CoLe: A Cooperative Distributed Data Mining Model

Jie Gao[1], Jörg Denzinger[1], and Robert C. James[2]

[1] Department of Computer Science
University of Calgary, Canada
{gaoj,denzinge}@cpsc.ucalgary.ca
[2] Aechidna Health Informatics
rob@aetiologic.ca

Abstract. We present *CoLe*, a cooperative, distributed model for mining knowledge from heterogeneous data. *CoLe* allows for the cooperation of different learning algorithms and the combination of the mined knowledge into knowledge structures that no individual learner can produce. *CoLe* organizes the work in rounds so that knowledge discovered by one learner can help others in the next round. We implemented a *CoLe*-based system for mining diabetes data, including a genetic algorithm for learning event sequences, improvements to the *PART* algorithm for our problem and combination methods to produce hybrid rules containing conjunctive and sequence conditions. In our experiments, the *CoLe*-based system outperformed the individual learners, with better rules and more rules of a certain quality. Our improvements to learners also showed the ability to find useful rules. From the medical perspective, our system confirmed hypertension has a tight relation to diabetes, and it also suggested connections new to medical doctors.

1 Introduction

In recent years, data mining has been a hot topic that attracts both database and machine learning researchers. With the rapid development of data storage capacity, data mining calls for methods to handle large quantities of data, which could be heterogeneous with various types of data. However, most of the existing data mining methods are only capable of processing homogeneous data. Even many large-scale distributed data mining methods do not consider the heterogeneity of data much. Another problem in data mining is that methods are more and more specialized (as was indicated in [1]). They perform well on ideal data sets. But when applied to real-world data, they often cause unsatisfactory results. Quite often different parts of the same data set are heterogeneous in characteristics. A method may need different tuning for those different parts.

Therefore it is necessary to propose a data mining model that allows for multiple different methods (or differently tuned methods) to work on one data set to handle its heterogeneity. In this model, it is not satisfactory to leave the results from multiple learners isolated. We need to consider the relations among

them and produce integrated results as the knowledge discovered from the whole heterogeneous data set.

In this paper, we present a cooperative distributed data mining model in a multi-agent system framework. In such a multi-agent system, the agents could use different methods to handle different types or parts of information in heterogeneous data sets. The results are combined to get integrated results. Critically, this model implies that **agents cooperate with each other**, so that our multi-agent mining model is not merely applying multiple algorithms to a data set, but trying to gain synergetic effects through the cooperation.

The rest of this paper is organized as follows: The *CoLe* model is introduced in Sect. 2. A proof-of-concept implementation in mining medical data is described in Sect. 3, and experimental results with this system are presented in Sect. 4. Section 5 compares our model with some related work in distributed data mining. Finally Sect. 6 concludes this paper with some remarks and future work.

2 A Cooperative Learning Model

Our model for cooperative data mining is called *CoLe* (**Co**operative **Le**arning).

First of all, our model works on different types of data. We can take them altogether as a "super data set". So we can define the data like this:

Definition 1. *Let $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n$ be n data sets related to the same topic but in different types (or with different characteristics). We call the data set*

$$\mathcal{D} = \mathcal{D}_1 \bowtie \mathcal{D}_2 \bowtie \cdots \bowtie \mathcal{D}_n$$

a heterogeneous data set *about the topic. And each of the $\mathcal{D}_1, \cdots, \mathcal{D}_n$ is called a* simplex data set *in $\mathcal{D}$.*

In this definition, the data sets $\mathcal{D}_1, \cdots, \mathcal{D}_n$ are related to the same topic, which means that there are some keys to keep the relations among them.

