Improving the Efficiency of Distributed Data Mining Using an Adjustment Work Flow

Jie Gao and Jörg Denzinger

Department of Computer Science,
University of Calgary Calgary, Canada
jie.gao.email@gmail.com, denzinge@cpsc.ucalgary.ca

Abstract. We present an extension of the usual agent-based data mining cooperative work flow that adds a so-called adjustment work flow. It allows for the use of various knowledge-based strategies that use information gathered from the miners and other agents to adjust the whole system to the particular data set that is mined. Among these strategies, in addition to the basic exchange of hints between the miners, are parameter adjustment of the miners and the use of a clustering miner to select good working data sets. Our experimental evaluation in mining rules for two medical data sets shows that adding a loop with the adjustment work flow substantially improves the efficiency of the system with all the strategies contributing to this improvement.

1 Introduction

Data mining has become an important tool for decision makers in all kinds of areas. Tools like Weka (see [5]) are available to provide a wide variety of data mining algorithms but also methods for preparing data for mining and analyzing mining results under a common user interface. Despite the efforts in these tools to simplify the task of running the entire data mining work flow, finding suitable algorithms, parameters for each algorithm, and relevant sub sets of data to work on remains a very time-consuming and difficult task.

Using groups of cooperating mining agents (miners) presents a first step towards a solution to overcome the difficulties mentioned above, since the agents can try out different mining methods in parallel and, by exchanging good mining results and other information, they can essentially create a "super"-miner combining the strengths of the individual miners. As works like [4] or [3] (see also [11] for an overview) have shown, this combination can achieve substantial synergetic gains. But there are also quite a number of problems around agent-based approaches. These include the decision when to communicate what to which other agents, as well as focusing individual agents on the right parts of the data at the right time and detecting and replacing useless agents.

In this paper, in order to solve the problems from the last sentence, we propose an approach named CoLe^2 that enhances the CoLe cooperative mining system (see [4]), by adding an outer *adjustment* work flow to the existing *cooperative*

P. Perner (Ed.): MLDM 2013, LNAI 7988, pp. 69–83, 2013.

work flow. This outer work flow essentially iterates over the usual preparation-mining-analysis data mining work flow to automate the process of finding the right set-up for the miners to create the desired results efficiently. To achieve this, the CoLe^2 system uses so-called knowledge-based strategies that incorporate knowledge about data mining in general, the mentioned three phases, the miners, and how the miners act in the cooperative environment.

Our CoLe^2 approach uses the adjustment work flow to perform iterative data selection for the miners using an X-means clustering based method (see [12]), which runs asynchronously with the inner distributed cooperation work flow. In addition, it selects miners based on their performance history in the inner cooperation iterations. It also does parameter adjustment for the miners during the mining which includes feature selection based on hints from the other miners. And we use so-called combiners that form additional more complex rules out of the results of the miners.

We instantiated the general concepts of CoLe^2 in a system aimed at mining rules about two medical databases, one focussing on diabetes (as in [4]) and the other focussing on chronic kidney disease. In our experiments we compared this system to CoLe and also looked at the individual knowledge-based strategies and their contributions. Our experiments showed that CoLe^2 substantially improves the efficiency of the mining while creating rules of the same or even better quality. The experiments also showed that all knowledge-based strategies contribute to this improvement.

2 Data Mining Basics

Data mining is the attempt to extract patterns or knowledge from data with the help of computer programs. The typical data mining process involves 4 steps (see [6]): data cleaning and integration, data selection and transformation, executing a data mining algorithm, and evaluating and presenting the results. There are many different types of data mining tasks within the mining process. The most common ones are classification, clustering and association analysis. There are also various representations for knowledge for these different tasks, like decision trees, neural networks or rules. While our CoLe and CoLe^2 concepts can be applied to all the tasks and to most of the knowledge representations, the system we use for evaluating CoLe^2 is performing data mining of rules for classifying patients.

In general, a (relational) database D can be seen as a number of tables T_1,...,T_l, where a table has a set of attributes A_1,...,A_m. Then an entry in a table is a tuple $(a_1, ..., a_m)$, where a_i is a value for A_i. A rule over D has the form *condition* $\Rightarrow$ *consequence* where *condition* is an expression consisting of predicates and certain operators, as is *consequence* (although *consequence* is usually very short). Predicates are about values of the attributes, usually having the form *att rel-op value*, where *rel-op*, a relational operator, compares the attribute *att*'s value of an entry in D with *value*. In our system, we allow $=$, $\neq$, $>$, $\geq$, $<$, and $\leq$ as relational operators. On the condition level, we use a logical operator (*and*)

and a temporal operator (*before*), together with parenthesis which override the precedence of the operators. The (*or*) logical operator is not explicitly used in the rules. Instead, the rules in a rule set have the (*or*) relation between each other.

