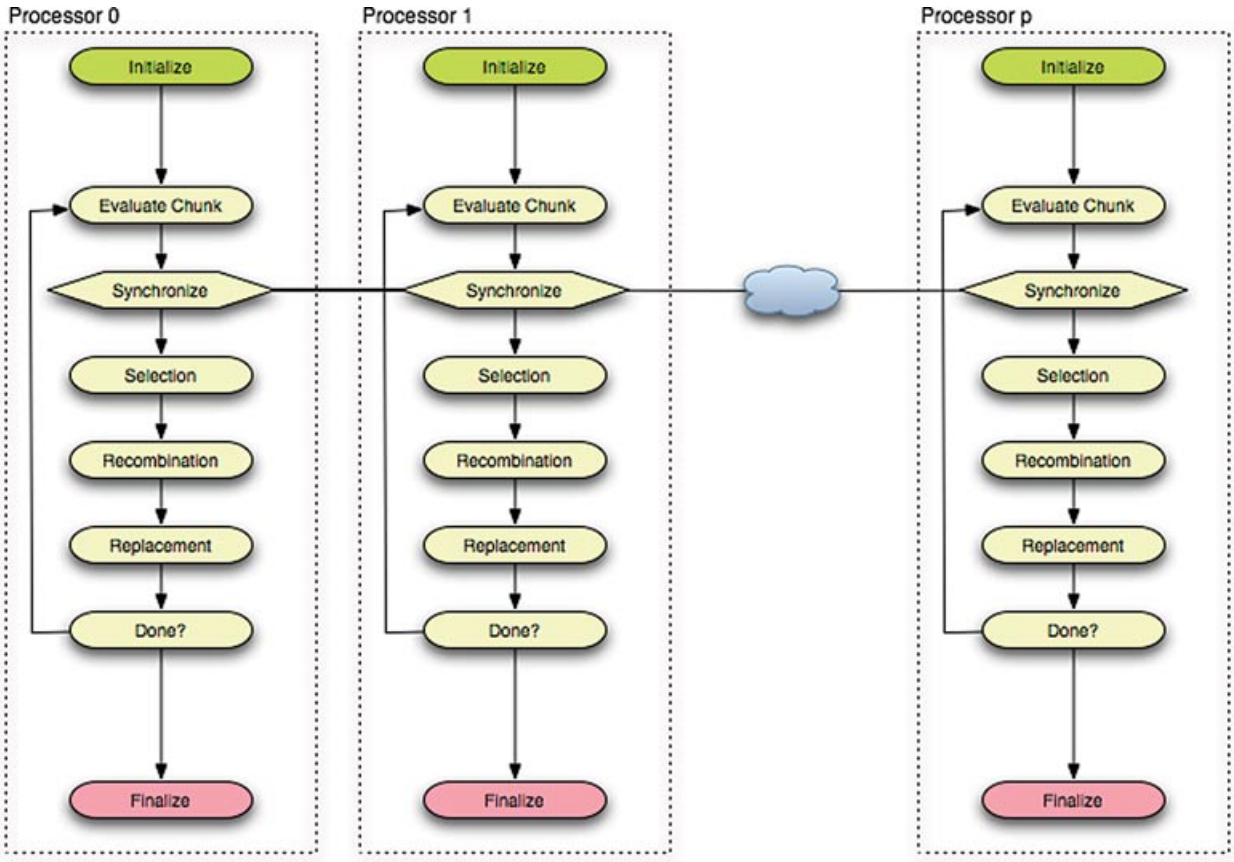

FIGURE 6 | Parallel architecture of the NAX system. © Natural Computing by Kluwer Academic Publishers. Reproduced with permission of Kluwer Academic Publishers in the format Journal via Copyright Clearance Center.

applying the IRL approach in which each processor runs an identical copy of the algorithm—all seeded in the same manner, and, hence performing the same genetic operations. They only differ in the portion of the population being evaluated. Thus, the population is treated as collection of chunks where each processor evaluates its own assigned chunk, sharing the fitness of the individuals in its chunk with the rest of processors. In this manner, fitness can be encapsulated and broadcasted, maximizing the occupation of the underlying packing frames used by the network infrastructure. Moreover, this approach also removes the need for sending the actual rules back and forth between processors—as a master–slave approach would require—thus, minimizing the communication to the bare minimum—namely, the fitness values. Figure 6 presents a conceptual scheme of the parallel architecture of NAX.

Fine-Grained Parallel Models

There are specific examples of GBML systems that contain fine-grained parallelism at the core of their design such as GALE (*Genetic and Artificial Life Environment*),[50] which integrates elements of the *Pittsburgh* approach and cellular automata models. GALE uses a two-dimensional grid board formed by $m \times n$ cells for spreading spatially the evolving population. Each cell of the grid contains either one or zero individuals. Each individual is a complete solution to the classification problem, following the Pittsburgh approach of GBML. Several knowledge representations (even mixed on the same board, but with restrictions) have been implemented within GALE: rule sets, decision trees, and sets of synthetic prototypes. Genetic operators are restricted to the immediate neighborhood of the cell in the grid. The size of the neighborhood is r. Given a cell and $r = 1$, the neighborhood of that cell is defined by the eight adjacent cells to it. Thus, r is the number of hops that defines the neighborhood. GALE's speedup model shows that when r equals 1, the speedup grows linearly with the number of processors used.[112]

The evolutionary model proposed by GALE is based on the local interactions among cells as described above. Each cycle in GALE has three main

operations: merge, split, and survive. Individuals can merge (given a certain probability) with one of its present neighbors, essential performing a crossover operation to generate a single off-spring with replaces the individual undergoing the merge. Afterward, the split operator clones and mutates the individual in the cell. The new mutated individual is placed in an empty cell from the neighborhood of the original individual. The survival stage decides whether the individual in an occupied cell will die (vacating the cell) or remain, depending on the number of existing neighbors and the individual's fitness. For further details about GALE model, please see Refs 50, 112, 113.

There are other examples of GBML models that, while they were never implemented as fine-grained parallelism, they have the potential to be easy transformed into such type of parallel architecture. One such example is Ref 114 which proposes a distributed LCS architecture designed to be used in very simple agents with limited memory. Each agent only stores a small part of a rule set, and multiple agents are chained to provide a complete solution to the classification problem. In such model good speedups would be obtained when all agents are continuously active, similarly to the pipeline architecture of modern CPUs.

DATA-INTENSIVE COMPUTING

When solving large-scale optimization or machine learning problems using EC techniques, researchers have realized that the population requirements may become infeasible if approached with traditional high-performance computing techniques.[115] Most models required maintaining the entire or at least a sample of large populations during the evolutionary process. However, the data abundance provided by such large populations have enabled data-intensive computing techniques to become a viable alternative parallelization scheme for EC techniques.[116–119] Moreover, such approaches also provide three key advantages when compared to their traditional high-performance computing counterparts:

(1) They do not require detailed knowledge of the underlying hardware architecture and their complex programming techniques, which are hard to debug.
(2) They do not require intensive check-pointing to tolerate failures quite common on large jobs than may run for days.
(3) They scale well on commodity clusters; usually the efficiency of traditional parallel programming frameworks (e.g., Message Passing Interface—MPI) relies on expensive high-quality interconnection networks.

