Prog Artif Intell (2012) 1:71–87
DOI 10.1007/s13748-011-0004-4

REVIEW

Scaling up data mining algorithms: review and taxonomy

Nicolás García-Pedrajas · Aida de Haro-García

Received: 4 June 2011 / Accepted: 26 September 2011 / Published online: 13 January 2012

Abstract The overwhelming amount of data that are now available in any field of research poses new problems for data mining and knowledge discovery methods. Due to this huge amount of data, most of the current data mining algorithms are inapplicable to many real-world problems. Data mining algorithms become ineffective when the problem size becomes very large. In many cases, the demands of the algorithm in terms of the running time are very large, and mining methods cannot be applied when the problem grows. This aspect is closely related to the time complexity of the method. A second problem is linked with performance; although the method might be applicable, the size of the search space prevents an efficient execution, and the resulting solutions are unsatisfactory. Two approaches have been used to deal with this problem: scaling up data mining algorithms and data reduction. However, because data reduction is a data mining task itself, this technique also suffers from scalability problems. Thus, for many problems, especially when dealing with very large datasets, the only way to deal with the aforementioned problems is to scale up the data mining algorithm. Many efforts have been made to obtain methods that can be used to scale up existing data mining algorithms. In this paper, we review the methods that have been used to address the problem of scalability. We focus on general ideas, rather than specific implementations, that can be used to provide a general view of the current approaches for scaling up data mining methods. A taxonomy of the algorithms is proposed, and many examples of different tasks are presented. Among the different techniques used for data mining, we will pay special attention to evolutionary methods, because these methods have been used very successfully in many data mining tasks.

N. García-Pedrajas (✉) · A. de Haro-García
Computational Intelligence and Bioinformatics Research Group, University of Córdoba, Córdoba, Spain
e-mail: npedrajas@uco.es

A. de Haro-García
e-mail: adeharo@uco.es

Keywords Data mining · Scaling-up · Parallel algorithms · Very large datasets

1 Introduction

The question of how to solve large problems is ubiquitous in almost any machine learning task [30]. We have classification problems, where the number of instances and features is very large in many areas of research, such as in bioinformatics [12,137], text mining [85,129] and security [18,86]; optimization problems, which often include large-scale problems that are usually found in different areas such as scheduling [89], inverse problems chemical kinetics [75] and gene structure prediction [16]; and other problems in almost any field in which machine learning methods are applied.

Many of the widely used methods in data mining were developed when the common dataset size was on the order of hundreds or thousands of instances. Now, it is not uncommon to have to deal with datasets with hundreds of thousands or millions of instances and tens of thousands of features. When facing these problems, data mining algorithms become less effective unless they are efficiently scaled up. Very large datasets present a challenge for both humans and machine learning algorithms. As Leavitt notes [79]:

"The two most significant challenges driving changes in data mining are scalability and performance. Organizations want data mining to become more powerful so that they can analyze and compare multiple datasets, not just individual large datasets, as is traditionally the case."

Along with the large amount of data available, there is also a compelling need to produce results accurately and quickly. Efficiency and scalability are, indeed, the key issues when designing data mining systems for very large datasets [27]. In data mining, most scalability issues are linked with large datasets, with millions of instances and/or thousands of features.

In this paper, we review attempts to scale up data mining methods. We include ideas that have been used for different data mining tasks, such as feature and instance selection, classification or clustering. We are concerned with methods that are generally applicable to different data mining algorithms rather than specific methods for a certain algorithm.

Among the different techniques used for data mining, we will pay special attention to evolutionary methods, because these methods have been used very successfully in many data mining tasks [29,102,136] and, at the same time, are severely affected by scaling up problems. The evolutionary approach has a major bottleneck in the evaluation of the fitness function.

One of the potential straightforward solutions for scaling up data mining algorithms is the application of data mining algorithms to a sample of the available data. This brings up the question of whether there is a real need to scale up mining algorithms. The first reason for scaling up is that increasing the size of the learning set often increases the accuracy of learned models [107]. It is also obvious that valuable information might be lost if any form of sampling is used, especially when a large portion of instances is disregarded.

Holte et al. [62] identified overfitting as the source of the degradation in accuracy when learning from smaller samples due to the need to allow the program to learn *small disjuncts*. Small disjuncts are elements of a class description that relate to a small number of data items. In some domains, small disjuncts make up a large portion of the class description. In such domains, high accuracy depends on the ability to learn small disjuncts to account for these special cases. The existence of noise in the data, which is very common in many current research areas where massive data production is coupled with a lower reliability of the data, further complicates the problem because it is impossible to determine the difference between a special case and a spurious data point in a small sample.

Furthermore, an important issue when facing scalability problems is the quality of the obtained solutions. In many problems, scalability issues appear in the training stage, which is usually performed off-line. Thus, although the learning process may take a long time, it is only performed once. In such a case, we should be concerned with keeping the quality of the solutions when dealing with huge datasets and not only with making the algorithm faster. Many times, this important issue is overlooked.

Thus, scalability is not only a matter of time but also a matter of the quality of the solutions. Most data mining algorithms suffer from the “curse of dimensionality” [9], which means that their performance on any task deteriorates, sometimes very quickly, as the dimensionality of the problem increases [127]. Thus, scaling up a data mining algorithm not only involves improving its speed but also maintaining the quality of its solution.

When scaling up learning algorithms, the issue is not as much one of speeding up an algorithm as one of turning an impracticable algorithm into a practicable one [107]. The crucial issue is seldom “how fast” we can run a certain problem but instead “how large” a problem we can deal with.

Our first natural question is, what we do mean by a “large problem”? We must consider that due to the fast development of hardware and programming tools, this is a difficult question to answer. In fact, what it is today considered to be “large”, will soon not be considered to be large. The size of the datasets that we consider to be huge is growing rapidly. As an example the pattern analysis, statistical modeling and computational learning (PASCAL) challenge[1] includes a number of datasets that are large dataset challenges. These challenges include datasets with up to 50,000,000 instances, 15,000 features and 325,056 categories. In general, we consider that a large or very large problem is one that cannot be handled in an efficient way for a standard data mining algorithm.

Scalability is inherently pragmatic [107]. Although theoretical studies provide us with a better understanding of scalability issues, many relevant aspects are purely practical, such as memory thrashing when large amounts of memory are used, latency for storage access, etc. From a statistical point of view, issues involving file systems, virtual memory managers and task schedulers are very difficult to include in a formal theory. Of course, this does not mean that a particular theory is useless, but rather that in many cases, the programmer who is scaling up a data mining algorithm is at least as important as the theory.

In the context of inductive algorithms, Provost and Kolluri [107] identified three fundamental approaches for scaling up learning methods: (i) designing fast algorithms, (ii) partitioning the data, and (iii) using a relational representation. These three approaches are independent and can thus be combined into any data mining algorithm. We propose a new taxonomy with some common points with that of Provost and Kolluri. The approach based on the relational representation of the information is not applicable, in general, to most data mining methods and is not considered in this paper.

Our taxonomy proposal divides the scaling up methods into three groups. We first identify two main approaches: modifying the data mining algorithm, with the aim of making it faster, and modifying the data mining task, with the aim of making it simpler. The first group, which is similar to the

[1] http://pascallin2.ecs.soton.ac.uk/Challenges/.

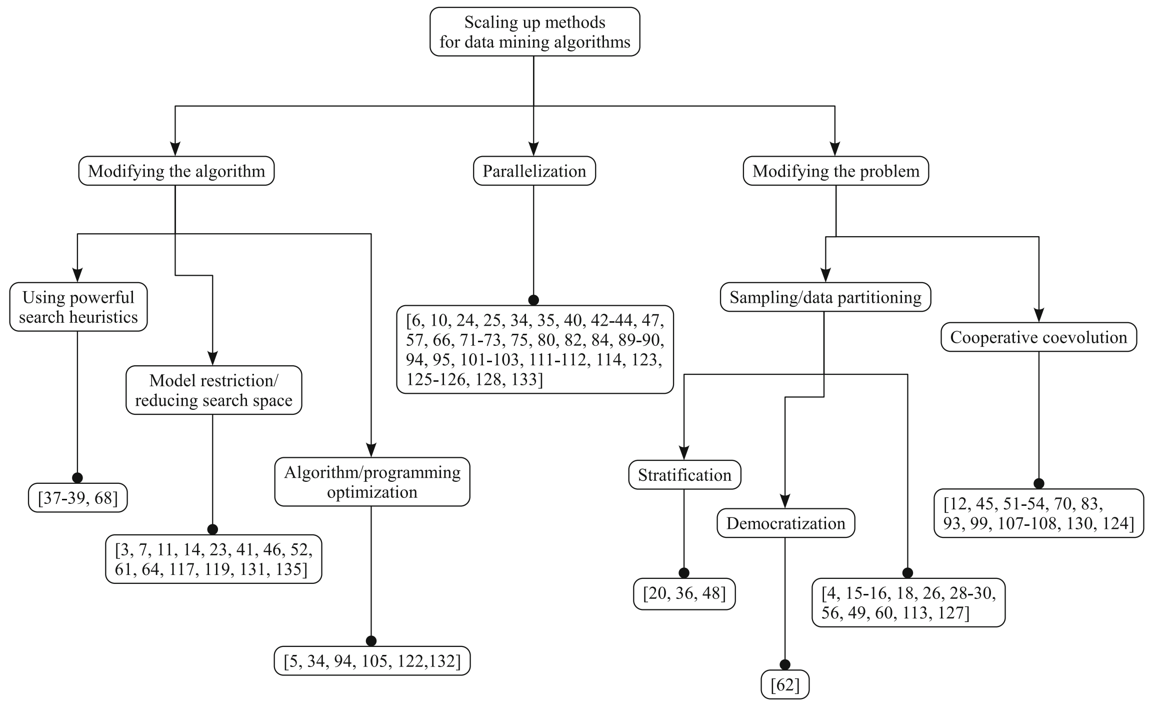


Fig. 1 Taxonomy of the different methods for addressing the scalability problem of data mining algorithms with some examples of the different paradigms

fast algorithm approach of Provost and Kolluri, includes a wide variety of algorithm design techniques for reducing the asymptotic complexity, for optimizing the search and representation, or for finding approximate solutions instead of exact solutions.