A rule is true for an entry in a table if whenever the *condition* is true for the attribute values in the tuple, then also the *consequence* is true. Such an entry is a true positive. On the contrary, if for a tuple *condition* is true, but *consequence* is false, then this tuple is a false positive for the rule. A false negative for a rule is a tuple that does not fulfill its *condition*, but the *consequence* of the rule is true. A general goal of mining of rules is to find rules with very few (preferably zero) false positives. And the combination of all mined rules should have very few (preferably zero, again) false negatives, if classification is the goal. Based on this, there are various measures for the quality of a rule in the literature. For example, accuracy is the ratio of the number of true positives to the sum of true and false positives for the rule. Generalness is the percentage of true positives out of all entries in D for which the consequence of the rule is true. In our system in Section 4, we will combine these two measures into a *rule fitness*.

3 CoLe²

In this section, we present our agent-based CoLe^2 data mining model. We first provide a general overview, then present the cooperative work flow that is essentially identical to the CoLe work flow from [4] and finally look at the new outer adjustment work flow loop and some knowledge-based strategies that are possible to use due to this loop.

3.1 The CoLe² Model: Overview

CoLe^2 extends the CoLe approach (CoLe stands for **Co**operative **Le**arning) by having two loops (hence CoLe squared). It has not only a loop where several miners perform data mining iterations in parallel – the cooperative work flow, but also a loop iterating over the data selection, data mining, and result evaluation sequence of the usual data mining work flow, which we call the adjustment work flow. This produces the general work steps for a CoLe^2-based system as depicted in Figure 1. Naturally, details of these agents and steps depend on the concrete application. Here we provide a high-level view. An application example is provided in the next section.

Within a CoLe^2-based system, there are 3 types of agents, a controller, several miners, and one or more combiners. The controller is the agent in charge of the whole CoLe^2 work flow, realizing Steps 2, 3 and 6 in Figure 1 (naturally with help from the other agents). The controller is also responsible for Step 1 and Step 7 of Figure 1, but these steps are rather standard and not really part of the distributed agent-based approach. We will look more closely at the controller and its various tasks in the next two subsections.

The miner agents obviously are at the core of the approach, performing the mining of the data they are given. In contrast to a standard mining algorithm,

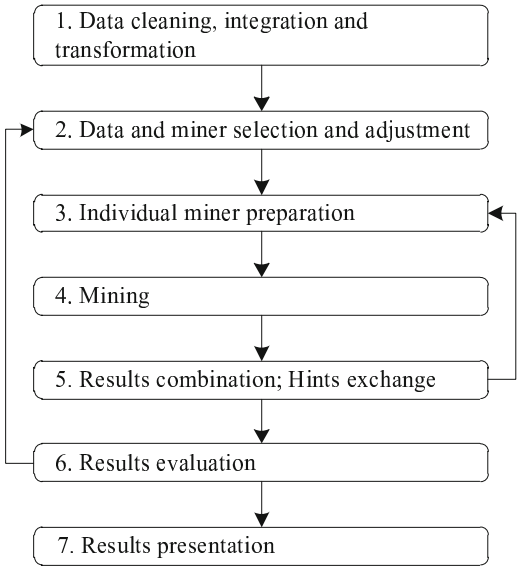

Fig. 1. CoLe^2 work flow

they have to additionally be able to make use of information communicated from other miners that run concurrently and from combiners, so-called hints, into their mining as much as possible. How much use a miner can make of hints depends on the concrete mining algorithm the miner uses. We use a group of miners that are heterogeneous, i.e. they all use different mining algorithms, although it would be possible to also have some miners using the same mining algorithm, if too much redundant work by such miners can be avoided (by, for example, having them focus on different parts of the data). Importantly, as demonstrated in CoLe, the miners can also be heterogeneous in the kind of knowledge they produce. CoLe used miners for different kinds of rules (conjunctive ones that use only *and* and temporal ones that use only *before*) and the combiners allowed to create rules combining the types. CoLe^2 allows also to use miners that do clustering to help with the adjustment work flow (to provide miners with better focus, see Section 3.3).