The following section will focus on a specific case: the Hadoop[44] data-intensive framework, widely used to do large-scale data processing, and its application to parallelize the eCGA's model building process.[120]

Hadoop and the MapReduce Model

Hadoop[44] is Yahoo!'s open source MapReduce framework. Modeled after Google's MapReduce paper,[121] Hadoop builds on the *map* and *reduce* primitives present in functional languages. Hadoop relies on these two abstractions to enable the easily development of large-scale distributed applications as long as your application can be modeled around these two phases.

In this model, the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce framework then groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. The intermediate values are supplied to the user's Reduce function via an iterator. This allows the model to handle lists of values that are too large to fit in main memory.

Conceptually, the Map and Reduce functions supplied by the user have the following types:

$$\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2), \tag{2}$$

$$\text{reduce}(k_2, list(v_2)) \rightarrow \text{list}(v_3), \tag{3}$$

that is, the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values.

The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Reduce invocations are distributed by partitioning the

intermediate key space into R pieces using a partitioning function, which is $hash(key)\%R$ according to the default Hadoop configuration. The number of partitions (R) and the partitioning function are specified by the user. The overall execution is thus orchestrated in two steps: first all Mappers are executed in parallel, then the Reducers process the generated key value pairs by the Reducers. A detailed explanation of this framework is beyond the scope of this paper and can be found elsewhere.[122]

MapReducing eCGA

Ref 120 has presented a Hadoop-based implementation of eCGA's[69] model-building process using data-intensive computing. As mentioned above, eCGA is an EDA that models a problem's structure as a set of nonoverlapped BBs. The quality of a model is assessed by a fitness function based on the minimum description length (MDL) principle.[122] The model building process starts with one BB per problem variable that are iteratively merged (in a greedy fashion, merging two BBs at a time) while the merged BBs improve eCGA's MDL metric. The model building is an important step in eCGA and can become the bottleneck if implemented sequentially. However, it is also difficult to parallelize this step because of the interdependence of these steps. The solution is to split the population among different mappers. Each mapper computes the BB scores for the individuals it processes as well as the score of every possible two-way merge of BBs. Then a single reducer aggregates the scores to compute the global MDL metric of all candidate BB merges, picks the best merge and sends the merged partition to the mappers for the next iteration of model building until the MDL metric cannot be improved.

Other Data-Intensive Computing Frameworks

Other data-intensive computing frameworks have also been applied to parallelize EC methods.[118–120] MongoDB (http://www.mongodb.org/) is a scalable, high-performance, open source, schema-free, document-oriented database. Among others, MongoDB provides MapReduce tasks as a primitive of the query interface. When documents are stored in sharded collections (collections of documents broken in to shards distributed across different servers), MongoDB is able to run MapReduce tasks in parallel making it an appealing alternative to Hadoop. Moreover, Meandre[123] is the National Center for Supercomputing Applications's data-intensive computing infrastructure for science, engineering, and humanities. Meandre provides a more flexible programming model that allows to create complex data flows, which could be regarded as complex and possible iterating MapReduce stages. Dataflow execution engines provide a scheduler that determines the firing (execution) sequence of components. Meandre uses a *decentralized scheduling policy* designed to maximize the use of multicore architectures. Also groups of components can be placed across different machines and hence also scale by distributed execution. Meandre also allows works with processes that require directed cyclic graphs, thus extending beyond the traditional MapReduce directed acyclic graphs.

VISUALIZATION OF GBML RESULTS

Knowledge extraction is a crucial part of the data mining process, because in this context it is not enough to use GBML systems (and in general, any kind of machine learning) to generate just predictions. The end users of these methods expect to discover *new knowledge* in the data. Hence, understanding and visualizing the solutions produced by GBML methods is a very important topic. GBML is in a very good position in this context, as a very large majority of their systems are rule-based and generate solutions that are directly human-readable as a set of logic predicates. However, in most case it is not enough to just show rule sets. To generate a complete picture of the GBML solutions some kind of visualization is required.

A recent work using Michigan LCS systems[124] uses a technique inspired in the concept of heatmaps. A heatmap is a graphical visualization technique, frequently used to represent biological data, in which colors are used to represent the values in a matrix where rows and columns are the samples (instances) and genes/proteins (attributes), respectively. Rows and columns are sorted using a hierarchical clustering technique. The effect of this sorting is that patterns of colors can be easily identified in the heatmap which may lead to indicate the most important variables associated to a specific group of samples. In the application of heatmaps to represent rule sets, the rows and columns are the instances and attributes of the domain and the color of the heatmap for a particular cell indicates if the attribute of that cell is important for the prediction of that sample based on the rules generated by the Michigan LCS. By sorting the instances in the dataset properly, this technique becomes useful to identify salient interactions between attributes that are important to predict a large number of samples in the dataset.

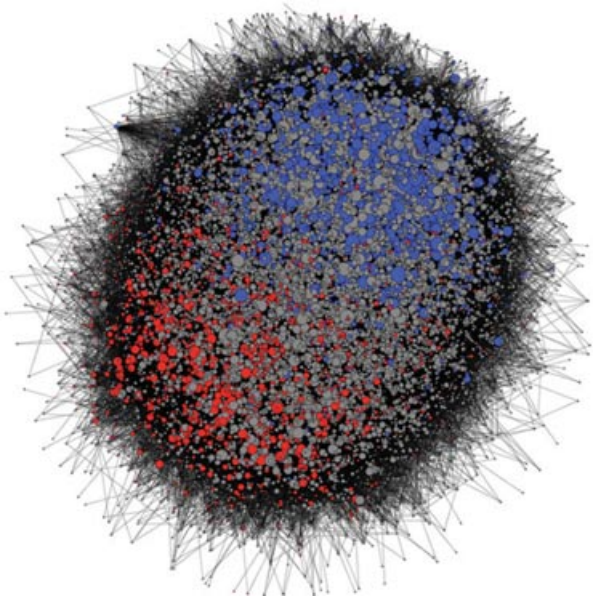

FIGURE 7 | Visualization of a large number of rule sets as a network of interactions.[104] Nodes = attributes. Edges = attributes appearing together in a rule.

The previous method was used to visualize a single rule set, the final population of a Michigan LCS. Other methods integrate multiple rule sets in a single visualization. In Ref 104, the BioHEL GBML system was used to generate rule sets from a microarray dataset about plant seed germination. This dataset had 119 samples and almost 14,000 attributes, and each time that BioHEL was run on this data (using different random seeds) it ended up producing a totally different rule set, although some attributes appeared in the rules far more frequently than others. This issue was exploited by running BioHEL 20,000 times (in this dataset a BioHEL run just takes a few seconds) to generate 20000 different rule sets. Rankings of important attributes and their interactions were generated by simply counting how many times (1) an attribute was used in a rule and (2) a pair of attributes appeared together in a rule. Finally, an interaction network of the dataset's attributes was generated by connecting together with an edge the attributes (nodes) that appeared together in the rules. This network is shown in Figure 7. The color of the nodes in the network is determined by some preexisting domain knowledge that had not been used in the training process. The regular distribution of the colors shows that the network (and hence the rule sets) have been able to reconstruct this domain knowledge from the data.