The second group of methods, consisting on modifying the problem, is based on the general principle of *divide-and-conquer*. The idea is to perform some kind of data partitioning or problem decomposition. In data mining problems, the most common situation is data partitioning, breaking the dataset up into subsets, learning from one or more of these subsets, and possibly combining the results.

Finally, we consider parallelization to be the third paradigm, separated from the other two approaches. We consider this separation because parallelization can benefit either from a fast algorithm approach, in the sense that the most costly parts are performed concurrently, or from the data partitioning approach, in the sense that the same algorithm is applied concurrently to different views of the data. In this way, parallelization would be between the other two groups. Furthermore, with parallelization there is the possibility of addressing the scaling up of the mining methods without either simplifying the algorithm or the task.

Thus, from a general point of view, we distinguish three broad approaches: modifying the algorithms, trying to make the algorithms more efficient; modifying the problems, trying to simplify them somehow; and parallelization. This taxonomy of algorithms is shown in Fig. 1, along with relevant examples of each method. We will be concerned with methods that can be applied to different data mining tasks rather than methods specifically designed for a certain algorithm.

The ideal objective of any scaling effort is to obtain linear time complexity methods. However, that objective is not achievable in most cases. Regarding the number of learning samples, Huber [66] postulated that the maximum tolerable time complexity is $O(n^{3/2})$. However, time complexity studies for many of the most sophisticated and useful data mining methods are still limited to a small subset of problems and configurations. In many cases, theory is only applicable to "toy problems". The difficulty of the theoretical representation of all the richness of the most powerful methods and meta-heuristics, together with their stochastic nature, makes the theoretical study a formidable task [95,98].

The remainder of the paper is organized as follows: Sect. 2 reviews the methods based on modifications of the algorithms to achieve scalability; Sect. 3 discusses the attempts based on parallelization; Sect. 4 reviews the methods based on modifications of the problems; and finally, Sect. 5 discusses conclusions of the paper.

2 Modifying the algorithm

This approach consists of modifying different aspects of the algorithm, aiming at a faster execution. The relevant difference with the other approaches is that the objective is to

Table 1 Summary of the methods based on modifying the algorithm for scaling up data mining procedures

Methodology	Method	Application
Model restriction and reducing the search space	Using preprocessing to simplify the problem removing some solutions	ARIMA models [48], feature selection [34,115,117], classification [13]
	Limiting the available architectures	Neural network automatic design [54,131]
	Limiting the parameters to be optimized	Neural network automatic design [22]
	Reducing the number of possible solutions	Feature selection [63], flow-shop problems [135]
Using powerful heuristics	Deriving minimum number of instances in function of desired performance	Clustering [39], EM algorithm [40], mining massive data streams [41], mining complex models [67]
	Improving the efficiency of the search algorithm	Minimum spanning trees [94]
Algorithm/programming optimization	Bookkeeping	Decision trees [5]
	Use of k-d trees	Nearest neighbor search [103,132], feature selection [120], clustering [90], classification [134]

solve the original problem using all of the available data, which is not simplified or modified in any way. Basically, we try to improve the performance of the algorithm. Provost and Kolluri [107] identified four different methods in this approach: (i) restricted model space search; (ii) using powerful search heuristics; (iii) algorithm/programming optimizations; and (iv) parallelization. As stated, we have considered parallelization to be in its own group. Table 1 summarizes the methods reviewed in this section.

2.1 Model restriction and reducing the search space

Restricting the model space has an immediate advantage in that the search space is also reduced. Furthermore, simple solutions are usually faster to obtain and evaluate and, in many cases, are competitive with more complex solutions. The major problem is when the intrinsic complexity of the problem cannot be met by a simple solution. Examples of this strategy are many, including linear models, perceptrons, and decision stumps. The combination of simple methods, such as in AdaBoost [114], can also produce powerful models that are faster than searching for a complex one. However, it is not an established fact that a simple model means a simple search. In some cases, the learning process might be longer.

The same philosophy can be applied by means of reducing the search space to a manageable size. This method can be generalized to almost any field of application. This can be done in many ways. Representation can be designed to disallow certain solutions to be explored, and search operators can be designed to avoid certain search regions. The application of this method is highly problem-dependent because the reduction of the search space depends on the problem being addressed and the search engine being used.

Preprocessing can also be used to reduce the search space. Flores et al. [48] used a preprocessing step for the design and training of ARIMA models. The preprocessing consisted of conducting a preliminary statistical analysis of the inputs involved in the model design and their impact on the quality of the results; as an output of the statistical analysis, inputs that were irrelevant for the prediction task were eliminated before using the learning method. The risk with these kinds of approaches is that the preprocessing step can remove useful information for the data mining algorithm that it is not judged as such by the fast preprocessing method.

Sebban and Nock [117] used a similar approach for feature selection [61]. Filter methods are usually less efficient than wrapper approaches [76] for feature selection. However, wrapper approaches are far more computationally expensive, with unfeasible running times when we have large datasets or thousands of features. Sebban and Nock [117] used first a filter approach to reduce the number of features to a manageable subset and then applied a wrapper method. Another interesting hybrid filter-wrapper approach was introduced by Ruiz [115] who combined a univariately preordered feature ranking filter method with an incrementally augmenting wrapper method.

Boullé [13] proposed a method for large-scale learning based on three steps: data preprocessing using discretization or value grouping, variable selection, and model averaging. The preprocessing step has a time complexity of $O(KN \log N)$ for N instances and K variables. Imposing certain constraints, the variable selection step is of time complexity $O(KN(\log K+\log N))$. The final stage of model averaging has no additional complexity as it uses results from the previous step. The storage complexity of the method is $O(KN)$. The space complexity means that the method might not be able to deal with huge datasets that will not fit in the memory. The authors developed a chunking strategy for such cases. The data are loaded from disk sequentially to avoid thrashing the memory.

De Haro et al. [34] proposed a conceptual structure for the feature selection problem called a "feasibility pyramid". Very large feature sets cannot be approached with powerful feature selection methods for computational reasons. Therefore, they recommend a cascade feature selection whereby simple methods are used to "jump" from larger to smaller feature subsets (up the levels of the pyramid). At the bottom of the pyramid (corresponding to the largest feature sets), only univariate and pseudo-multivariate methods are feasible. A systematic search traversing the candidate feature subsets would be extremely slow.

The next highest level of the feasibility pyramid includes methods suitable for simple searches among the candidate subsets, such as pairwise methods and genetic algorithms. The criteria for the evaluation of the subsets can be more sophisticated in these methods. At the second level of the pyramid, because the feature subsets that are going to be evaluated are very small, training and testing a classifier will not require prohibitive time, and thus wrapper methods can be used. This level can reveal more intricate feature dependencies compared with the base level but will not allow a small powerful and stable feature subset to be pinpointed.

On the next highest level, sequential forward and backward selection become feasible, in addition to various random-guided searches, including tabu search techniques, variants of genetic algorithms, simulated annealing, and others. Finally, when the data set has only few dimensions but the classes have a complicated structure, an exhaustive search can be applied over all possible subsets.

In evolutionary computation, the search space can be reduced by limiting the available architectures [54,131] or the parameters that are optimized by the algorithm [22]. The search space can also be reduced by means of representations that do not allow certain solutions [60,135] or by using a heuristic approach that removes some unreasonable solutions [3]. Many other examples can be found in the literature [7,11,43].

Hong and Cho [63] used a genetic algorithm for very large-scale feature selection for microarray data. In the worst case, the problem involved finding a subset of features among more than 16,000 variables. To scale up the genetic algorithm, they set a limit of 25 features as the maximum number of features that could be selected. Of course, such a drastic reduction in the possible number of solutions is likely to harm the performance of the algorithm, as the search space is highly constrained. Furthermore, that method is constrained to problems that share the characteristics of microarray data (i.e., many features but few instances). If we also had many instances, the approach would still have scalability problems.

2.2 Using powerful search heuristics

Using a more efficient search heuristic avoids artificially constraining the possible models and tries to make the search process faster. An interesting, and generalizable, proposal is presented by Domingos and Hulten [39–41] whose approach to scaling up learning algorithms is based on Hoeffding bounds [64]. The method can be applied either to choose among a set of discrete models or to estimate a continuous parameter. The method consists of three steps: first, it must derive an upper bound on the relative loss between using a subset of the available data and the whole dataset in each step of the learning algorithm. Then, it must derive an upper bound of the time complexity of the learning algorithm as a function of the number of samples used in each step. Finally, it must minimize the time bound, via the number of samples used in each step, subject to the target limits on the loss of performance of using a subset of the dataset.

Although this method can produce interesting results, the need to derive these bounds makes its application troublesome for many algorithms. Moreover, the complexity of the method is independent of the process of generating candidate solutions, but only if the method does not need to access the data during this process. In that way, it is applicable to randomized search processes. Finally, the experiments reported by the authors [40] showed that the dataset size must be several million instances for the method to be worthwhile.

The general framework proposed has been used for scaling up decision trees, Bayesian network learning, k-means clustering, and the EM algorithm for mixtures of Gaussians. In a subsequent study [67], Domingos and Hulten developed a method for inductive algorithms based on discrete searches.

In evolutionary computation, we can focus on improving the efficiency of the operators or improving the efficiency of the search algorithm. The former tries to obtain better individuals when mutation and mating is performed. The latter tries to perform a more efficient search, avoiding local minima, stagnate populations or meaningless searches in large plateaus or unpromising regions of the search space. The selection of the fitness function, the evolutionary operators, and the mechanism of evolution may have a large impact on the time complexity of the resulting algorithm. Neumann et al. [94] showed that the complexity of a genetic algorithm for obtaining minimum spanning trees can be reduced from exponential to $O(n^3)$ if the problem is modeled as a multi-objective optimization task.

There are several different aspects of evolutionary methods that can be modified with the objective of improving their search efficiency. The representation of the individuals, the genetic operators, or the method of evolution can be modified to address the scalability problem.