In general, combiner agents, as the name suggests, collect data mining results from several miners and combine them into single rule sets (i.e. realizing Step 5 in Figure 1). This goes beyond just creating the union of these rule sets. Instead, as mentioned above, the combiners use knowledge-based combination strategies to create out of pieces derived from the results of the various miners new hybrid rules. Combiners can be used at the end of each cooperation phase, in which case they are also generating hints that are made available to the miners and the controller. But a combiner can also be used as the final step of the result evaluation and to help the controller with the result preparation.

3.2 The Cooperative Work Flow

The inner loop around the cooperative work flow in CoLe^2 is still the core of a CoLe^2-based cooperative mining system. Multiple data mining agents using different mining algorithms work in parallel on creating new rules. In the first iteration of the cooperative work flow, a miner receives the data it is supposed to mine, together with any additional information the controller deems necessary for this miner (like values for its parameters). In the other iterations, a miner is only receiving information (hints) from other miners and the combiner. This information in addition to the data is used by the miner during the mining (Step 4 in Figure 1), but also in the preparation phase (Step 3) of the miner.

The preparation phase of the work flow is performed by each miner individually. In addition to the integration of information from the other agents, this allows the miner to do its individual preparation based on the requirements and characteristics of the concrete mining algorithm it uses. Usually nearly all such algorithms can profit from *attribute selection*, techniques to reduce the number of attributes in the data, which usually leads to a smaller amount of data, potentially less sparse, and faster run times. In our example application, we use an attribute selection technique based on relevance factors (as in [10]) and the hints from other agents for two of our miners (PART and the Apriori miner). The relevance factor is based on the idea that an attribute is relevant for a particular concept, if it has different distributions over data instances that are in the concept and instances that are not in the concept. Attributes that are relevant (based on a threshold regarding the difference of the distributions) get preferential treatment by the miners when creating rules. The same is true for attributes that occur often (again, defined via a threshold) in hints from other miners.

The preferential treatment during the mining phase, as mentioned before, is the main modification that we make in miners for usage in CoLe^2. Predicates using relevant attributes and predicates occurring in the best rules from other miners (as determined by the combiner, see below) should be chosen with higher probability whenever a miner has a choice that involves predicates. While from an implementation point of view, this is usually easy to accomplish (the choice needs to be made, so there must be a clearly defined place making an evaluation and only this evaluation needs to be modified), it is naturally very specific to the particular miner.

The final phase of the cooperative work flow is the domain of the combiners. Each miner sends its best rules (based on the rule fitness) to one combiner (in our example application we use only this one combiner, but it is naturally possible to use several), which then performs four subtasks: original rule evaluation, candidate rule generation, combined rule evaluation and pruning, and hint generation. It has to be noted that a combiner has to be able to handle different kinds of knowledge (hybrid rules in our case). The first subtask re-evaluates the rules from the miners, allowing for a different set of data entries to be used in the fitness evaluation (for example the combination of the sets of the different miners, if they got different data sets from the controller), but also collects

information on the frequency and quality of predicates or groups of predicates to be used in the next subtask.

In this next subtask, good predicates and groups of predicates are connected by logical or temporal operators to construct candidate rules (that now often contain pieces of knowledge from different miners). Naturally, there are many different ways how such combinations can be done, which means that there are many possible combiner agents. One simple way is to just concatenate the conditions from two original rules, but also more sophisticated methods are possible, like adding predicates to an original rule that have high potential for improvement and are connected using operators not used in the original rule.

In the third substep, these candidate rules are evaluated and pruned. The evaluation computes the fitness of the new rules (using the combiner's data set) and only rules above given thresholds survive and will be put into the result set for this iteration of the outer loop. Those rules coming from the miners that are also above the thresholds and that are not covered by a combined rule generated out of it will also be put into this result set. Finally, the combiner will also use the predicates and groups of predicates it identified as of good quality as hints that are sent to the miners for the next iterations of the cooperative loop.

3.3 The Adjustment Work Flow

Our concept of an adjustment work flow loop is motivated by the observation that decision makers that use data mining usually go through quite a number of mining attempts before they get the knowledge they are interested in. While this is partially because the decision makers have additional knowledge that they have to find ways to include into the mining process (usually by taking away data, which is what non-experts in data mining can manipulate and understand the easiest), it is also due to the various parameters and inputs todays data mining algorithms have that need to be adjusted the right way to focus on the right data to create the knowledge a user is interested in. And having several miners (and additional agents like combiners) available makes this even more difficult.