The previous two examples used visualization to represent the *content* of the rule sets. Other visualizations have been proposed to represent *the structure* of the rule sets. In *Parallelization Models*, we have mentioned an example of a (potentially) fine-grained LCS architecture where the prediction are performed by a distributed set of simple agents chained between them.[115] The authors of that work generated a series of visualizations of the patterns of connections of their rule set agents for a variety of problems that was very useful to understand the behavior of their distributed architecture.

Finally, there are also examples of visualizations of the solutions of GBML systems that do not evolve rule sets. In Ref 125, the authors proposed a method to evolve the topology of a hidden Markov model (HMM) as the core of a Fisher Kernel used for protein sequence classification tasks. The HMMs had a hierarchical structure divided in blocks. A GA was used to evolve the connections from block to block (or to itself), while an heuristic algorithm was used within each block to generate the low-level HMM subsolutions. In this specific case the visualization of the evolved solutions is simply the representation of the solution's phenotype: the diagram of connected blocks. Different block shapes were used to indicate different types of block.

THEORETICAL MODELS OF GBML

The previous sections have shown the enormous variety of GBML mechanisms that we can combine to improve the efficiency of GBML methods, but a few questions remain: how to choose the appropriate method? Moreover, what is the best way of using them? Recent years have seen the development of many theoretical models about GBML systems, either about the whole system or from one or more of the components of the method. Their motivation is not just for the sake of modeling the system, but as tools to adjust in principled ways the components/system they are modeling.

We have already mentioned above the success model of the ILAS windowing scheme,[55] which can be used, for the specific case of a Pittsburgh LCS, to estimate the maximum number of strata to use in ILAS without compromising the learning abilities of the system. Furthermore, a simple model of the the GAssist[56] system's initialization stage was proposed[14] to adjust automatically the main parameter controlling its initialization (p).

Michigan LCS are the systems that have been modelled in most detail. Butz[15] proposed several

models for the XCS[126] system covering all of its functioning: initialization, crossover, mutation and learning time. Moreover, these models were used to estimate bounds of the minimal population size required for the system to function correctly. Later in time, Orriols[16] created versions of these models specifically adapted to deal with a very important challenge in large-scale data mining: class imbalance. These models were able to inform how to modify the parameters in XCS to make sure that accurate rules were learnt for all classes of the problem including the minority ones.

Finally, recently Franco et al.[17] have proposed models for the initialization stage of the BioHEL[3] GBML system. These models are inspired in those developed for XCS but with three major changes: (1) using the ALKR and GABIL representation, instead of the ternary representation of XCS and (2) dealing with $\chi-ary$ variables instead of the binary variables of the original models and (3) explicitly modeling the concept of rule overlap, which is a known factor of difficulty for GBML systems.[18] These models were able to inform about how to adjust BioHEL's initialization parameters to ensure the creation of a good initial population.

REAL WORLD EXAMPLES

The literature is rich in applications of GBML methods (e.g., see Refs 127–130). In the specific case of large-scale datasets there are several examples of GBML methods applied to the well-known KDDCUP-99 large-scale benchmark[60,132–134] as well as applications to specific problems, mostly in the bioinformatics/biomedicine domains.[58,86,91,101–104,107,124]

In the following section, we are going to describe in detail a specific example as a case study: the application of the BioHEL[3] rule-based GBML system to protein contact map (CM) prediction.

Case Study: Contact Map Prediction Using GBML

PSP[134–135] is one of the most challenging problems in the field of computational biology. Proteins are crucial molecules for the functioning of all processes in living organisms. A protein is constructed as a chain of amino acids (AAs). As it is being constructed, this chain folds to create very complex three-dimensional shapes. The function of a protein if a consequence of its structure. Thus, knowing the structure of a protein is a crucial step for understanding its function. However, it is extremely difficult and costly to experimentally determine the structure of a protein. Given that proteins fold on their own as they are being constructed, the general consensus of the community is that the sequence of a protein should contain enough information to predict its structure. This assumption gave place to the PSP field. The impact of having better PSP methods would be enormous, not just for understanding life better, but also to design new/modified proteins for a variety of applications such as the production of better crops or more efficient drugs. This case study focuses on a specific subproblem of PSP: CM prediction. A CM is a binary matrix with as many rows and columns as AA in the protein, where each cell indicates whether a certain pair of AAs in a protein is in contact (their euclidean distance in the structure is less than a certain threshold). Reconstructing a CM is generally treated as a machine learning problem, and a very challenging one because of three reasons: very large training sets, high-dimensionality representations, and huge class imbalance.

For a protein of N AAs, we can have roughly N^2 possible contacts. A typical protein dataset tend to have around 2000 sequences of an average size around 200 AAs. This roughly means 80 millions contacts. Moreover, a CM is a very *sparse* matrix, and the number of real contacts generally is around 2% of the total number of possible pairs of AAs. Hence, this is a prediction problem presenting an extremely high class imbalance. Finally, the state-of-the-art representations for this problem tend to have hundreds of attributes (if not thousands). Most methods do not tend to use datasets as difficult as described above. Preprocessing techniques including thinning out the set of employed proteins,[136] rebalancing the positive and negative examples[137] or reducing the size of the alphabet[101] are sometimes employed to alleviate these challenges.

The system employed for this problem, BioHEL[3] is a GBML system applying the IRL[98] paradigm. In this paradigm of rule learning, the rule set presented as final solution of the learning process is learnt one rule at a time in a iterative way. Once a rule is learnt, the examples covered by it are discarded from the training set and the process is started again to learn the next rule. This iterative process ends when there all training examples have been covered. The fitness function of such systems has two goals: generate rules that are not only accurate but also general (covering as many examples as possible). It is not easy to find the correct balance between these two goals. In BioHEL this is achieved by redefining the concept of coverage as a

piecewise function that rewards rules that cover *at least* a certain, user-specified, percentage of examples. In this way, the system avoid learning accurate but overspecific rules, although its performance completely depends on setting up this parameter correctly citeFranco2012. Moreover, BioHEL was designed explicitly to cope from large-scale datasets, incorporating several of the efficiency enhancement mechanisms that we have described throughout this paper: the ILAS windowing scheme, the ALKR knowledge representation, a GPGPU-based fast fitness computation and ensembles for consensus prediction. For a complete description of BioHEL, please see Ref 3.