A useful approach is to design efficient operators that are able to accomplish a more efficient search. This way, the algorithm can be scaled to larger problems without compromising its performance. The major drawback of this approach is that these operators are useful for a certain problem and typically cannot be generalized. Furthermore, improvements over standard operators are usually obtained, but a dramatic reduction in the execution time can hardly be expected.

2.3 Algorithm/programming optimization

Algorithm/programming optimization consists of using all of the resources given by the operating system or the programming language to perform an efficient implementation. This approach differs from the use of powerful heuristics in that it consists of efficiently implementing the data mining task, avoiding redundant or unnecessary steps, but without modifying the method itself. However, the differences are subtle, and some methods can be indistinctly classified in both groups. As programming is sometimes an art [74], it is difficult to show general methods in this approach.

The most generalizable method for improving the scalability using algorithm or programming optimization may be bookkeeping. The idea under bookkeeping is that, frequently in data mining tasks, matching the hypotheses against the whole dataset is not necessary. In most cases, it is enough to use statistics that are inferred from the dataset. If the generation of these sufficient statistics is performed as a separate step, then, throughout the mining task there is no need to access the whole dataset. We can achieve large improvements in both run time and memory requirements.

Using this technique, Aronis and Provost [5] achieved tremendous run-time efficiencies for the induction of decision trees when attributes had large value sets. Other examples are k-dimensional (kd)-trees. A kd-tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts. Kd-trees can be used to perform, among other tasks, fast and efficient nearest neighbor searches [103,132]. Therefore, they can be used in any task using a nearest neighbor search as part of the mining process, such as relief algorithms for feature selection [120], clustering [90], or classification [134].

3 Parallelization

It is very likely that the development of high-performance scalable data mining tools must necessarily rely on parallel computing techniques. Two main architectures are used for parallel implementations of data mining algorithms: distributed-memory and shared-memory parallel machines. In distributed-memory machines, each processor has a private memory and local disks and communicates with other processors only via passing messages. Parallel distributed-memory machines are essential for scalable massive parallelism. On the other hand, shared-memory multiprocessor systems are capable of delivering high performance for a low to medium degree of parallelism. Individual nodes of parallel distributed-memory machines are also increasingly being designed to be shared-memory nodes. A shared-memory system offers a single memory address space that all processors can access. Processors communicate through shared variables in memory and are capable of accessing any memory location. Synchronization is used to coordinate processes. Any processor can also access any disk attached to the system. The advantage of distributed-memory parallel machines is that they can be built using clusters of inexpensive machines, whereas shared-memory machines are more expensive. Massive parallelism can only be achieved by means of distributed-memory machines for economic reasons. Parallelization is a powerful tool for scaling up data mining algorithms, both in shared-memory [70] and distributed-memory processors. Table 2 summarizes the methods reviewed in this section.

Table 2 Summary of methods based on parallelization for scaling up data mining procedures

Methodology	Application
Task parallelization	Decision trees [2,124,133], clustering [6,37,42,72,80,100,123,125]
Data parallelization	Support vector machines [23,44,59,88,87], boosting classifiers [45,78], clustering [73]
Parallel evolutionary algorithms	Database searching [31], image classification [84], vector quantization [68], feature selection [92], rule extraction [112], bioinformatics [110], image processing [46] and computational chemistry [36]

Parallelization can be applied to almost any problem, although the simplicity of the parallel version and its efficiency depend on the particular task. Parallel data mining can exploit *data parallelization* and *task parallelization*. One major issue with parallelization is that no general method for a parallel implementation of a data mining algorithm exists. In some cases, a natural parallel version of a method is easily found, and usually such implementation has a good performance and is consistent across different problems and platforms. However, in many cases, there is no obvious *ideal* parallelization of the data mining method, and there are problems with the performance, the consistency of results across problems, or both [25] in the devised implementations.

In task parallelization, the task is decomposed such that different processors perform different subtask in parallel. Load balancing and interprocess communication add additional complexity and overhead. In general, this type of parallelization does not address the problem of very large datasets because each processor will have to deal with all of the data. Another alternative, closely related to data partitioning, is to divide the dataset into smaller subsets that are processed concurrently by different processing elements.

In the field of classification, parallelization has been extensively used [49,121]. Parallelization has been used for learning many classification methods, such as in rule learning and decision trees [133]. The cost of evaluating a node in a decision tree is very high, but is also highly decomposable [2,124]. Nodes in the search space are hypothesized and each is matched against many examples to gather statistics. In the parallel matching approach, the intensive matching process is performed by distributing the dataset and matching routines to a parallel machine, while the main learning algorithm may run on a sequential front end.

Support vector machines (SVMs) are among the best performing algorithms for classification. However, SVMs are usually affected by large datasets, requiring a very long time to learn and large amounts of memory when the number of instances is in the tens of thousands. Thus, parallelization has been used extensively to improve the ability of SVMs to deal with large or very large datasets. Instead of analyzing the whole training set in one optimization step, Graf et al. [59] split the data into subsets and optimized separately with multiple SVMs. The partial results are combined and filtered again in a 'Cascade' of SVMs until the global optimum is reached. The Cascade SVM can be spread over multiple processors with minimal communication overhead and requires far less memory because the kernel matrices are much smaller than those for a regular SVM. Convergence to the global optimum is guaranteed with multiple passes through the Cascade, but already a single pass provides good generalization.

Eitrich and Lang [44] relied on a decomposition scheme, which in turn used a special variable projection method, for solving the quadratic program associated with SVM learning. By using hybrid parallel programming, this approach can be combined with the parallelism of a distributed cross-validation routine and parallel parameter optimization methods.

Chang et al. [23] improved the scalability of SVMs using a parallel SVM algorithm (PSVM), which reduces memory use by performing a row-based, approximate matrix factorization that loads only essential data to each machine to perform parallel computation. PSVM reduces the memory requirement from O(n^2) to O(np/m) and improves the computation time to O(np^2/m), where n denote the number of training instances, p the reduced matrix dimension after factorization (p is significantly smaller than n), and m the number of machines.

Lu et al. [88] proposed a distributed parallel support vector machine (DPSVM) training mechanism in a configurable network environment for distributed data mining. The basic idea presented in this study is to exchange support vectors among a strongly connected network so that multiple servers may work concurrently on distributed data set with a limited communication cost and a fast training speed. A subsequent study [87] extended the philosophy to a parallel randomized support vector machine (PRSVM) and a parallel randomized support vector regression (PRSVR) algorithm based on a randomized sampling technique. The authors claimed that the proposed algorithms achieved an average convergence rate that is so far the fastest bounded convergence rate among all SVM decomposition training algorithms. The fast average convergence bound is achieved by a unique priority-based sampling mechanism.

The Boosting [52] method is one of the most common and successful methods used for classification. Boosting [8,113, 116] is based on combining many *weak* learners into a powerful learner. The learners are trained in a stepwise fashion in which each new learner receives a different distribution of instances biased toward the instances that were found to be most difficult by previous classifiers. One of the problems for a parallel implementation of boosting is that boosting is inherently sequential. To train a classifier at step t, the output of all the previous $t-1$ classifiers for every training instance must be known. However, there are parallel versions of boosting that soften this constraint.

Fan et al. [45] developed boosting for scalable and distributed learning, where each classifier was trained using only a small fraction of the training set. In this distributed version, the classifiers were trained either from random samples (r-sampling) or from disjoint partitions of the data set (d-sampling). In r-sampling, a fixed number of instances were randomly picked from the weighted training set (without replacement), where all instances had an equal chance of being selected. In d-sampling, the weighted training set was partitioned into a number of disjoint subsets, where the data from each site was taken as a d-sample. At each round, a different d-sample was given to the weak learner. Both methods can be used for learning over very large datasets, but d-sampling is more appropriate for distributed learning, where data at multiple sites cannot be pulled together to a single site.

Lazarevic and Obradovic [78] proposed parallel and distributed versions of boosting. The first boosting modification represents a "parallelized" version of the boosting algorithm for a tightly coupled shared-memory system with a small number of processors. This method is applicable when all of the data can fit into the main memory, with the major goal being to speed up the boosting process.

In the second boosting adaptation, at each boosting step, the classifiers are first learned on disjoint datasets and then exchanged among the sites. The exchanged classifiers are then combined, and their weighted voting ensemble is constructed on each disjoint data set. The final ensemble represents an ensemble of ensembles built on all local distributed sites. The performance of ensembles is used to update the probabilities of drawing data samples in succeeding boosting iterations.

Clustering is also a data mining technique where parallelization has been widely and successfully applied. Many different methods have been developed [99]. Both, hierarchical and partitional clustering techniques are very time consuming [93]. The computational bottleneck of minimum spanning tree-based clustering algorithms is in the step of construction of a minimum spanning tree (MST). Despite the sequential essence of MST construction, a number of algorithms have been developed to parallelize calculations [93]. All parallel algorithms present parallelization of calculations with heavy Message Passing Interface (MPI). The parallelization of MSTs dates as back as far as 1980 [10]. Olson [100] achieved a time complexity of $O(V\log(V))$ with $V/\log(V)$ processors, with V being the number of vertices of the graph, and Johnson and Metaxas [71] of $O(\log^{3/2} V)$ with $V+E$ processors.

In [109], the authors implemented a parallel version of the SLINK algorithm [119] using single instruction, multiple data (SIMD) array processors. The parallel version of hierarchical clustering is presented in [80] on a n-node hypercube and a n-node butterfly. Another implementation of a parallel approach is reported in [42] with calculations distributed between processors and with the use of MPI. Using different types of parallel random access memory (PRAM) model, Dementiev et al. found [37] a good solution for minimizing the time of communication using external memory when constructing an MST, while in [6], authors dealt with the construction of a spanning tree using Symmetric Multiprocessors, though there is no guarantee that it will find an MST.

For partitional clustering, there are also parallel implementations. Stoffel and Belkoniene [125] implemented a parallel version of k-means, where the nearest center of each instance is computed concurrently. Although the complexity of k-means is not high, $O(Iknm)$, for I iterations, K clusters, n instances, and m features, this implementation may require a long time if the number of iterations for convergence is large. Othman et al. [101] developed a similar solution for clustering DNA data. There are many other parallel versions of k-means based on the principle of data distribution [73]. Judd et al. [72] developed a parallel clustering method for large-scale datasets using a master/slave architecture. The method is essentially a k-means algorithm where the assignment of the instances to the clusters is performed by the slaves.