Our adjustment work flow tackles this second cause for repetitive attempts at data mining by integrating knowledge about the miners and how to influence them into the controller in form of knowledge-based strategies. Similar to the supervisor in a teamwork-based search system (see [2]), but able to deal with heterogeneous agents, the controller combines general knowledge with the observations made in the mining run so far, in order to adjust the whole agent-based mining system towards more efficiently performing mining and reducing the need for a human user to provide help. Again, the type of miners and the types of knowledge mined and generated will influence how the adjustment work flow needs to be instantiated, but there are several general knowledge-based strategies that offer guidance in this regard and that we will present in the following.

The first step in an iteration of the adjustment work flow is the selection of miners and parameters for them and the selection of the data sets these miners should work on. While in the first iteration there is not really much

to help with this, so that a standard team of miners with standard parameter settings will have to be used and, if the database is too big, standard methods for data selection, like random sampling of the data, the moment performance data for miners is available it is possible to adjust their parameters and to provide focus to the data selection. Parameter adjustment of the miners should aim to improve the whole system's effectiveness and therefore needs information about the effectiveness of the components from previous iterations of the outer loop. This information includes the number and quality of newly discovered rules. Using this information, the controller can

- inform miners with too few rules (in the last few iterations) to produce more rules (for many miners, this means lowering thresholds) and
- inform miners that produced only sub-par rules to improve rule quality (by raising thresholds, for example).

While data mining is aiming at large data sets, the reality is that most mining algorithms run into serious efficiency problems if the data set to mine gets too big. Consequently, some selection of the data to mine needs to take place and while theoretically this could be integrated into the cooperative work flow, conceptually it fits better into the adjustment work flow and in this flow we also can make use of more information. There are several possible data selection strategies, the easiest is the already mentioned random sampling. Random sampling can be slightly improved by biasing the sampling, for example by requiring the same distribution of a particular attribute in the sample as in the whole data set. Another data selection strategy aims at rule coverage by not selecting entries that are already covered by rules in the result sets from previous iterations.

For an agent-based mining system, data selection based on clustering is a very interesting method. As the name suggests, the basic idea of this method is to run a clustering algorithm (resp. agent) over the data set to create clusters of similar data entries. Then the entries in each cluster can be used as the data set for all miners for one iteration of the outer loop. It should be noted that this data selection method, due to using a rather complex algorithm, naturally itself also has a potential data selection problem. This has to be solved using another data selection strategy, like random sampling, again. And still, the clustering miner might take several iterations of the outer loop to finish. Therefore, in our experiments, we run this miner asynchronously in parallel to the adjustment work flow, using the results when they become available and before that we use random sampling or rule coverage for data selection.

The next step in the outer loop is the inner loop that we presented in the last subsection. After each iteration of the inner cooperative loop, the miners and the combiner also provide the controller with information about this last iteration, especially number and quality of the rules found by the miners and execution time differences between the miners. If a miner is not pulling its weight over several iterations, the controller can stop the inner loop prematurely and then also finish the outer loop iteration, so that it can perform adjustments in the then immediately starting next outer loop iteration.

The final step of the adjustment work flow is the results evaluation, which includes both looking at the produced rules, but also the performance of the miners. We already presented these tasks, since they also represent the first steps of the next iteration of the adjustment work flow.

4 Case Studies with CoLe^2

In this section, we present an instantiation of the CoLe^2 method to mine medical databases. One interest of doctors is the prediction of a diagnosis (disease) based on attributes other than the diagnosis itself in order to perform the appropriate tests early and then treat the disease earlier (with usually less cost and higher chances for a cure). This will be the application of our mining system. We will first provide more detail about this application, then describe the CoLe^2 instantiation and finally report on our experimental evaluation of the system.

4.1 Mining Medical Databases

We evaluated our CoLe^2 example system with two medical data sets, one about diabetes and one about kidney disease. The diabetes set was already used to evaluate CoLe (see [4]) and contains data from between 1995 and 2000 about 3150 people with a diabetes diagnosis in 2001 and 6300 control cases. Together, these people produced 436776 transactions with the Calgary health care system, each of which, among others, have a date and at least 1, but up to 3 diagnoses in the international ICD-9 code (see [7]).

The kidney disease data set has been collected by the Alberta Kidney Disease Network and contains data about all Alberta patients that had a chronic kidney disease related test in the time interval July 2001 to March 2006. Our medical partners were interested in finding rules predicting death of a patient and we had 11775 such patients among the 110452 in the database. The data base contains 6476150 transactions for these patients, again, among others, containing a date and between 1 and 3 diagnoses in ICD-9.