For the prediction of CM a training set of 2413 proteins representing a broad range of different structures was generated. In this dataset (after thinning out), there were over 32 million potential contacts, each of them represented using 631 attributes. The training set occupies 56.7 GB of disk space. In order to alleviate both the training set size and the class imbalance, 50 samples out of this huge dataset were selected. Each sample had 660,000 examples and a ratio of two noncontacts for each contact. Hence, rebalancing the training sets. Next, BioHEL was trained on each of the samples 25 times, each time using a different random seed. This created a total of 1250 different rule sets, one resulting from each of the 50 × 25 runs. These rule sets were integrated into an ensemble that was performing a simple majority voting as represented in Figure 8. The whole training process took around 25,000 CPU hours using Intel Xeon E5472 processors at 3.00 GHz. For a complete description of this CM prediction architecture, please see Ref 59. This prediction method participated in the CM category of the eighth and ninth editions of CASP (critical assessment of techniques for protein structure prediction), a biannual experiment held to assess the start of the art in PSP methods. The method participated in CASP8[138] as ID72 and in CASP9[139] as ID51. The results of the assessment showed that this method ranked very high according to most performance metrics. Particularly, in CASP9 this method had the highest rank among all sequence-based predictors in five out of the six evaluated metrics.[59]

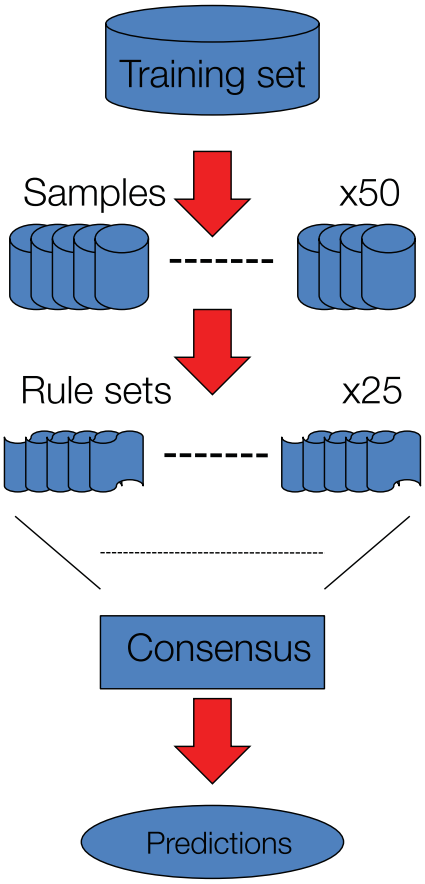

FIGURE 8 | Representation of the training and prediction process for contact map prediction using the BioHEL genetics-based machine learning system.

CLASSIFICATION AND ANALYSIS OF LARGE-SCALE GBML MECHANISMS

In this paper, we have described a very broad range of mechanisms that have been proposed to tackle the challenges of large-scale data mining, both from the point of view of efficiency, as it is natural, but also from the point of view of the quality of the generated solutions. That is, whether the mechanisms modify the learning capacity of the GBML systems. To summarize the myriad of mechanisms described in this paper, we propose a taxonomy based on answering these five questions:

Pos Does the method have a positive impact on the quality of the generated solutions?

Neg May the method suffer from a certain (moderate) degree of solution quality loss?

Enh Does the method improve efficiency by changing the algorithm?

HW Does the method improve efficiency by using special hardware?

Par Does the method improve efficiency by parallelizing the algorithm?

Table 1 shows the classification of all the methods described in this paper based on this taxonomy.

TABLE 1 | Classification of GBML Methods for Large-Scale Data Mining

Method	Pos	Neg	Enh	HW	Par
Windowing mechanisms[53–55,60]	y	y	y	n	n
Rule match precomputing[63,64]	n	n	y	n	n
Reordering attributes by specificity[65]	n	n	y	n	n
Attribute list knowledge representation[3,132]	y	n	y	n	n
Hybrid EDA–GBML methods[9,71,73]	y	n	y	n	n
Hybrid memetic-GBML methods[8,11,77–79]	y	n	y	n	n
Fitness surrogates[81,82]	n	y	y	n	n
Vectorial matching[85,86]	n	n	y	y	n
GPGPU matching[90–91,96]	n	n	n	y	n
Ensemble mechanisms[100,105,107]	y	n	n	n	y
Master–slave parallel models[86]	n	n	n	n	y
Island parallel models[108,109]	y	y	n	n	y
Fine-grained parallel models[50,114]	y	y	n	n	y
Data-intensive computing[116–120,123]	n	n	n	n	y

As we can see in Table 1 there are methods that produce an impact in more than one of the categories of the taxonomy while others only affect a single category. Particularly interesting are the three cases (windowing mechanisms, island models, and fine-grained parallel models) that can both affect positively and negatively the learning process of the GBML systems. In these cases, adjusting the parameters of the mechanisms (e.g., the number of strata in ILAS, or the number of agents in the island/fine-grained models) is crucial to avoid constraining the learning process.

Moreover, many of these mechanisms can be successfully combined among them to accumulate their individual benefits. We have shown in the case study on CM prediction how in the BioHEL system we combine the ILAS windowing scheme, the ALKR representation, GPGPU matching and ensemble mechanisms. Some of these mechanisms can be easily combined with any other, while some cannot. For instance, the ensemble methods can be used on top of any other mechanism because they combine the models extracted from several independent runs of the system, so there is no explicit coupling between mechanisms.

The windowing mechanisms are a very delicate case. Given that they modify how the system learns (because essentially they create a dynamic optimization environment by changing the instances used to compute the fitness calculations at every GA iteration), the methods that use the content of the instances to perform their work may not be suitable for combining with it. For instance, it would be almost impossible to combine the windowing with the match precomputing algorithms/fitness surrogates, as the benefit of these techniques would mostly be lost if the precomputing needs to happen at each iteration instead of once for the whole run. The specific case of MPLCS showed how windowing can be combined successfully with memetic operators, but only when the number of strata in ILAS is correctly adjusted. When the strata stop being good representatives of the whole training set, the memetic operators may start to overfit the rules to the current stratum.

As we have discussed above, GPGPUs can be combined with other efficiency enhancement techniques such as the ALKR knowledge representation, but only when the right degree of parallelism is used. That degree of parallelism roughly speaking is that that achieves optimal memory access patterns and an utilization of the GPU computational resources close to maximal. Moreover, some algorithms, like the memetic operators, are essentially serial, so it is very difficult to reimplement them with a parallel approach. Furthermore, the binary/SSE encoding technique for vectorial matching probably would not be efficient in combination with the ALKR representation because of the sparseness of that representation. The vectorial matching requires rules to be tightly packed and have uniform structure in order to quickly perform the match of multiple attributes in a single logical operation. When the rules stop having a uniform structure or the packing needs to be edited too often, the efficiency of these mechanisms quickly drops.

Finally, we have to keep in mind the suitability of these mechanisms to the type of problems being tackled. For instance, data-intensive computing and to a lesser extent the GPGPU matching can only be

applied if the volume of data in the training set is large enough. The trade-off between the costs of performing an individual's evaluation and the communication required for such evaluation is important to decide between traditional master–slave models or adapted versions such as NAX. Most hybrid methods are only able to significantly outperform traditional genetic search on problems of high enough dimensionality.