In the same way, the bisecting k-means algorithm [123] has also been implemented in parallel with the parallel bisecting k-means with prediction (PBKP) algorithm [83] for message-passing multiprocessor systems. Bisecting k-means tends to produce clusters of similar sizes and smaller entropy (i.e., purer clusters) than k-means does. The PBKP algorithm fully exploits the data-parallelism of the bisecting k-means algorithm and adopts a prediction step to balance the workloads of multiple processors to speed up the processing of this algorithm.

3.1 Parallelization in evolutionary computation

Evolutionary computation is a field that has used parallelization at large. The parallelization of a genetic algorithm aims to improve its scalability without compromising its virtues. The most costly step in any evolutionary algorithm is the evaluation of the fitness function. Thus, when the objective of a parallel implementation is to speed up the algorithm, the parallelization focuses on the fitness function. However, there are other approaches that aim not only to speed up the search but also to improve its ability to find good solutions. In the literature of parallel implementations, scalability is described as the relationship between the size of the number of processing units and the speed-up achieved by the algorithm [108]. Synchronizing the tasks and communication often decrease the ability to speed up the parallel algorithms.

Parallel implementations of evolutionary algorithms also have the advantage of scaling up the methods to large, and even very large, problems. Furthermore, improved performance results have also been reported [31,84,118]. Three basic architectures can be considered for parallel genetic algorithms [20,82]. These architectures differ in the number of populations, the application of the genetic operators and the type of hardware architecture that they are most suitable for. The basic idea is based on a divide-and-conquer approach. The task is divided into chunks and each chunk is solved concurrently in a different processor or node. These three basic architectures are

– Global single population master/slave parallel genetic algorithms. In this architecture, there is only a single population that evolves using a standard method. Only the evaluation of the fitness of the individuals is distributed. Selection and crossover consider the whole populations. This is usually referred to as a global parallel genetic algorithm. This architecture it only aimed at scaling up the genetic algorithm, and the results are the same as a sequential algorithm. The parallelization of the evaluation of the fitness function can be synchronous or asynchronous. In the first case, the evolution is exactly the same as a standard genetic algorithm. If the evaluation is asynchronous, the faster processors do not wait for the slower ones, and there are differences in the evolution between the sequential and the parallel genetic algorithms.

The synchronous evaluation of the fitness functions has the advantage of producing more predictable results and a better understanding of the evolution. However, if the algorithm uses heterogeneous resources, slower processors may hinder the efficiency of the whole system, as faster ones will need to wait for them to finish [82]. This architecture can be implemented in both shared-memory and distributed-memory computers.

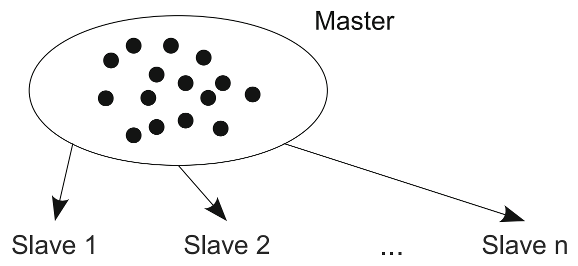
- Single population, fine-grained, or cellular parallel genetic algorithms. In this architecture, there is a single population that is spatially structured. It is designed to be run in parallel in a massively parallel processing system. Selection and matting are restricted to small neighborhoods that may overlap. In the ideal situation, each individual is evolved in a different processing element.
- Multi-population, multi-deme, or island parallel genetic algorithms. This algorithm consists of several populations that evolve independently and exchange individuals. This exchanging of individuals is called migration and it is tightly controlled by several parameters that are usually implemented in a distributed cluster of machines. The computation to communication ratio is high, so these algorithms are also called coarse-grained genetic algorithms.

Figure 2 shows the three paradigms [82]. There is a fourth alternative that combines a multi-population genetic algorithm with master-slave or fine-grained methods. This alternative is usually called a hierarchical parallel genetic algorithm [82] because we have a multi-population genetic algorithm at the higher level and a single-population genetic algorithm at the lower level.

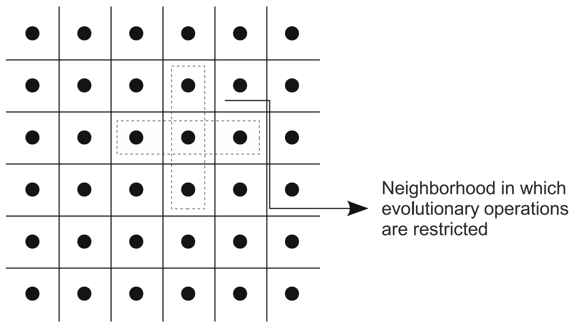
Many architectures, based on these three basic architectures, have been developed to solve different problems, such as vector quantization [68], job shop scheduling [65], OneMax problem [128], feature selection [92], rule extraction [112], and aerodynamic airfoil design [82], as well as in different application fields, such as bioinformatics [110], image processing [46], and computational chemistry [36].

Rodríguez et al. [112] proposed a distributed genetic algorithm for rule extraction in data mining to tackle large datasets. The absence of a master process to guide the search provided a better scalability of the method. This absence avoids the idle time required due to synchronization and network bottlenecks that are typically associated with master-slave architectures. Lately, several models have used the flexibility of the MapReduce framework [32] to develop parallel applications of genetic algorithms for many problems.

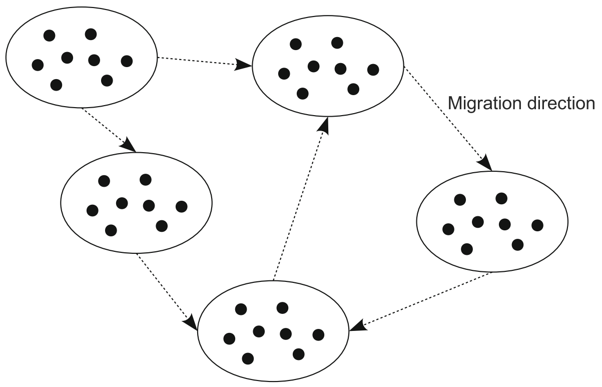
The reader interested in parallel genetic algorithms can refer to the many reviews that have been conducted on this topic [1,21,77,96].



(a) Single population master-slave genetic algorithm


Neighborhood in which evolutionary operations are restricted

(b) Cellular parallel genetic algorithm



(c) Island parallel genetic algorithm

Fig. 2 Basic models for parallel genetic algorithms

The parallel execution of evolutionary algorithms is one of the most promising ways for the efficient scaling-up of these algorithms. However, there is also a price to pay for this efficiency. First, the complexity of the algorithm is greatly increased. As one of the most interesting features of evolutionary programming is in their simplicity, this aspect is a negative side effect of the parallelization of the algorithm. Furthermore, in many applications, the major bottleneck is the evaluation of the fitness function, and several times the evaluation itself must be executed in an individual task. This case is particularly common in distributed-memory systems, which are usually more common due to their reduced price with respect to shared-memory systems.

Table 3 Summary of methods based on modifying the dataset for scaling up data mining procedures

Methodology	Method	Application
Data partitioning/sampling	Data partitioning/sampling	Feature selection [17], classification [15,24,27,28, 58,111,126]
	Stratification	Instance selection [19,38], prototype selection [50]
	Democratization	Instance selection [51,33]
Problem decomposition	Cooperative coevolution	Neural network automatic design [54,55,69,91,97,106], ensembles of classifiers [55], instance selection [53]

4 Modifying the problem/dataset

The approaches based on transforming the algorithm have some major drawbacks. First, the data mining method must be modified to be adapted to a specific situation where scalability issues have arisen. In many cases, this adaptation will be troublesome or very difficult. In the worst case, no such modification may be possible. Furthermore, the scalable data mining algorithms obtained are not usually easy to generalize. In that way, only parallelization can be considered to be a general purpose method for scaling up data mining methods. However, for parallelization, we need both a parallelizable algorithm and access to a parallel computer.

On the other hand, we can approach the issue of scaling up a data mining algorithm from the side of the problem. The algorithm is basically the same, and we try to simplify the problem. There are two major advantages on this approach. First, the scalability methods obtained in this way are easy to generalize to similar domains. Second, the speedups obtained are usually much more significant than those obtained with the modification of the method. However, these two advantages usually have the problem of more significant damage on the performance. Depending on the data mining task, we will have two basic approaches: data partitioning and problem decomposition.

One advantage of data partitioning is that this method avoids thrashing by memory management systems when algorithms try to load huge datasets into the main memory. In addition, if a learning algorithms time complexity is worse than linear in the number of examples, processing small, fixed-size data subsets sequentially can make it linear, with the constant term dependent on the size of the subsets. In either case, it may be possible to use a system of distributed processors to mine the subsets concurrently, coupling data partitioning and parallelization. Data partitioning can take the form of using subsets of instances or subsets of features.

Most of the methods for scaling up data mining algorithms in this paradigm are based on some form of sampling. The mining method is applied to a sample of the original data, or a partition of the data is performed and the algorithm is applied to different subsets. Table 3 summarizes the approaches reviewed in this section.

4.1 Sampling and data partitioning

One of the most simple methods to scale up a mining algorithm when dealing with large datasets is to sample the data, using only a, possibly small, subset of the whole dataset. The remaining data is not considered in the data mining algorithm. Of course, this method has a high cost; the possible relevant information that is stored in the disregarded data is lost. Furthermore, noise is usually linked to learning using a small portion of the data. However, several techniques that address these problems have been proposed and have produced some of the most useful algorithms for scaling up evolutionary computation. Sharing the same philosophy, we can conduct a partition of the data into small subsets and can process each subset separately. We must expect some performance degradation as the mining method applied to each subset has only a partial view of the whole problem.