Since both data sets come from the Alberta health care system, they have a similar structure. We mined both data sets with a specific goal (target) in mind, namely finding rules predicting diabetes, respectively rules predicting death. This means that the consequences of the mined rules for each data set are fixed. Similar to [4], we need some preprocessing on the data, since ICD-9 is very specific and as a result there are too many only slightly different basic diagnoses. Instead of those, we used the disease groups that ICD-9 associates with the different diagnoses.

4.2 Instantiating CoLe^2

For the mining task described in the previous subsection, we instantiated the CoLe^2 method as follows. We use five miners and two combiners in addition to the controller. The four miners for the inner loop are a PART conjunctive miner

and an Apriori miner from the Weka mining library (see [5]), a sequence miner SeqGA from [4] and a relevance-factor-based descriptive miner. Due to lack of space, in the following we concentrate on the modifications we made to these miners for CoLe^2 and otherwise describe only the basic ideas of these miners.

The PART miner is based on the well-known C4.5 mining approach (see [13]) and builds partial decision trees, which are transformed into conjunctive rules by simply collecting all of the predicates on a path in the tree from the root to a leaf. We modified the PART implementation from Weka by using the predicates in hints from other miners and combiners to do attribute selection (selecting them and adding also some randomly selected attributes). We also integrated an additional pruning of the created rules before they are forwarded to the combiner, by checking for each predicate in a rule, if its elimination does not influence the rule fitness. If a predicate is found irrelevant in this way, it is eliminated. This is in addition to the usual rule pruning in a PART miner. If the controller wants the PART miner to spend less time, the miner implements this by having less pruning rounds (and vice versa).

The Apriori association miner (see [1]) also creates rules that have conjunctions of predicates as conditions. The Apriori miner builds rules from the bottom up by determining item (predicate) combinations that appear frequently in the transactions. Starting with single items that appear more often than a threshold, items (combinations) are combined and this is iterated with combinations that are above the threshold. Finally, combinations with the target predicate are split into a rule where the target predicate is the consequence and all other predicates form the condition. We modified this general method for CoLe^2 by, again, using hints from other miners for attribute selection, thus limiting the potentially huge number of item combinations. The Apriori miner can adjust its runtime by raising or lowering the threshold mentioned above.

The SeqGA miner uses a genetic algorithm (GA) to create sequence rules, i.e. rules with a condition that is a (temporal) sequence of events. The individuals of the GA are exactly such sequences, with events being a particular diagnosis and the sequence indicating that each element in it happened before the next element. As usual, the GA works on sets of such individuals and each individual in the set is evaluated with the rule fitness. We use genetic operators to create new individuals that replace the least fit individuals. Crossover just chooses two individuals, cuts them at a randomly chosen point (in the sequence) and concatenates the head part of one individual with the tail of the other. Mutation randomly either adds an event to an individual, deletes an event or substitutes an event with a random new one (or one from the received hints). In addition to sending the rules with the best fitness (above a threshold) to the combiner, the SeqGA miner also sends the predicates in these rules to the other miners as hints. The SeqGA miner can react to adjustment requests by the controller by adjusting the number of generations and, again, by adjusting the fitness threshold.

All these 3 miners evaluate rules using the same rule fitness. As in [4], this fitness fit of a rule r is calculated as

$$fit(r) = \left(\frac{tp}{tp + fp}\right)^x \times \frac{\log(tp)}{\log(positive)}$$

with tp being the number of data entries correctly classified by r, fp the number of incorrectly classified entries and *positive* being the number of entries for which the rule condition is true. x is a real number parameter that was set to 1.5 in all our experiments (following [4]).

One of the strengths of multi-agent-based data mining is the cooperation of miners, which also allows for the use of miners that alone would never be very good or even acceptable. Our relevance-factor-based miner is such a miner that essentially acts as "material generator" for the other miners. It first generates a set of predicates from attributes in the data set given to it. Nominal attributes create predicates that test equality with each of the possible values. Predicates for numerical attributes are collected from all the communicated hints from previous iterations of the cooperative work flow (in previous outer loop iterations). Then the miner calculates the relevance factor for each predicate and all predicates that are relevant (according to the factor) are retained. For each of those predicates p, a rule is constructed either of the form $p \Rightarrow$ *target concept* (if the predicate had a positive relevance) or $not(p) \Rightarrow$ *target concept* (else). These rules are sent to the combiner and the predicates are sent as hints to the other miners. The relevance-factor-based miner is only used in the first iteration of the cooperative loop within an adjustment work flow iteration.