CONCLUSION

This paper has reviewed the broad range of methods that have been applied to improve/adapt GBML techniques to cope with large-scale datasets. These methods build upon the natural parallel capacity of EC methods, but are not limited to only the traditional efficiency enhancement techniques of EC (e.g., fitness surrogates, parallel GAs, hybrid exploration, and so on). GBML methods have specific scaling up mechanisms that span representations, learning paradigms, inference mechanisms but also parallel/special hardware implementations. Moreover, theoretical models have been proposed in recent years about a variety of aspects of a GBML system that can help adjust them in a principled way, which is crucial in the context of large-scale data mining because in most cases it is not possible to afford a full parameter sweep. The paper has shown how these techniques are prepared to deal with extremely difficult real-world problems of high impact for society, and that they have quickly adapted to embrace the latest advances of high-performance computing hardware such as GPGPUs.

The appearance in the last one/two years of buzz words such as 'big data' or 'data science', especially in industry, indicates that the road ahead of us can provide many benefits for the GBML methods and community, if they solve some of their limitations. We have seen how in many cases the good performance of GBML scaling up mechanisms depends on the careful choice of parameters. This is a drawback for a wide dissemination of these methods, as practitioners that are not GBML experts may opt for other, simpler to set up, techniques. To solve this problem, it is crucial to develop automatic parameter setting heuristics, and the theoretical models of all aspects of a GBML system are very important to design principled adjusting mechanisms for all these parameters. We have shown in this paper that many theoretical models already exist, but many of them are limited to simple cases (e.g., binary representations), so there is still a lot of work to do in this direction. Another aspect that still has a large potential for improvement are the data-intensive computing implementations. The example that we have illustrated in this paper is the core eCGA optimization algorithm, hence not exclusively designed for machine learning tasks. GBML methods cover a range of problems relevant enough that it is worth and necessary to propose specific, for example, Hadoop implementations of GBML mechanisms. Moreover, there are very sophisticated knowledge representations in GBML (e.g., hyperellipsoid conditions[140]) that are not ready yet to cope with high dimensionality domains, so it may be possible to extend them using some of the strategies that we have discussed throughout this manuscript. Finally, a very crucial aspect for the future is dissemination, making sure that GBML methods are known to a very broad audience of potential practitioners. A possible strategy to achieve this aim may be to integrate implementations of GBML methods in popular machine learning packages such as WEKA,[141] KEEL[142] (a machine learning platform for GBML methods), or Mahout[143] (the machine learning extension of Hadoop).

REFERENCES

1. Kovacs T. Genetics-based machine learning. In: Rozenberg G, Bäck T, Kok J, eds. *Handbook of Natural Computing: Theory, Experiments, and Applications*. Berlin: Springer Verlag; 2011.
2. Fernández A, Garciá S, Luengo J, Bernadó-Mansilla E, Herrera F. Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study. *IEEE Trans Evolut Comput* 2010, 14:913–941.
3. Bacardit J, Burke EK, Krasnogor N. Improving the scalability of rule-based evolutionary learning. *Memetic Comput* 2009, 1:55–67.
4. Lanzi PL, Rocca S, Solari S. An approach to analyze the evolution of symbolic conditions in learning classifier systems. In: *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2007, 2795–2800.
5. Browne WN, Ioannides C. Investigating scaling of an abstracted LCS utilising ternary and s-expression alphabets. In: *GECCO '07: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*. ACM Press, New York; 2007, 2759–2764.

6. Llorà X, Priya A, Bhargava R. Observer-invariant histopathology using genetics-based machine learning. *Nat Comput* 2008, 8:101–120.
7. Butz M, Lanzi P, Wilson S. Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Trans Evolut Comput* 2008, 12:355–376.
8. Bacardit J, Krasnogor N. Performance and efficiency of memetic pittsburgh learning classifier systems. *Evolut Comput J* 2009, 17:307–342.
9. Butz MV, Pelikan M, Llorà X, Goldberg DE. Automated global structure extraction for effective local building block processing in XCS. *Evolut Comput* 2006, 14:345–380.
10. Llorà X, Sastry K, Lima CF, Lobo FG, Goldberg DE. Linkage learning, rule representation, and the x-ary extended compact classifier system. In: Bacardit J, Bernadó-Mansilla E, Butz MV, Kovacs T, Llorà X, Takadama K, eds. Learning Classifier Systems, Lecture Notes In Artificial Intelligence, Vol. 4998. Berlin, Heidelberg: Springer-Verlag; 2008, 189–205.
11. Wyatt D, Bull L. A memetic learning classifier system for describing continuous-valued problem spaces. In: *Recent Advances in Memetic Algorithms*. Berlin: Springer; 2004, 355–396.
12. Loiacono D, Marelli A, Lanzi PL. Support vector regression for classifier prediction. In: *GECCO '07: Proceedings of the 9th Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2007, 1806–1813.
13. Tamee K, Bull L, Pinngern O. Towards clustering with XCS. In: *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2007, 1854–1860.
14. Bacardit J. Analysis of the initialization stage of a Pittsburgh approach learning classifier system. In: *GECCO '05: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2005, 1843–1850.
15. Butz MV. Rule-Based Evolutionary online learning systems: a principled approach to lcs analysis and design. In: *Studies in Fuzziness and Soft Computing*. Berlin: Springer; 2006.
16. Orriols-Puig A. *New Challenges in Learning Classifier Systems: Mining Rarities and Evolving Fuzzy Models*. PhD Thesis. Barcelona, Spain: Ramon Llull University; 2008.
17. Franco MA, Krasnogor N, Bacardit J. Modelling the initialisation stage of the ALKR representation for discrete domains and GABIL encoding. In: *GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York: 2011.
18. Ioannides C, Barrett G, Eder K. XCS cannot learn all boolean functions. In: *GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York: 2011, 1283–1290.
19. Bernadó-Mansilla E, Llorà X, Garrell JM. XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In: IWLCS 2001, LNAI 2321, Berlin: Springer Verlag; 2002, 115–132.
20. Bacardit J, Butz MV. Data mining in learning classifier systems: comparing XCS with GAssist. In: Kovacs T, Llorà X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW, eds. *Advances at the Frontier of Learning Classifier Systems*. Berlin: Springer-Verlag; 2007, 282–290.
21. Orriols-Puig A, Casillas J, Bernadó-Mansilla E. Genetic-based machine learning systems are competitive for pattern recognition. *Evolut Intell* 2008, 1:209–232.
22. Genbank release notes. Available at: ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt. (Accessed November 21, 2012).
23. Physicists brace themselves for LHC 'data avalanche'. Available at: http://www.nature.com/news/2008/080722/full/news.2008.967.html. (Accessed November 21, 2012).
24. Pop M, Salzberg SL. Bioinformatics challenges of new sequencing technology. *Trends Genet* 2008, 24:142–149.
25. The netflix prize. Available at: http://www.netflixprize.com. (Accessed November 21, 2012).
26. The netflix prize leaderboard. Available at: http://www.netflixprize.com/leaderboard. (Accessed November 21, 2012).
27. Allison DB, Page GP, Beasley TM, Edwards JW, eds. *DNA Microarrays and Related Genomics Techniques: Design, Analysis, and Interpretation of Experiments*. Boca Raton, FL: Chapman & Hall; 2005.
28. Glaab E, Garibaldi J, Krasnogor N. Arraymining: a modular web-application for microarray analysis combining ensemble and consensus methods with cross-study normalization. *BMC Bioinformatics* 2009, 10:358.
29. The ICOS PSP benchmarks repository. Available at: http://icos.cs.nott.ac.uk/datasets/psp_benchmark.html. (Accessed November 21, 2012).
30. Reuters-21578 text categorization collection. Available at: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html. (Accessed November 21, 2012).
31. Otero F, Freitas A, Johnson C. A hierarchical multilabel classification ant colony algorithm for protein function prediction. *Memetic Comput* 2010, 2:165–181.
32. Japkowicz N, Stephen S. The class imbalance problem: a systematic study. *Intelli Data Anal* 2002, 6:429–450.