A very efficient approach is the stratification strategy [19]. The stratification strategy splits the training data into disjoint strata with equal class distribution.[2] The training data, T, are divided into t disjoint datasets, D_j, of approximately equal size:

$$T = \bigcup_{j=1}^{t} D_j. \tag{1}$$

Then, the data mining algorithm is applied to each subset separately and the results of all of the subsets are combined for the final solution. If we have an algorithm of quadratic complexity, $O(N^2)$, for a number of instances, N, and we employ M strata, we will have a time complexity $O(N/M)^2$. Because we have to apply the method to the M strata, the resulting complexity is $O(N^2/M)$, which is M times faster than the original algorithm. If we are able to

[2] When dealing with class-imbalance problems the distribution of classes in each strata may be modified with respect to the whole training set to obtain a less skewed distribution.

run the algorithm in parallel in all the strata, our complexity will be $O(N^2/M^2)$ with a speedup of M^2. Of course, the drawback of this approach is the likely decrease in the performance of the algorithm. Derrac et al. [38] used stratification to scale up a steady-state memetic algorithm for instance selection. Similar principles can be applied to other data mining methods and problems [50]. One of the problems with stratification is determining the optimum the size of the strata [38].

Clustering has also been proposed as a method for reducing large datasets. Andrews and Fox [4] considered the problem of reducing a potentially very large dataset to a subset of representative prototypes. Rather than searching over the entire space of prototypes, they first roughly divided the data into balanced clusters using bisecting k-means and spectral cuts and then found the prototypes for each cluster by affinity propagation. Their "divide and conquer" approach actually performed more accurately on datasets that were well bisected, as the greedy decisions of affinity propagation are confined to classes of already similar items. The problem with this kind of approach is that the clustering process is itself a data mining task that can also suffer from scaling up problems.

Brill et al. [17] used sampling to speed up feature selection. Each individual was evaluated using its nearest neighbor error, an operation which is of $O(N^2M)$ time complexity, for N instances and M features. To avoid this complexity, a small random sample of instances was chosen to evaluate the individuals of each generation. They found that the results were competitive with the use of the whole dataset when tested on an artificial problem of 30 variables.

Similar ideas were used by De Haro-García and García-Pedrajas [35] for instance selection. The problem with the above methods is that the performance is degraded for some problems. For the case of instance selection using a genetic algorithm, the authors found that the evolutionary algorithm was too conservative when applied to subsets of the datasets. Thus, many useless instances where retained [35]. This problem was addressed with a recursive application of the stratified approach discussed above. After a first application of the stratification and the evolutionary algorithm to the different strata, a new round of stratification was applied with the selected values. This recursive approach was applied until a certain criterion was met. Although this method was very successful for instance selection, its generalization to other mining algorithms is troublesome. This approach was further developed in the *democratization* paradigm that is explained below.

In the field of classification, this philosophy has also been used. The divide-and-combine strategy [111] decomposes an input space into possibly overlapping regions, assigns a local predictor to each region, and combines the local predictors to derive a global solution to the prediction problem. The Bayesian committee machine [126] partitions a large data set into smaller ones, and the SVMs trained on the reduced sets jointly define the posteriori probabilities of the classes into which the test objects are categorized. The method proposed by Collobert et al. [28] divides a set of input samples into smaller subsets, assigns each subset to a local expert, and forms a loop to re-assign samples to local experts according to how well the experts perform. The cascade SVM method [58] also splits a large data set into smaller sets and extracts support vectors (SVs) from each of them. The resulting SVs are combined and filtered in a cascade of SVMs. A few passes through the cascade ensures that the optimal solution is found.

In the same way, Chang et al. [24] proposed a method that decomposes a large data set into a number of smaller datasets and trains SVMs on each of them. If each smaller problem deals with s samples, then the complexity of solving all the problems is on the order of $(n/s) \times s^2 = ns$, which is much smaller than n^2 if n is significantly higher than s. Decomposing a large problem into smaller problems has the added benefit of reducing the number of SVs in each of the resultant SVMs. Because a test sample is classified by only one of these SVMs, the decomposition strategy reduces the time required for the testing process in which the number of SVs dominates the complexity of the computation. One additional benefit of the decomposition approach is the ease of using multi-core/parallel/distributed computing for further increasing the speed of the process because the SVM problems associated with the decomposed regions can be parallelized idealistically.

Data parallelization is closely related to data partitioning methods. Dividing the dataset into small subsets is a way of scaling up a data mining algorithm. The next step would be to parallelize the application of the data mining algorithm to the small subsets.

Breiman [15] proposed pasting votes to build many classifiers from small training sets or "bites" of data. He presented two strategies of pasting votes: Ivote and Rvote. In Ivote, the small training set (bite) of each subsequent classifier relies on the combined hypothesis of the previous classifiers, and the sampling is done with replacement. Ivote is very similar to boosting, but the "bites" are much smaller in size than the original data set. Thus, Ivote sequentially generates training sets (and thus classifiers) by importance sampling. Rvote creates many random bites and is a fast and simple approach.

Breiman found that Rvote was not competitive in terms of accuracy to Ivote or AdaBoost. Moreover, sampling from the pool of training data can entail multiple random disk accesses, which could swamp the CPU times. Thus, Breiman proposed an alternate scheme: a sequential pass through the data set. In this scheme, an instance is read and checked to see if it will make the training set for the next classifier in the aggregate. However, the sequential pass through the data set approach led to a degradation in accuracy for the majority of the datasets. Breiman also mentioned that this approach of sequentially reading instances from the disk will not work for highly skewed datasets.

To deal with these problems, Chawla et al. [27] distributed Breiman's algorithms, dividing the original data set into T disjoint subsets and assigning each disjoint subset to a different processor. On each of the randomly sampled disjoint partitions, they followed Breiman's approach of pasting small votes. Chawla et al. combined the predictions of all the classifiers by majority vote. Again, using the above framework of memory requirement, if we break the data set up into T disjoint subsets, the memory requirement will decrease by a factor of 1/T, which is substantial. Thus, DIvote can be more scalable than Ivote in terms of memory. One can essentially divide a data set into subsets that can be easily managed by the computer's main memory.

Pasting DRvotes follows a procedure similar to that for DIvotes. The only difference is that each bite is a bootstrap replicate of size N. Each instance through all iterations has the same probability of being selected. However, DRvote also provides lower accuracies than DIvote. This finding agrees with Breiman's observations on Rvote and Ivote.

In this context of classification, a very interesting issue was raised by Brain and Webb [14]. Most approaches based on sampling or parallelization implicitly assume that the algorithms that have been designed in the context of small to medium datasets are still suitable for large datasets. In this way, their only problem would be computational efficiency. Brain and Webb hypothesized that for small datasets, the variance part of the error tends to be the most relevant part. Thus, algorithms designed to deal with small datasets are usually aimed at reducing variance. However, as the size of the problems increases, the variance term of the error decreases. Thus, for large datasets, algorithms with a lower bias would be preferable to algorithms with a lower variance. The major problem with this approach is that producing algorithms with a low bias seems to be a difficult task.

Undersampling was developed for class-imbalance problems [26], where a class, the minority class, is highly under-represented with respect to the other class, the majority class. Random undersampling consists of randomly removing instances from the majority class until a certain criterion is reached. In most studies, instances are removed until both classes have the same number of instances. However, undersampling can also be considered as a way to scale up classification methods. In many case of interest [57], the majority class is the only cause of the large amount of data, as the minority class comprises a few hundreds or thousands of instances. Undersampling the majority class works in the same way as random sampling, with the particularity that the less represented instances are kept.

One step forward in sampling techniques was developed for instance and feature selection by García-Osorio et al. [33,51]. This method combines the divide-and-conquer approach of sampling with the combination principle of the ensembles of classifiers. The method is composed of three basic steps: (i) divide the problem data into small disjoint subsets, (ii) apply the learning algorithm of interest to every subset separately, and (iii) combine the results of the different applications. The first two steps are repeated several times. As the learning method is always applied to small datasets, the speed of the methodology is impressive. As the method combines results of the application of the same learning method with different subsets of the available dataset, it is called *democratization* of algorithms.

Democratization scaling method for each step i divides the dataset S into several small disjoint datasets $S_{i,j}$. A data mining algorithm is applied with no modifications to each of these subsets, and a result $C_{i,j}$ is produced from each subset of the data. After all of the n_s rounds are applied (which can be done in parallel, as all rounds are independent from each other), the combination method constructs the final result C as the output of the data mining process.

The key difference introduced by the democratization approach is in modifying the view of data partitioning. In previous methods, the application of the learning algorithm to a subset was considered to be a valid result and a concept is learned from the subset. However, the data we are dealing with basically provide local learning from just a subset of the original dataset and suffer from locality, making the algorithm more vulnerable to noise. Consequently, democratization considers that new concepts can only be learned from the combination of the application of the learning algorithm to partitioning the whole dataset into many subsets. This difference, although subtle, is the cornerstone of the success achieved by this method. It also differs from other approaches in that the data partitioning step is repeated many times, using the basis of the ensembles of classifiers, which combine several weak classifiers to produce an efficient result. This repetition assures that we have gathered enough information to produce relevant knowledge.

4.2 Cooperative approach in evolutionary computation

In the area of evolutionary computation, the same philosophy of sampling is shared by the method of cooperative co-evolution. Cooperative co-evolution is another approach that is used to tackle large problems using a divide-and-conquer strategy [106]. As we have stated, we find scalability problems in different situations in the field of evolutionary computation. All of these problems can be considered to be some form of the *"curse of dimensionality"*, which implies that the performance deteriorates rapidly when the search space grows [91,127]. Potter [105] stated in the first steps of cooperative coevolution: “As evolutionary algorithms are applied to the solution of increasingly complex systems, explicit notions of modularity must be introduced to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents”.

In cooperative coevolution, a number of species evolve together. Cooperation among individuals is encouraged by rewarding the individuals for their joint effort in solving a target problem. The work in this paradigm has shown that cooperative coevolutionary models present many interesting features, such as specialization through genetic isolation, testing, and efficiency [106]. Cooperative coevolution approaches the design of modular systems in a natural way, as the modularity is part of the model. Other models need some a priori knowledge to decompose the problem *by hand*. In many cases, either this knowledge is not available or it is not clear how to decompose the problem. The cooperative coevolutionary model offers a very natural way of modeling the evolution of cooperative parts.