As stated in the last section, we also use an X-means clustering miner (see [12]) as kind of an assistant to the controller to help with the selection of data sets for the miners. In contrast to the well-known k-means clustering, X-means does not produce a pre-determined number of clusters, but determines the best number of clusters between given lower and upper bounds using the so-called Schwarz criterion. In our CoLe^2-based system, we use random sampling to produce a data sub-set for the X-means miner, perform a full clustering, and use the cluster with the highest score of the Schwarz criterion as the data set for the next execution of the cooperative loop. After that, we determine all entries that are correctly predicted by the resulting rules, remove them from the remaining clusters created by the X-means miner, update the cluster centers and reassign the remaining data entries to the clusters and repeat the cooperative loop with the new best cluster. If there are no clusters left, we repeat the general X-means clustering with a new data sub-set of the whole database.

With the description of the miners, we have already presented most of the instantiation of the cooperative work flow. What remains is the instantiation of the combiner. The concrete combiner we use follows rather closely what we described in Section 3.2. It reevaluates all rules communicated by the miners and puts those above a certain threshold into the candidate set. It then performs *direct combination* and *cross combination* to create more candidates. Direct combination simply combines the conditions of two rules with an *and* operator. Cross combination computes the number of occurences of the predicates in the rules

from the miners. It then goes through all of these rules and randomly chooses a number of predicates to add and the insertion points of these predicates into the rule. For each of these points, a predicate is chosen with predicates with higher occurrence counts having a higher chance to be selected. The operator to connect the predicate to the condition is then randomly chosen and the new rule is evaluated. If it is better than the parent rule, this process is repeated until no improvement is achieved and the rule before that last attempt is added to the candidate set. Then we prune the candidate set, which includes simplifying the conditions of a rule by eliminating redundant occurrences of a predicate, throwing away predicates for which a stronger predicate is present in the condition and throwing away duplicates of a rule and rules similar to a rule with a smaller fitness. Finally, all remaining rules below a given threshold are also eliminated and the remaining rules are the result of this iteration.

Rule similarity is based on the similarity of the predictions the rules make, which essentially creates a similarity on conditions. In theory, we should compare the performance of two rules on each of the data entries, but this becomes computationally much too expensive. Instead, we assign the data entries into n bins (in our experiments we used $n = 256$) and compare two rules based on the number of correct classifications (match score) for the bins. If the difference between two rules with this regard is below a threshold, the rules are considered similar.

We have also already presented many of the pieces that form the adjustment work flow, like the data selection and how miners can adjust their parameters. As already stated in the last section, the controller performs the data selection for the first round using random sampling. It uses one agent from each of the miner types (that produce rules) in each of the iterations and tries to keep their output over all iterations balanced by advising individual miners what parameter adjustments they should make. The X-means clusterer runs in parallel to the cooperative loop, so that its results are only available in the following iteration of the inner loop. The controller collects the result sets from the combiner over all of the loops. After a given number of iterations of the adjustment loop, the controller lets a variant of the combiner perform a final simplification of the complete result set (as already described for the combiner).

4.3 Experimental Results

In this subsection, we present some experimental evaluations of our CoLe^2-based system. As stated in the introduction, our main claim is that CoLe^2 improves the efficiency of an agent-based mining system, so that our experiments focus on this aspect. Naturally, efficiency improvement should not be achieved by reducing quality, so that we also look at the quality of the rules (expressed using the rule fitness). Since some of the miners and some of the knowledge-based strategies used in the loops involve random decisions and in an agent-based system timing of messages can happen in slightly different ways, we did not perform only single mining runs, but always look at the results of three runs.

Table 1. Runtimes: with and without outer loop

Data Set	Experiment No.	Running Time (sec) without outer loop	with outer loop	Time Ratio
diabetes	1	1715.54	1234.57	71.96%
	2	2719.63	1329.73	48.89%
	3	1795.72	1535.19	85.49%
	Average	2076.96	1366.49	65.79%
kidney	1	43263.95	7242.02	16.74%
	2	17714.54	5403.13	30.50%
	3	43597.24	6590.54	15.12%
	Average	34858.58	6411.90	18.39%