33. Ho TK, Basu M. Complexity measures of supervised classification problems. *IEEE Trans Pattern Anal Mach Intell* 2002, 24:289–300.
34. Bernadó-Mansilla E, Ho TK. Domain of competence of XCS classifier system in complexity measurement space. *IEEE Trans Evolut Comput* 2005, 9:82–104.
35. Luengo J, Fernández A, García S, Herrera F. Addressing data complexity for imbalanced data sets: analysis of SMOTE-based oversampling and evolutionary undersampling. *Soft Comput* 2010, 15:1909–1936.
36. Dumbill E. What is big data?, 2012. Available at: http://radar.oreilly.com/2012/01/what-is-big-data.html.
37. Cantú-Paz E. *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer Academic; 2000.
38. Alba E. *Parallel Metaheuristics: A New Class of Algorithms*. Hoboken, New Jersey: Wiley Interscience; 2005.
39. The HDF5 data storage system. Available at: http://www.hdfgroup.org/HDF5/. (Accessed November 21, 2012).
40. The hadoop distributed file system. Available at: http://hadoop.apache.org/docs/hdfs/current/hdfs_design.html. (Accessed November 21, 2012).
41. The mongodb document-oriented database. Available at: http://www.mongodb.org/. (Accessed November 21, 2012).
42. Sastry K. Principled efficiency enhancement techniques. In: *GECCO '05: Proceedings of the 7th annual conference companion on Genetic and evolutionary computation*; 2005. Available at: http://www.illigal.uiuc.edu/web/kumara/2005/11/24/principled-efficiency-enhancement-techniques/. (Accessed November 21, 2012).
43. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. In: *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*. USENIX, Berkeley, CA; 2004, 137–150.
44. The hadoop distributed computing framework. Available at: http://hadoop.apache.org/. (Accessed November 21, 2012).
45. Fürnkranz J. Integrative windowing. *J Artif Intell Res* 1998, 8:129–164.
46. Quinlan JR. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann; 1993.
47. John GH, Langley P. Static versus dynamic sampling for data mining. In: Simoudis E, Han J, Fayyad UM, eds. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA; 1996, 367–370.
48. Maloof MA, Michalski RS. Selecting examples for partial memory learning. *Mach Learn* 2000, 41:27–52.
49. Freitas AA. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Berlin: Springer-Verlag; 2002.
50. Llorà X. *Genetics-Based Machine Learning using Fine-grained Parallelism for Data Mining*. PhD Thesis. Barcelona, Spain: Enginyeria i Arquitectura La Salle, Ramon Llull University; 2002.
51. Derrac J, García S, Herrera F. Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability. *Memetic Comput* 2010, 2:183–199.
52. García S, Derrac J, Cano J, Herrera F. Prototype selection for nearest neighbor classification: taxonomy and empirical study. *IEEE Trans Pattern Anal Mach Intell* 2012, 34:417–435.
53. Sharpe PK, Glover RP. Efficient GA based techniques for classification. *Appl Intell* 1999, 11:277–284.
54. Gathercole C, Ross P. Dynamic training subset selection for supervised learning in genetic programming. In: Davidor Y, Schwefel H-P, Männer R. eds. *Parallel Problem Solving from Nature III*. Jerusalem: Springer-Verlag; 1994, 312–321.
55. Bacardit J, Goldberg D, Butz M, Llorà X, Garrell JM. Speeding-up pittsburgh learning classifier systems: modeling time and accuracy. In: Yao X, Burke EK, Lozano JA, Smith J, Merelo-Guervós JJ, Bullinaria JA, Rowe JE, Tiňo P, Kabán A, Schwefel H-P, eds. *Parallel Problem Solving from Nature—PPSN 2004*. LNCS. Berlin: Springer-Verlag; 2004, 1021–1031.
56. Bacardit J. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, Generalization, and Run-Time*. PhD Thesis. Barcelona, Spain: Ramon Llull University; 2004.
57. Bacardit J, Stout M, Krasnogor N, Hirst JD, Blazewicz J. Coordination number prediction using learning classifier systems: performance and interpretability. In: *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2006, 247–254.
58. Stout M, Bacardit J, Hirst JD, Krasnogor N. Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics* 2008, 24:916–923.
59. Bacardit J, Widera P, Márquez-Chamorro A, Divina F, Aguilar-Ruiz JS, Krasnogor N. Contact map prediction using a large-scaleensemble of rule sets and the fusion of multiple predicted structural features. *Bioinformatics*. 2012, 28:2441–2448.
60. Song D, Heywood MI, Zincir-Heywood AN. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans Evolut Comput* 2005, 9:225–239.
61. Frank A, Asuncion A. *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science; 2010. Available

at: http://archive.ics.uci.edu/ml. (Accessed November 21, 2012).