In general, the species are evolved independently with the only interaction possibly happening during fitness evaluation. The method works in three basic steps:

1. Decompose the problem into smaller problems that can be efficiently solved by evolutionary algorithms. This step is critical as two conflicting requirements must be met. The decomposition must be made in such a way that the subproblems are meaningful. At the same time, those subproblems must be small enough to avoid the scaling problem. The procedure used to perform the decomposition is problem dependent.
2. Subcomponent optimization. Each subproblem is solved separately.
3. Subcomponent co-adaptation. This step tries to capture the interdependencies among the subproblems.

The major problem with this approach is the lack of explicit considerations of the interdependencies of the different subproblems. Ways of explicitly considering the interactions are usually damaging when scaling up the algorithm. For instance, García-Pedrajas et al. [54,55] developed a cooperative method for evolving neural networks and ensembles of classifiers, where a population of subcomponents was used to combine the best individuals. Although this method achieved better results, the scalability of the approach decreased. Yang et al. [130] developed a method to account for these dependencies for the case of real variable function optimization. They proposed a group-based framework to split an objective vector into subcomponents of lower dimensionality. For coadaptation, these authors proposed an adaptive weighting strategy, which applied a weight to each of the evolved subcomponents after a cycle. The weight vector was also obtained by evolution. The algorithm is repeated for a number of cycles, with different random groups of the inputs. Li and Yao [81] extended the same concepts to particle swarm optimization. They found good results with functions consisting of up to 1,000 dimensions.

García-Pedrajas et al. [54] modified the standard method using two populations that evolved together. With a population over the populations that evolved the subproblems, they try to account for the dependencies among the subcomponents. They applied that method to the evolution of neural networks [54], ensembles of classifiers [55], and instance selection [53]. In the coevolution of instance selection, they found that the method scaled better to large problems than monolithic evolutionary algorithms.

The constrictive approach shares some basic components of cooperative approaches, where the solution is built constructively [56,69,97]. In a constructive approach, solutions are created from simple models first, and complexity is added in a stepwise manner [56]. This way, the complexity of the solutions can be arranged according to the available resources. Furthermore, when complexity is added to the model, the search space is not the same as looking for a complex model from the beginning. The partial solution that has been obtained so far limits the possible solutions and effectively reduces the search space.

The constructive approach has additional advantages, apart from its possibilities for scaling up the evolutionary algorithm [104]:

– Flexibility for exploring the search space.
– Potential for matching the intrinsic complexity of the learning task. Constructive algorithms search for simple solution first and thus offer the potential for discovering near optimal solutions of low complexity.
– Trade-offs among performance measures. Different constructive learning methods allow tradeoffs in certain performance measures, such as tradeoffs between learning time and accuracy [122].
– Incorporation of prior knowledge. These methods provide a framework to introduce problem-specific knowledge into initial problem solutions [47].

The major problem with this approach is that not all problems are suitable for a constructive approach. For this approach, we need simple solutions that can be constructed and evaluated in a consistent way. Furthermore, we need simple solutions to be developed; more complex solutions could be constructed from these solutions and could be evaluated without redoing the evaluations of the previous evolution.

5 Conclusions and future work

With the growing size of almost any problem faced by data mining methods, the need for algorithms that are able to deal with large and very large problems is becoming more critical. Data mining methods are powerful tools in many research areas for obtaining actual knowledge from raw data, but the success of these methods will be threatened unless the scalability issues are solved. In this paper, we have presented the most successful ways of scaling up data mining algorithms.

A new taxonomy based on three different approaches have been introduced: methods based on modifying the data mining algorithm to make it require fewer resources, methods based on a parallel implementation of the algorithm, and methods based on sampling or data partitioning. The methods that modify the algorithm to adapt it to large-scale problems are usually less efficient and more difficult to develop. Also, the achieved reduction in terms of running time is usually less dramatic. However, these methods tend to produce better performances than the methods that focus on modifying the problem. On the other hand, the methods based on the simplification of the problem, by means of sampling, data partitioning or in general a divide-and-conquer strategy, are more efficient in scaling up the algorithms. However, there is a reduction in the performance of the algorithm in most cases.

Parallelization is an alternative to these two methods. Its major advantage is the potential to scale up any algorithm to huge datasets. However, there are also problems with this approach. Parallel implementations are more complex than the corresponding sequential algorithms. Furthermore, in many cases, parallel versions of the algorithms are very difficult to develop and their benefits can be hampered by bottlenecks in the execution flow or the exchange of information among the tasks.

As a general approach that can be applied to many different algorithms, the two choices that are usually at the disposal of the researcher are a sampling strategy to apply the, basically, unmodified algorithm to subsets of the whole large dataset, and, when a parallel computer or a cluster of computers is available, to modify the algorithm to allow a parallel/distributed implementation. These two methods are applicable to most data mining methods.

On the other hand, specific solutions can be devised for a given data mining method using the different strategies that have been discussed in this paper. The efficiency of these solutions for scaling up mining methods varies. In this paper, we have shown several examples of these solutions.

We must also bear in mind that the benefits obtained from increasing the size of the dataset are less relevant as the dataset grows. Thus, strategies such as random sampling or stratification can be used as a first resource, due to their simplicity. In many cases, such simple approaches are as good as more complex approaches.

Acknowledgments This work was supported in part by the Grant TIN2008-03151 of the Spanish "Comisión Interministerial de Ciencia y Tecnología" and the Grant P09-TIC-4623 of the Regional Government of Andalucía.

References

1. Alba, E., Nebro, A.J., Troya, J.M.: Heterogeneous computing and parallel genetic algorithms. J. Parallel Distrib. Comput. **62**, 1362–1385 (2002)
2. Aldinucci, M., Ruggieri, S., Torquati, M.: Porting decision tree algorithms to multicore using fastflow. In: Balcázar, J.L. Bonchi, F., Gionis, A., Sebag, M. (eds.) Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases ECML PKDD, Lecture Notes in Computer Science, vol. 6321, pp. 7–23 (2010)
3. Anderson, P.G., Arney, J.S., Inverso, S.A., Kunkle, D.R., Lebo, T., Merrigan, C.: Good halftone masks via genetic algorithms. In: Proceedings of the 2003 Western New York Image Processing Workshop (2003)
4. Andrews, N.O., Fox, E.A.: Clustering for data reduction: A divide and conquer approach. Technical Report, Virginia Tech (2007)
5. Aronis, J., Provost, F.: Increasing the efficiency of data mining algorithms with breadth-first marker propagation. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, pp. 119–122. AAAI Press, Menlo Park (1997)
6. Bader, D.A., Cong, G.: A fast, parallel spanning tree algorithm for symmetric multiprocessors (smps). J. Parallel Distrib. Comput. **65**(9), 994–1006 (2005)
7. Barolli, L., Ikeda, M., de Marco, G., Durresi, A., Koyama, A., Iwashige, J.: A search space reduction algorithm for improving the performance of a ga-based qos routing method in ad-hoc networks. Int. J. Distrib. Sens. Netw. **3**, 41–57 (2007)
8. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Mach. Learn. **36**(1/2), 105–142 (1999)
9. Bengtsson, T., Bickel, P., Li, B.: Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems. In: Probability and Statistics: Essays in Honor of David A. Freedman, IMS Collections, vol. 2, pp. 316–334. Institute of Mathematical Statistics (2008)
10. Bentley, J.L.: Parallel algorithm for constructing minimum spanning trees. J. Algorithms **1**, 51–59 (1980)
11. Berger, J., Barkaoui, M.: A new hybrid genetic algorithm for the capacitated vehicle routing problem. J. Oper. Res. Soc. **54**, 1254–1262 (2003)
12. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I., Bourne, P.E.: The protein data bank. Nucleic Acids Res. **28**, 235–242 (2000)
13. Boullé, M.: A parameter-free classification method for large scale learning. J. Mach. Learn. Res. **10**, 1367–1385 (2009)
14. Brain, D., Webb, G.I.: The need for low bias algorithms in classification learning from large data sets. In: Proceedings of the 16th European Conference Principles of Data Mining and Knowledge Discovery (PKDD'2002), Lecture Notes in Artificial Intelligence, vol. 2431, pp. 62–73. Springer Verlag, New York (2002)
15. Breiman, L.: Pasting small votes for classification in large databases and on-line. Mach. Learn. **36**(1–2), 85–103 (1999)
16. Brent, M.R., Guigó, R.: Recent advances in gene structure prediction. Curr. Opin. Struct. Biol. **14**, 264–272 (2004)
17. Brill, F.Z., Brown, D.E., Martin, W.N.: Fast genetic selection of features for neural networks classifiers. IEEE Trans. Neural Netw. **3**(2), 324–334 (1992)
18. Brugger, S.T., Kelley, M., Sumikawa, K., Wakumoto, S.: Data mining for security information: A survey. In: Proceedings of the