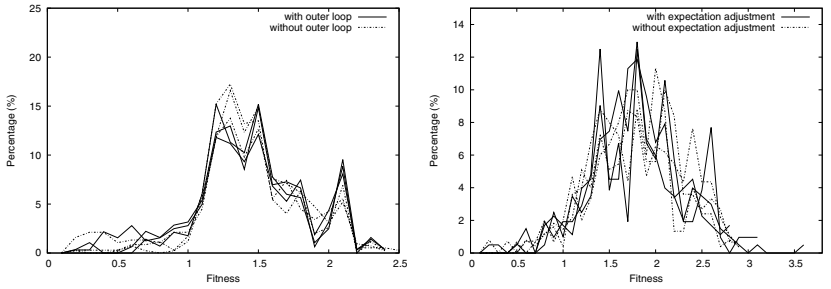

Fig. 2. Fitness histograms: left: comparison with vs without outer loop; right: comparison with vs without parameter adjustment

Effect of the Outer Loop: Obviously, the outer adjustment loop is the major contribution of this paper, so that our first experimental series evaluates the effect of this loop against a configuration of the system that simulates just the CoLe method. More precisely, the CoLe mode still uses random sampling to provide small enough working data sets for the miners and performs a loop around the inner loop that creates different working data sets, but the other features of the outer loop are not used. Both systems used otherwise the same parameter settings and were run until more than 95% of the data entries were used in at least one run of the inner loop.

As Table 1 shows, the outer loop clearly reduces the run time for the system and importantly does so substantially for the larger data set. For this data set, even the worst mining run with outer loop is 3 times faster than the run not using the outer loop (but starting with the same initial working data set).

Figure 2 (left) shows a fitness histogram of the quality of the mined rules for the 6 runs on the larger kidney set. While it could be argued that the rule quality with the outer loop is slightly better (higher peaks after a fitness value of 2), using the outer loop definitely does not result in loosing quality, so that our primary goal with CoLe^2 is clearly fulfilled.

Parameter Adjustment: One of the knowledge-based strategies employed by the controller is adjusting the parameters of the miners to have them contribute

Table 2. Run time (seconds): with and without parameter adjustment

Dataset	Test No.	Without Parameter Adjustment	With Parameter Adjustment	Time Ratio
diabetes	1	1722.55	1223.51	71.03%
	2	1667.44	907.87	54.45%
	3	2473.14	1014.09	41.00%
kidney	1	5122.68	3943.20	76.98%
	2	9710.18	6156.67	63.40%
	3	8805.31	4655.57	52.87%

evenly to the mining results and to have them running in the inner loop iterations without large idle times. To test this strategy, we compare in Table 2 runs of the system using all knowledge-based strategies with runs where no parameter adjustments of the miners take place.

As can be seen, using all strategies always results in faster runs and for the smaller data set it can be up to only half of the time when using this strategy versus not using it. And, as Figure 2 (right) shows, this, again, is achieved without loss of quality.

X-means Data Selection: To look into the improvement the X-means miner represents, we measured the quality of rules generated by the miners and the combiner of essentially one iteration of the outer loop and compared the average rule quality using X-means directed data selection versus random sampling. We did this for the larger kidney data set, since this data set definitely needs data selection.

As Table 3 shows, while the results with regards to the miners are mixed, when combined by the combiner, the average rule quality is clearly enhanced, showing the usefulness of this knowledge-based strategy.

Table 3. Average rule fitness of one iteration: X-means vs random sampling data selection

Test No.	Miner or Combiner	Data Selection Method	
		Random Sampling	X-means
1	PART miner	0.324	0.242
	SeqGA miner	0.851	0.867
	Relevance factor miner	0.553	0.576
	Combiner	**1.275**	**1.464**
2	PART miner	0.228	0.600
	SeqGA miner	0.985	0.625
	Relevance factor miner	0.558	0.560
	Combiner	**1.374**	**1.790**
3	PART miner	0.309	0.310
	SeqGA miner	1.043	1.107
	Relevance factor miner	0.558	0.560
	Combiner	**1.606**	**1.858**

Other Strategies: Due to lack of space, we cannot provide detailed results for the other strategies. But our experiments showed that having an asynchronous workflow resulted in improvements between 30 and 70 percent. Although [4] already showed the usefulness of the created hints, we performed appropriate experiments, again, and the hints produce an efficiency improvement between 20 and 50 percent and, more importantly, substantial improvements in the quality of the produced rules (compared to not using hints in the miners).