62. Cano J-R, García S, Herrera F. Subgroup discover in large size data sets preprocessed using stratified instance selection for increasing the presence of minority classes. *Pattern Recognit Lett* 2008, 29:2156–2164.
63. Giráldez R, Aguilar-Ruiz JS, Santos JCR. Knowledge-based fast evaluation for evolutionary learning. *IEEE Trans Syst Man Cybern C* 2005, 35:254–261.
64. Mellor D., Nicklin SP. A population-based approach to finding the matchset of a learning classifier system efficiently. In: *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2009, 1267–1274.
65. Butz MV, Lanzi PL, Llorà X, Loiacono D. An analysis of matching in learning classifier systems. In: *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2008, 1349–1356.
66. Langdon WB. Fitness causes bloat in variable size representations. Technical Report CSRP-97-14. Birmingham, UK: School of Computer Science, University of Birmingham; 1997. (Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97.)
67. Larrañaga P, Lozano J, eds. *Estimation of Distribution Algorithms, A New Tool for Evolutionnary Computation*. Genetic Algorithms and Evolutionnary Computation. Norwell, MA: Kluwer Academic Publishers; 2002.
68. Krasnogor N, Smith J. A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Trans Evolut Comput* 2005, 9:474–488.
69. Harik G. Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010. Urbana and Champaign, IL: Illinois Genetic Algorithms Lab, University of Illinois at Urbana-Champaign; 1999.
70. Pelikan M, Goldberg DE, Cantú-Paz E. BOA: the Bayesian optimization algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Morgan Kaufmann, San Mateo, CA; 1999, 525–532.
71. Llorà X, Sastry K, Goldberg DE. The compact classifier system: Scalability analysis and first results. In: *Proceedings of the Congress on Evolutionary Computation 2005*. New York: IEEE Press, 2005, 1:596–603.
72. Harik G, Lobo F, Goldberg D. The compact genetic algorithm. *IEEE Trans Evolut Comput* 1999, 3:287–297.
73. Llorà X, Sastry K, Lima C, Lobo F, Goldberg DE. Linkage learning, rule representation, and the X-ary extended compact classifier system. In: Bacardit J, Bernadó-Mansilla E, Butz M, Kovacs T, Llorà X, Takadama K, eds. *Learning Classifier Systems. Lecture Notes in Computer Science*. Berlin/Heidelberg: Springer; 2008, 189–205.
74. Harik GR. Finding multimodal solutions using restricted tournament selection. In: Eshelman L, ed. *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, CA; 1995, 24–31.
75. Sierra B, Lazkano E, Inza I, Merino M, Larrañaga P, Quiroga J. Prototype selection and feature subset selection by estimation of distribution algorithms. A case study in the survival of cirrhotic patients treated with TIPS. *Lecture Notes in Computer Science*. Berlin/Heidelberg: Springer; 2001, 20–29.
76. Abegaz T, Dozier G, Bryant K, Adams J, Shelton J, Ricanek K, Woodard D. SSGA amp; EDA based feature selection and weighting for face recognition. In: New York: *IEEE Congress on Evolutionary Computation (CEC) 2011. IEEE*; 2011, 1375 –1381.
77. Grefenstette JJ. Lamarckian learning in multi-agent environments. In: Belew R, Booker L, eds. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufman, San Francisco, CA; 1991, 303–310.
78. Casillas J, Martínez P, Benítez AD. Learning consistent, complete and compact sets of fuzzy rules in conjunctive normal form for regression problems. *Soft Comput* 2008, 13:451–465.
79. Butz MV, Goldberg DE, Lanzi PL. Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. *IEEE Trans Evolut Comput* 2005, 9:452–473.
80. Miller BL, Goldberg DE. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolut Comput* 1996, 4:113–131.
81. Llorà X, Sastry K, Yu T-L, Goldberg DE. Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2007, 1798–1805.
82. Orriols-Puig A, Bernadó-Mansilla E, Sastry K, Goldberg DE. Substructrual surrogates for learning decomposable classification problems: implementation and first results. In: *GECCO '07: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*. ACM, New York; 2007, 2875–2882.
83. Sastry K, Lima CF, Goldberg DE. Evaluation relaxation using substructural information and linear estimation. In: *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2006, 419–426.
84. Yu T-L, Goldberg DE, Yassine A, Chen Y-P. Genetic algorithm design inspired by organizational theory: pilot study of a dependency structure matrix driven

genetic algorithm. In: *GECCO '03: Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2003, 1620–1621.

85. Llorà X, Sastry K. Fast rule matching for learning classifier systems via vector instructions. In: *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2006, 1513–1520.
86. Llorà X, Priya A, Bhargava R. Observer-invariant histopathology using genetics-based machine learning. *Natural Comput* 2009, 8:101–120.
87. NVIDIA. NVIDIA CUDA programming guide 2.0, 2008.
88. Maitre O, Baumes LA, Lachiche N, Corma A, Collet P. Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. ACM, Montreal; 2009, 1403–1410.
89. Langdon WB, Harrison AP. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Comput* 2008, 12:1169–1183.
90. Lanzi PL, Loiacono D. Speeding up matching in learning classifier systems using CUDA. In: Bacardit J, Browne W, Drugowitsch J, Bernadó-Mansilla E, Butz M, eds. *Learning Classifier Systems. Lecture Notes in Computer Science*. Berlin/Heidelberg: Springer; 2010, 1–20.
91. Langdon WB. Large scale bioinformatics data mining with parallel genetic programming on graphics processing units. In: de Vega F, Cantú-Paz E, eds. *Parallel and Distributed Computational Intelligence*. Studies in Computational Intelligence. Berlin/Heidelberg: Springer; 2010, 113–141.
92. Prabhu R. SOMGPU: an unsupervised pattern classifier on graphical processing unit. In: *IEEE Congress on Evolutionary Computation*. New York: IEEE; 2008, 1011–1018.
93. Sharp T. Implementing decision trees and forests on a GPU. In: *Computer Vision—ECCV 2008*. Berlin: Springer; 2008, 595–608.
94. Steinkraus D, Buck I, Simard P. Using GPUs for machine learning algorithms. In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on* IEEE, New York; 2005. 2:1115–1120.
95. Catanzaro B, Sundaram N, Keutzer K. Fast support vector machine training and classification on graphics processors. In: *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*. 2008, 111.
96. Franco MA, Krasnogor N, Bacardit J. Speeding up the evaluation of evolutionary learning systems using GPGPUs. In: *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2010, 1039–1046.
97. Venturini G. SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In: Brazdil PB, ed. *Machine Learning: ECML-93—Proceedings of the European Conference on Machine Learning*. Springer-Verlag: Berlin, 1993, 280–296.
98. NVIDIA. NVIDIA CUDA SDK—Data-Parallel algorithms. Available at: http://www.nvidia.com/content/cudazone/cuda_sdk/Data-Parallel_Algorithms.html#reduction. (Accessed November 21, 2012).
99. various authors. Special issue on integrating multiple learned models. *Mach Learn* 1999, 36:5–139.
100. Bacardit J, Krasnogor N. Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In: *Learning Classifier Systems, Revised Selected Papers of IWLCS 2006-2007*. LNAI. Berlin: Springer-Verlag; 2008, 255–268.
101. Bacardit J, Stout M, Hirst JD, Valencia A, Smith RE, Krasnogor N. Automated alphabet reduction for protein datasets. *BMC Bioinformatics*, 2009, 10:6.
102. Lee MC, Boroczky L, Sungur-Stasik K, Cann AD, Borczuk AC, Kawut SM, Powell CA. Computer-aided diagnosis of pulmonary nodules using a two-step approach for feature selection and classifier ensemble construction. *Artif Intell Med* 2010, 50:43–53.
103. Armananzas R, Inza I, Santana R, Saeys Y, Flores J, Lozano J, Peer Y, Blanco R, Robles V, Bielza C, Larrañaga. P. A review of estimation of distribution algorithms in bioinformatics. *BioData Min* 2008, 1:6.
104. Bassel GW, Glaab E, Marquez J, Holdsworth MJ, Bacardit J. Functional network construction in arabidopsis using rule-based machine learning on large-scale data sets. *Plant Cell* 2011, 23:3101–3116.
105. Zhang Y, Bhattacharyya S. Genetic programming in classifying large-scale data: an ensemble method. *Inf Sci* 2004, 163:85–101.
106. Folino G, Pizzuti C, Spezzano G. GP ensembles for large-scale data classification. *IEEE Trans Evolut Comput* 2006, 10:604–616.
107. Holden N, Freitas A. Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Comput* 2009, 13:259–272.
108. Dam HH, Abbass HA, Lokan C. DXCS: an XCS system for distributed data mining. In *GECCO '05: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2005, 1883–1890.
109. Bull L, Studley M, Bagnall A, Whittley I. Learning classifier system ensembles with rule-sharing. *IEEE Trans Evolut Comput* 2007, 11:496–502.
110. Holland JH, Reitman JS. Cognitive systems based on adaptive algorithms. In: Hayes-Roth D, Waterman F,