8th Association for Computing Machinery Conference on Computer and Communications Security (2001)
19. Cano, J.R., Herrera, F., Lozano, M.: Stratification for scaling up evolutionary prototype selection. Pattern Recognit. Lett. **26**(7), 953–963 (2005)
20. Cantú-Paz, E.: A survey of parallel genetic algorithms. Calc. Paralleles **10**, 141–171 (1997)
21. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publisher, Dordrecht (2001)
22. Cantú-Paz, E., Kamath, C.: Evolving neural networks to identify bent-double galaxies in the first survey. Neural Netw. **16**, 507–517 (2003)
23. Chang, E.Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z.: Psvm: Parallelizing support vector machines on distributed computers. In: Advances in Neural Information Processing Systems vol. 20, pp. 329–340 (2007)
24. Chang, F., Guo, C.Y., Lin, X.R., Lu, C.J.: Tree decomposition for large-scale SVM problems. J. Mach. Learn. Res. **11**, 2855–2892 (2010)
25. Chattratichat, J., Darlington, J., Ghanem, M., Guo, Y., Hüning, H., Köhler, M., Sutiwaraphun, J., To, H.W., Yang, D.: Large scale data mining: challenges and responses. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, pp. 143–146 (1997)
26. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: Synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
27. Chawla, N.W., Hall, L.O., Bowyer, K.W., Kegelmeyer, W.P.: Learning ensembles from bites: A scalable and accurate approach. J. Mach. Learn. Res. **5**, 421–451 (2004)
28. Collobert, R., Bengio, S., Bengio, Y.: A parallel mixture of SVMs for very large scale problems. Neural Comput. **14**, 1105–1114 (2002)
29. Cordón, O., Herrera-Viedma, E., López-Pujalte, C., Luque, M., Zarco, C.: A review on the application of evolutionary computation to information retrieval. Int. J. Approx. Reason. **34**, 241–264 (2003)
30. Craven, M., DiPasquoa, D., Freitagb, D., McCalluma, A., Mitchella, T., Nigama, K., Slatterya, S.: Learning to construct knowledge bases from the world wide web. Artif. Intell. **118**(1–2), 69–113 (2000)
31. Cui, J., Fogarty, T.C., Gammack, J.G.: Searching databases using parallel genetic algorithms on a transputer computing surface. Future Gener. Comput. Syst. **9**(1), 33–40 (1993)
32. Dean, J., Ghemawat, S.: Mapreduce: A flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010)
33. de Haro-García, A., García-Pedrajas, N.: Scaling up feature selection by means of pseudoensembles of feature selectors. IEEE Trans. Pattern Anal. Mach. Intell. (2011) (submitted)
34. de Haro-García, A., Kuncheva, L., García-Pedrajas, N.: Random splitting for cascade feature selection. Technical Report, University or Córdoba (2011)
35. de Haro-García, A., Pedrajas, N.G.: A divide-and-conquer recursive approach for scaling up instance selection algorithms. Data Min. Knowl. Discov. **18**(3), 392–418 (2009)
36. del Carpio, C.A.: A parallel genetic algorithm for polypeptide three dimensional structure prediction. a transputer implementation. J. Chem. Inf. Comput. Sci. **36**(2), 258–269 (1996)
37. Dementiev, R., Sanders, P., Schultes, D.: Engineering an eternal memory minimum spanning tree algorithm. In: Proceedings of the Third IFIP International Conference on Theoretical Computer Science (TCS'04), pp. 195–208 (2004)
38. Derrac, J., García, S., Herrera, F.: Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability. Memet. Comput. **2**, 183–189 (2010)
39. Domingos, P., Hulten, G.: A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 106–113. Morgan Kaufmann (2001)
40. Domingos, P., Hulten, G.: Learning from infinite data in finite time. In: Proceedings of Advances in Neural Information Systems, vol. 14, pp. 673–680. Vancouver, Canada (2001)
41. Domingos, P., Hulten, G.: A general framework for mining massive data streams. J. Comput. Graph. Stat. **12**(4), 945–949 (2003)
42. Du, Z., Lin, F.: A novel approach for hierarchical clustering. Parallel Comput. **31**, 523–527 (2005)
43. Eggermont, J., Kok, J.N., Kosters, W.A.: Genetic programming for data classification: Refining the search space. In: Proceedings of the 2004 ACM symposium on Applied computing. ACM Press, New York (2004)
44. Eitrich, T., Lang, B.: Data mining with parallel support vector machines for classification. In: Yakhno, T., Neuhold, E. (eds.) Proceedings of the Fourth Biennial International Conference on Advances in Information Systems, Lectures Notes in Computer Science, vol. 4243, pp. 197–206 (2006)
45. Fan, W., Stolfo, S., Zhang, J.: The application of Adaboost for distributed, scalable and on-line learning. In: Proceedings of the Fifth ACD SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 362–366. San Diego, CA, USA (1999)
46. Fan, Y., Jiang, T., Evans, D.J.: Volumetric segmentation of brain images using parallel genetic algorithms. IEEE Trans. Med. Imaging **21**(8), 904–909 (2002)
47. Fletcher, J., Obradovic, Z.: Combining prior symbolic knowledge and constructive neural networks. Connect. Sci. **5**(3, 4), 365–375 (1993)
48. Flores, J.J., Rodríguez, H., Graff, M.: Reducing the search space in evolutive design of arima and ann models for time series prediction. In: Proceedings of the 9th Mexican International Conference on Artificial Intelligence, Lecture Notes in Computer Science, vol. 6438, pp. 325–336 (2010)
49. Freitas, A.A.: A Survey of Parallel Data Mining. In: Arner, H.F., Mackin, N. (eds.) Proceedings of the 2nd International Conference on the Practical Applications of Knowledge Discovery and Data Mining, pp. 287–300. The Practical Application Company (1998)
50. García, S., Cano, J.R., Herrera, F.: A memetic algorithm for evolutionary prototype selection: A scaling up approach. Pattern Recognit. **41**, 2693–2709 (2008)
51. García-Osorio, C., de Haro-García, A., García-Pedrajas, N.: *Democratic* instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts. Artif. Intell. **174**, 410–441 (2010)
52. García-Pedrajas, N.: Supervised projection approach for boosting classifiers. Pattern Recognit. **42**, 1741–1760 (2009)
53. García-Pedrajas, N., del Castillo, J.A.R., Ortiz-Boyer, D.: A cooperative coevolutionary algorithm for instance selection for instance-based learning. Mach. Learn. **78**, 381–420 (2010)
54. García-Pedrajas, N., Hervás-Martínez, C., Muñoz-Pérez, J.: COVNET: A cooperative coevolutionary model for evolving artificial neural networks. IEEE Trans. Neural Netw. **14**(3), 575–596 (2003)
55. García-Pedrajas, N., Hervás-Martínez, C., Ortiz-Boyer, D.: Cooperative coevolution of artificial neural network ensembles for pattern classification. IEEE Trans. Evol. Comput. **9**(3), 271–302 (2005)
56. García-Pedrajas, N., Ortiz-Boyer, D.: A cooperative constructive method for neural networks for pattern recognition. Pattern Recognit. **40**(1), 80–99 (2007)
57. García-Pedrajas, N., Pérez-Rodríguez, J., García-Pedrajas, M.D., Ortiz-Boyer, D., Fyfe, C.: Class imbalance methods for transla-

tion initiation site recognition in dna sequences. Knowl. Based Syst. **25**, 22–34 (2012)
58. Graf, H.P., Cosatto, E., Bottou, L., Dourdanovic, I., Vapnik, V.: Parallel support vector machines: the cascade svm. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.) Neural Information Processing Systems, vol. 17, pp. 521–528 (2004)
59. Graf, H.P., Cosatto, E., Bottou, L., Durdanovic, I., Vapnik, V.: Parallel support vector machines: The cascade SVM. In: Advances in Neural Information Processing Systems, pp. 521–528. MIT Press, Cambridge (2005)
60. Griffin, J.D.: Methods for reducing search and evaluating fitness functions in genetic algorithms for the snake-in-the-box problem. Ph.D. thesis, The University of Georgia (2009)
61. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
62. Holte, R., Acker, L., Porterm, B.: Concept learning and the problem of small disjuncts. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pp. 813–818. Morgan Kaufmann (2002)
63. Hong, J.H., Cho, S.B.: Efficient huge-scale feature selection with speciated genetic algorithm. Pattern Recognit. Lett. **27**, 143–150 (2006)
64. Howffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**, 13–30 (1963)
65. Huang, D.W., Lin, J.: Scaling populations of a genetic algorithm for job shop scheduling problems using mapreduce. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, pp. 780–785 (2010)
66. Huber, P.: From large to huge: A statistician's reaction to kdd and dm. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, pp. 304–308. AAAI Press (1997)
67. Hulten, G., Domingos, P.: Mining complex models from arbitrarily large databases in constant time. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining, pp. 525–531. Edmonton, Canada (2002)
68. Hwang, W.J., Ou, C.M., Hung, P.C., Yang, C.Y., Yu, T.H.: An efficient distributed genetic algorithm architecture for vector quantizer design. Open Artif. Intell. J. **4**, 20–29 (2010)
69. Islam, M.M., Yao, X., Murase, K.: A constructive algorithm for training cooperative neural network ensembles. IEEE Trans. Neural Netw. **14**(4), 820–834 (2003)
70. Jin, R., Yang, G., Agrawal, G.: Shared memory parallelization of data mining algorithms: Techniques, programming interface, and performance. IEEE Trans. Knowl. Data Eng. **17**(1), 71–89 (2005)
71. Johnson, D.B., Metaxas, P.: A parallel algorithm for computing minimum spanning trees. In: Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92), pp. 363–372 (1992)
72. Judd, D., McKinley, P.K., Jain, A.K.: Large-scale parallel data clustering. IEEE Trans. Pattern Anal. Mach. Intell. **20**(8), 871–876 (1998)
73. Kerdprasop, K., Kerdprasop, N.: A lightweight method to parallel k-means clustering. Int. J. Math. Comput. Simul. **4**, 144–153 (2010)
74. Knuth, D.E.: The Art of Computer Programming. Addison-Wesley, Reading (1997)
75. Kononova, A.V., Ingham, D.B., Pourkashanian, M.: Simple scheduled memetic algorithm for inverse problems in higher dimensions: application to chemical kinetics. In: Proceedings of the IEEE world congress on computational intelligence CEC'2008, pp. 3906–3913. IEEE Press (2008)
76. Kumari, B., Swarnkar, T.: Filter versus wrapper feature subset selection in large dimensionality micro array: A review. Int. J. Comput. Sci. Inf. Technol. **2**, 1048–1053 (2011)
77. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the traveling salesman problem: A review of representations and operators. Artif. Intell. Rev. **13**(2), 129–170 (1999)
78. Lazarevic, A., Obradovic, Z.: Boosting algorithms for parallel and distributed learning. Distrib. Parallel Databases **11**, 203–229 (2002)
79. Leavitt, N.: Data mining for the corporate masses? Computer **35**, 22–24 (2002)
80. Li, X., Fang, Z.: Parallel clustering algorithms. Parallel Comput. **11**, 275–290 (1989)
81. Li, X., Yao, X.: Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In: Proceedings of the IEEE Congress on Eevolutionary Computation CEC'2009, pp. 1546–1556 (2009)
82. Lim, D., Ong, Y.S., Jin, Y., Sendhoff, B., Lee, B.S.: Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener. Comput. Syst. **23**, 658–670 (2007)
83. Lin, Y., Chung, S.M.: Parallel bisecting k-means with prediction clustering algorithm. J. Supercomput. **39**, 19–37 (2007)
84. Liu, Z., Liu, A., Wang, C., Niu, Z.: Evolving neural networks using real coded genetic algorithm (ga) for multispectral image classification. Future Gener. Comput. Syst. **20**(7), 1119–1129 (2004)
85. Lodhi, H., Saunders, C., Shawe-Taylor, J., Christiani, N., Watkins, C.: Text classification using string kernels. J. Mach. Learn. Res. **2**, 419–444 (2002)
86. Lu, C.T., Boedihardjo, A.P., Manalwar, P.: Exploiting efficient data mining techniques to enhance intrusion detection systems. In: Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2005 Knowledge Acquisition and Management), pp. 512–517 (2005)
87. Lu, Y., Roychowdhury, V.: Parallel randomized sampling for support vector machine (SVM) and support vector regression (SVR). Knowl. Inf. Syst. **14**(2), 233–247 (2008)
88. Lu, Y., Roychowdhury, V., Vandenberghe, L.: Distributed parallel support vector machines in strongly connected networks. IEEE Trans. Neural Netw. **19**(7), 1167–1178 (2008)
89. Marchiori, E., Steenbeek, A.: An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling, pp. 367–381. Lecture Notes in Computer Science. Springer, Berlin (2000)
90. Moore, A.: Very fast em-based mixture model clustering using multiresolution kd-trees. In: Kearns, M., Cohn, D. (eds.) Advances in Neural Information Processing Systems, pp. 543–549. Morgan Kaufman (1999)
91. Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. Mach. Learn. **22**, 11–32 (1996)
92. Moser, A., Murty, M.N.: On the scalability of genetic algorithms to very large-scale feature selection. In: Proceedings of Evo-Workshops 2000, Lecture Notes in Computer Science, vol. 1603, pp. 77–86. Springer-Verlag, New York (2000)
93. Murtagh, F.: Clustering in massive data sets. In: Handbook of Massive Data Sets, pp. 501–543. Kluwer Academic Publishers, Dordrecht (2002)
94. Neumann, F., Wegener, I.: Minimum spanning trees made easier. Nat. Comput. **5**(3), 305–319 (2006)
95. Nopiah, Z.M., Khairir, M.I., Abdullah, S., Baharin, M.N., Airfin, A.: Time complexity analysis of the genetic algorithm clustering method. In: Proceedings of the 9th WSEAS international conference on Signal processing, robotics and automation, pp. 171–176 (2010)
96. Nowostawski, M., Poli, R.: Parallel genetic algorithm taxonomy. In: Proceedings of the Third International Conference on Knowledge-Based Intelligent Information Engineering Systems, pp. 88–92 (1999)