5 Related Work

Agent-based data mining is of interest because of the ability to deal with distributed databases. The data mining work has to be done distributively either due to privacy concerns, or due to the large total data amount which prohibits a centralized database. An example of a method aimed at this reason is the JAM framework described in [14], where multiple data sites exist and each of them employs a classification agent to perform data mining and a meta learning agent to exchange and combine the classification models. Another example is the frame presented in [9], where the focus is to establish distributed cooperative data mining in a competitive and privacy restricted environment, with each data mining agent working on their own private data set using Naive Bayes classifiers. In these situations, it is natural that the agents in the data mining system cooperate towards the same data mining goal. In comparison to typical work in this area, we are not only interested in mining large amounts of data and enabling cooperation, but also particularly interested in producing hybrid knowledge that goes beyond what each of the individual mining agents can produce.

Most approaches in the literature that have some centralized control concept like our controller have it as part of the inner work flow and do not use heterogeneous agents (which naturally makes the control task much easier). In [3], a manager agent divides the data into disjunct working sets for analyzer agents that, after having done their own mining, vote on whether to keep the results of the other agents. [8] is another example for a simple mining loop. We are not aware of any other work using two loops and the benefits of the outer loop, like mining cooperatively a certain part of the database or having a specialized assistant miner to prepare for that.

6 Conclusion and Future Work

We presented CoLe^2, a concept for agent-based mining around two work flow loops, an inner cooperative loop and an outer adjustment loop. The outer loop allows for various knowledge-based strategies that allow a controller agent to adapt and focus the whole team of agents on the particular database and mining task. Our experimental evaluation showed that the outer loop substantially improves the efficiency of the mining system while still producing results of comparable quality to without using the outer loop.

The outer loop offers several more possibilities for using knowledge-based strategies that we intend to look into in the future. While in this paper we concentrated on generally applicable strategies, naturally for databases for which the mining needs to be periodically repeated it makes sense to integrate application specific knowledge into the mining. The outer loop also offers the possibility to integrate human judgement in form of advice to the controller.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Fayyad, U.M. (ed.) Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI Press (1996)
2. Denzinger, J., Kronenburg, M.: Planning for distributed theorem proving: The teamwork approach. In: Görz, G., Hölldobler, S. (eds.) KI 1996. LNCS (LNAI), vol. 1137, pp. 43–56. Springer, Heidelberg (1996)
3. de Paula, A.C.M.P., Ávila, B.C., Scalabrin, E.E., Enembreck, F.: Using distributed data mining and distributed artificial intelligence for knowledge integration. In: Klusch, M., Hindriks, K.V., Papazoglou, M.P., Sterling, L. (eds.) CIA 2007. LNCS (LNAI), vol. 4676, pp. 89–103. Springer, Heidelberg (2007)
4. Gao, J., Denzinger, J., James, R.C.: A cooperative multi-agent data mining model and its application to medical data on diabetes. In: Gorodetsky, V., Liu, J., Skormin, V.A. (eds.) AIS-ADM 2005. LNCS (LNAI), vol. 3505, pp. 93–107. Springer, Heidelberg (2005)
5. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations Newsletter 11, 10–18 (2009)
6. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques, 3rd edn. Morgan Kaufmann (2011)
7. Karaffa, M.C. (ed.): International Classification of Diseases, 9th Revision, Clinical Modification, 4th edn. Practice Management Information Corp., Los Angeles (1992)
8. Kargupta, H., Hamzaoglu, I., Stafford, B.: Scalable, distributed data mining using an agent based architecture. In: Proc. 3rd KDD, pp. 211–214 (1997)
9. Lisý, V., Jakob, M., Benda, P., Urban, Š., Pěchouček, M.: Towards cooperative predictive data mining in competitive environments. In: Cao, L., Gorodetsky, V., Liu, J., Weiss, G., Yu, P.S. (eds.) ADMI 2009. LNCS, vol. 5680, pp. 95–108. Springer, Heidelberg (2009)
10. Liu, H., Lu, H., Yao, J.: Toward multidatabase mining: Identifying relevant databases. IEEE Transactions on Knowledge and Data Engineering 13(4), 541–553 (2001)
11. Moemeng, C., Gorodetsky, V., Zuo, Z., Yang, Y., Zhang, C.: Agent-based distributed data mining: A survey. In: Cao, L. (ed.) Data Mining and Multi-agent Integration, pp. 47–58. Springer (2009)
12. Pelleg, D., Moore, A.W.: X-means: Extending k-means with efficient estimation of the number of clusters. In: Proc. 17th ML, pp. 727–734 (2000)
13. Quinlan, J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann (1993)
14. Stolfo, S.J., Prodromidis, A.L., Tselepis, S., Lee, W., Fan, D.W., Chan, P.K.: JAM: Java agents for meta-learning over distributed databases. In: Proc. 3rd KDD, pp. 74–81 (1997)