eds. *Pattern-directed Inference Systems*. New York: Academic Press; 1978, 313–329.

111. Amdahl G. Validity of the single processor approach to achieving large-scale computing capabilities. In: *AFIPS Conference Proceedings*. ACM, New York; 1967, 483–485.
112. Llorà X, Garrell J. Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: *GECCO '01: Proceedings of the 3th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann San Mateo, CA; 2001, 461–468.
113. Llorà X, and Garrell J. Evolving partially-defined instances with evolutionary algorithms. In *Proceedings of the 18th International Conference on Machine Learning (ICML'2001)*. Morgan Kaufmann Publishers, San Mateo, CA; 2001, 337–344.
114. Scheidler A., Middendorf M. Learning classifier systems to evolve classification rules for systems of memory constrained components. *Evolut Intell* 2011, 4:127–143.
115. Sastry K, Goldberg DE, Llorà X. Towards billion-bit optimization via a parallel estimation of distribution algorithm. In: *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2007, 577–584.
116. Jin C, Vecchiola C, Buyya R. MRPGA: An extension of mapreduce for parellelizing genetic algorithms. In: Press I, ed. *IEEE Fouth International Conference on eScience 2008*. IEEE: New York; 2008, 214–221.
117. Verma A, Llorà X, Goldberg DE, Campbell RH. Scaling genetic algorithms using mapreduce. In: *ISDA '09: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications*. Washington, D.C.: IEEE Computer Society; 2009, 13–18.
118. Llorà X. Data-intensive computing for competent genetic algorithms: a pilot study using meandre. In: *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York; 2009, 1387–1394.
119. Llorà X, Verma A, Campbell RH, Goldberg DE. When huge is routine: scaling genetic algorithms and estimation of distribution algorithms via data-intensive computing. In: Fernández de Vega F, Cantú-Paz E, eds. *Parallel and Distributed Computational Intelligence*. Berlin Heidelberg: Springer-Verlag; 2010, 1141.
120. Verma A, Llorà X, Venkataraman S, Goldberg DE, Campbell RH. Scaling eCGA model building via data-intensive computing. In: *IEEE Congress on Evolutionary Computation'10*. New York: IEEE; 2010, 1–8.
121. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. 2004.
122. Rissanen J. Modeling by shortest data description. *Automatica* 1978, 14:465–471.
123. Llorà X, Ács B, Auvil L, Capitanu B, Welge M, Goldberg DE. Meandre: semantic-driven data-intensive flows in the clouds. In: *Proceedings of the 4th IEEE International Conference on e-Science*. IEEE, New York; 2008, 238–245.
124. Urbanowicz RJ, Granizo-MacKenzie A, Moore JH. Instance-linked attribute tracking and feedback for michigan-style supervised learning classifier systems. In: *GECCO '12: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. ACM Press, New York; 2012, 927–934.
125. Won KJ, Saunders C, Prügel-Bennett A. Evolving fisher kernels for biological sequence classification. *Evolut Comput*. Available at: http://www.mitpressjournals.org/doi/abs/10.1162/EVCO_a_00065?journalCode=evco. (Accessed December 19, 2011).
126. Wilson S.W. Classifier fitness based on accuracy. *Evolut Comput* 1995, 3:149–175.
127. Bull L, ed. *Applications of Learning Classifier Systems*. Berlin: Springer-Verlag; 2004.
128. Ghosh A, Jain LC, eds. *Evolutionary Computation in Data Mining*. Berlin: Springer-Verlag; 2005.
129. Bull L, Bernadó-Mansilla E, Holmes J, eds. *Learning Classifier Systems in Data Mining*. Berlin: Springer-Verlag; 2008.
130. del Jesús MJ, Gámez JA, Puerta JM. Special issue on evolutionary and metaheuristics based data mining (EMBDM). *Soft Comput* 2009, 13:209–318.
131. Shafi K, Abbass HA. An adaptive genetic-based signature learning system for intrusion detection. *Expert Syst Appl* 2009, 36:12036–12043.
132. Bacardit J, Krasnogor N. A mixed discrete-continuous attribute list representation for large scale classification domains. In: *GECCO '11: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2009, 1155–1162.
133. Marín-Blázquez JG, Martínez Pérez G. Intrusion detection using a linguistic hedged fuzzy-XCS classifier system. *Soft Comput* 2009, 13:273–290.
134. Lesk AM. *Introduction to Protein Structure*. Oxford: Oxford University Press; 2001.
135. Moult J, Fidelis K, Kryshtafovych A, Rost B, Tramontano A. Critical assessment of methods of protein structure prediction. Round VIII. *Proteins* 2009, 77:1–4.
136. Baldi P, Pollastri G. The principled design of large-scale recursive neural network architectures DAG-RNNS and the protein structure prediction problem. *J Mach Learn Res* 2003, 4:575–602.

137. Punta M, Rost B. PROFcon: novel prediction of long-range contacts. *Bioinformatics* 2005, 21:2960–2968.
138. Ezkurdia I, Graña O, Izarzugaza JMG, Tress ML. Assessment of domain boundary predictions and the prediction of intramolecular contacts in CASP8. *Proteins* 2009, 77:196–209.
139. Monastyrskyy B, Fidelis K, Tramontano A, Kryshtafovych A. Evaluation of residue-residue contact predictions in CASP9. *Proteins* 2011, 79:119–125.
140. Butz MV, Lanzi PL, Wilson SW. Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In: *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York; 2006, 1457–1464.
141. Witten IH, Frank E. *Data Mining: Practical Machine Learning Tools and Techniques*. San Mateo, CA: Morgan Kaufmann; 2005.
142. Alcala Fdez J, Sánchez L, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera. F. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Comput* 2009, 13:307–318.
143. Mahout: Scalable machine learning and data mining. Available at: http://mahout.apache.org/. (Accessed November 21, 2012).