97. Obradovic, Z., Rangarajan, S.: Constructive neural networks design using genetic optimization, pp. 133–146. No. 15 in Mathematics and Informatics. University of Nis (2000)
98. Oliveto, P.S., He, J., Yao, X.: Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. Int. J. Autom. Comput. **4**(1), 100–106 (2007)
99. Olman, V., Mao, F., Wu, H., Xu, Y.: Parallel clustering algorithm for large data sets with applications in bioinformatics. IEEE/ACM Trans. Comput. Biol. Bioinforma. **6**(2), 344–352 (2009)
100. Olson, C.F.: Parallel algorithms for hierarchical clustering. Parallel Comput. V **21**, 1313–1325 (1995)
101. Othman, F., Abdullah, R., Rashid, N.A., Salam, R.A.: Parallel *k*-means clustering algorithm on dna dataset. In: Proceedings of the 5th International Conference on Parallel and Distributed Computing: Applications and Technologies, (PDCAT'04), Lecture Notes in Computer Science, vol. 3320, pp. 248–251 (2004)
102. Pal, S.K., Bandyopadhyay, S.: Evolutionary computation in bioinformatics: A review. IEEE Trans. Syst. Man Cybern. Part B Cybern. **36**, 601–615 (2006)
103. Panigrahy, R.: An improved algorithm finding nearest neighbor using kd-trees. In: Proceedings of the 8th Latin American Symposium, Lectures Notes in Computer Science, vol. 4957, pp. 387–398. Springer, Berlin (2008)
104. Parekh, R., Yang, J., Honavar, V.: Constructive neural-network learning algorithms for pattern classification. IEEE Trans. Neural Netw. **11**(2), 436–450 (2000)
105. Potter, M.A.: The design and analysis of a computational model of cooperative coevolution. Ph.D. thesis, George Mason University, Fairfax, Virginia (1997)
106. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evol. Comput. **8**(1), 1–29 (2000)
107. Provost, F.J., Kolluri, V.: A survey of methods for scaling up inductive learning algorithms. Data Min. Knowl. Discov. **2**, 131–169 (1999)
108. Quinn, M.J.: Parallel Computing: Theory and Practice. McGraw-Hill, New York (1994)
109. Rasmussen, E.M., Willet, P.: Efficiency of hierarchical agglomerative clustering using ICL distributed array processors. J. Doc. **45**(1), 1–24 (1989)
110. Rausch, T., Thomas, A., Camp, N.J., Cannon-Albrigth, L.A., Facelli, J.C.: A parallel genetic algorithm to discover patterns in genetic markers that indicate predisposition to multifactorial disease. Comput. Biol. Med. **38**, 826–836 (2008)
111. Rida, A., Labbi, A., Pellegrini, C.: Local experts combination through density decomposition. In: Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics, pp. 692–699 (1999)
112. Rodríguez, M., Escalante, D.M., Peregrín, A.: Efficient distributed genetic algorithm for rule extraction. Appl. Soft Comput. **11**, 733–743 (2011)
113. Rosset, S., Zhu, J., Hastie, T.: Boosting as a regularized path to a maximum margin classifier. J. Mach. Learn. Res. **5**, 941–973 (2004)
114. Rudin, C., Daubechies, I., Schapire, R.E.: The dynamics of adaboost: Cyclic behavior and convergence of margins. J. Mach. Learn. Res. **5**, 1557–1595 (2004)
115. Ruiz, R.: Incremental wrapper-based gene selection from microarray data for cancer classification. Pattern Recognit. **39**, 2383–2392 (2006)
116. Schapire, R.E., Freund, Y., Bartlett, P.L., Lee, W.S.: Boosting the margin: A new explanation for the effectiveness of voting methods. Ann. Stat. **26**(5), 1651–1686 (1998)
117. Sebban, M., Nock, R.: A hybrid filter/wrapper approach of feature selection using information theory. Pattern Recognit. **35**, 835–846 (2002)
118. Sena, G.S., Megherbi, D., Iserm, G.: Implementation of a parallel genetic algorithm on a cluster of workstations: Travelling salesman problem, a case study. Future Gener. Comput. Syst. **17**(4), 477–488 (2001)
119. Sibson, R.: Slink: An optimally efficient algorithm for the single link cluster method. Comput. J. **16**, 30–34 (1973)
120. Sikonja, M.R.: Speeding up relief algorithm with k-d trees. In: Proceedings of Electrotechnical and Computer Science Conference (ERK'98), pp. 137–140. Portoroz, Slovenia (1998)
121. Skillicorn, D.: Strategies for parallel data mining. IEEE Concurr. **7**(4), 26–35 (1999)
122. Smieja, F.: Neural-network constructive algorithms: Trading generalization for learning efficiency? Circuits Syst. Signal Process. **12**(2), 331–374 (1993)
123. Steinbach, M., Karypis, G., Kumar, V.: A comparison of document clustering techniques. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2000)
124. Steinhaeuser, K., Chawla, N.V., Kogge, P.M.: Exploiting thread-level parallelism to build decision trees. In: Proceedings of the ECML/PKDD Workshop on Parallel Data Mining (PDM). Berlin, Germany (2006)
125. Stoffel, K., Belkoniene, A.: Parallel k/h-means clustering for large data sets. In: Proceedings of the 5th International Parallel Processing Conference (Euro-Par'99), Lecture Notes in Computer Science, vol. 1685, pp. 1451–1454 (1999)
126. Tresp, V.: A bayesian committee machine. Neural Comput. **12**, 2719–2741 (2000)
127. van den Bergh, F., Engelbrecht, A.P.: A cooperative approach to particle swarm optimization. IEEE Trans. Evol. Comput. **8**, 225–239 (2004)
128. Verma, A., Llorà, X., Goldberg, D.E., Campbell, R.H.: Scaling genetic algorithms using mapreduce. In: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pp. 13–17 (2009)
129. Weiss, S.M., Indurkhya, N., Zhang, T., Damerau, F.: Text Mining: Predictive Methods for Analyzing Unstructured Information. Springer, Berlin, Germany (2010)
130. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Inf. Sci. **178**, 2985–2999 (2008)
131. Yao, X.: Evolving artificial neural networks. Proc. IEEE **9**(87), 1423–1447 (1999)
132. Yen, S.H., Shih, C.Y., Li, T.K., Chang, H.W.: Applying multiple kd-trees in high dimensional nearest neighbor searching. Int. J. Circuits Syst. Signal Process. **4**, 153–160 (2010)
133. Yıldız, O.T., Dikmen, O.: Parallel univariate decision trees. Neural Process. Lett. **28**, 825–832 (2007)
134. Yin, D., An, C., Baird, H.S.: Imbalance and concentration in k-nn classification. In: Proceedings of 20th International Conference on Pattern Recognition (ICPR'2010), pp. 2170–2173. IEEE Press (2010)
135. Yong, Z., Sannomiya, N.: A method for solving large-scale flow-shop problems by reducing search space of genetic algorithms. In: 2000 IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 1776–1781. IEEE Press (2000)
136. Yu, T., Davis, L., Baydar, C., Roy, R. (eds.): Evolutionary Computation in Practice, Studies in Computational Intelligence, vol. 88. Springer, Berlin (2008)
137. Zien, A., Rätsch, G., Mika, S., Schölkopf, B., Lengauer, T., Müller, K.R.: Engineering support vector machines kernels that recognize translation initiation sites. Bioinformatics **16**(9), 799–807 (2000)