From the data set we expect to discover *knowledge*. Although *knowledge* is an abstract concept and in many cases the knowledge representations depend on the specific learning/mining method, we can represent (or convert) the knowledge in the majority of cases into an "if...then..." form. A piece of knowledge in this representation can be regarded as a *rule*. So we can define our concept of *rule* like this:

Definition 2 (Rule). *Let condition be a pattern that can match some tuples in a data set, and conclusion be some characteristics a data instance can have. We call the following representation of a piece of knowledge a* rule*:*

$$condition \Rightarrow conclusion$$

In our learning model, we have several learning/mining methods employed to discover different rules from a given heterogeneous data set. For each simplex data set in it, we can have one or more suitable learning methods to discover knowledge from it. Such a learning method is called a *learner*:

Definition 3 (Learner). *Let $\mathcal{D}_i$ be a simplex data set in a given heterogeneous data set $\mathcal{D}$. A* learner *is a learning/mining method (algorithm) that resembles a function $l : \mathcal{D}_i^+ \rightarrow \mathcal{K}_i^+$, where $\mathcal{K}_i$ is the set of possible rules (knowledge) to be discovered by the learner.*

In our learning model, given the heterogeneous data set $\mathcal{D}$, the learner set will be $\mathcal{L}$. For all $l \in \mathcal{L}$, $\mathcal{R}_l$ is the set of possible rules to be discovered by l. And $\mathcal{C}_l = \{conclusion | condition \Rightarrow conclusion \in \mathcal{R}_l\}$. Then the following should hold for two different learners:

$$\text{For } l_1 \text{ and } l_2 \text{ in } \mathcal{L},\ \mathcal{C}_{l_1} \bigcap \mathcal{C}_{l_2} \neq \phi$$

so that we can combine the rule sets learned by different learners. Otherwise, the rules never have any conclusion in common, which gives us no hope to combine the rules. Another remark on the simplex data sets is that for $\mathcal{D}_i$ and $\mathcal{D}_j$, they do not necessarily need to be different. The different indices are mainly used for showing the learners' ownerships of the simplex data sets. It is quite possible that the same simplex data set is worked on by several different mining algorithms.

2.1 Our Cooperative Learning Model

The cooperative learning system we propose can be described by a multi-agent system model so that it is easy to describe the system model and the interaction of the components at an abstract level without unnecessary details.

In this cooperative multi-agent learning system, there are two types of agents. One type are learner agents, which implement the mining methods. (We can call them directly *learners* without ambiguity since the *learner* definition, Definition 3, and "learner agents" are referring to the same concept from different points of view.) The other type of agents are agents that combine the different types of rules. We always assume there is only one such agent in a system (multiple such agents can be taken as a whole as a super agent). We call this agent *combination agent*, or Ag_{CBN} for short. It is at the core of the cooperation in our learning system model.

Ag_{CBN} decides how the learners cooperate. Its main task is to receive the learned rule sets from individual learners, use some strategies to combine them, evaluate the combined rules against the whole heterogeneous data set and put the satisfactory ones into a final rule set, and at the same time extract helpful information from the rule sets and send it back to the learners so that each learner can utilize the discoveries from other learners.

The implementation of Ag_{CBN} can vary according to different needs. It can be a simple agent that only validates the different combinations of the rule sets. It can also be a group of agents (forming a super Ag_{CBN}) to achieve complex interaction and cooperation strategies with the learners.

Basically our cooperative learning model contains the following elements: A heterogeneous data set $\mathcal{D}$ (whose simplex data sets decide what learners to use),

a set of learner agents $\mathcal{L}$, and the combination agent $\mathrm{Ag_{CBN}}$. And we name this model *CoLe* (**Co**operative **Le**arning):

$$CoLe =< \mathcal{D}, \mathcal{L}, \mathrm{Ag_{CBN}} >$$

The agent interactions in *CoLe* are illustrated in Fig. 1: Given a heterogeneous data set $\mathcal{D}$, each learner l_i learns on the simplex data set ($\mathcal{D}_i$) it can handle. The learned set of rules ($\mathcal{R}_i$) are sent to $\mathrm{Ag_{CBN}}$, which combines the rules and considers their quality in the whole heterogeneous data set $\mathcal{D}$, and puts the combination results to the final rule set $\mathcal{R}$. At the same time, for each learner l_i, $\mathrm{Ag_{CBN}}$ extracts useful information from $\mathcal{R}_1, \cdots, \mathcal{R}_{i-1}, \mathcal{R}_{i+1}, \cdots, \mathcal{R}_n$ and sends this feedback to l_i to help its work. The feedback could be rule condition fragments to help form good rules, data set clusters to help concentrate on uncovered data cases, etc.. In this model, the learners are cooperating via the intermediate $\mathrm{Ag_{CBN}}$.

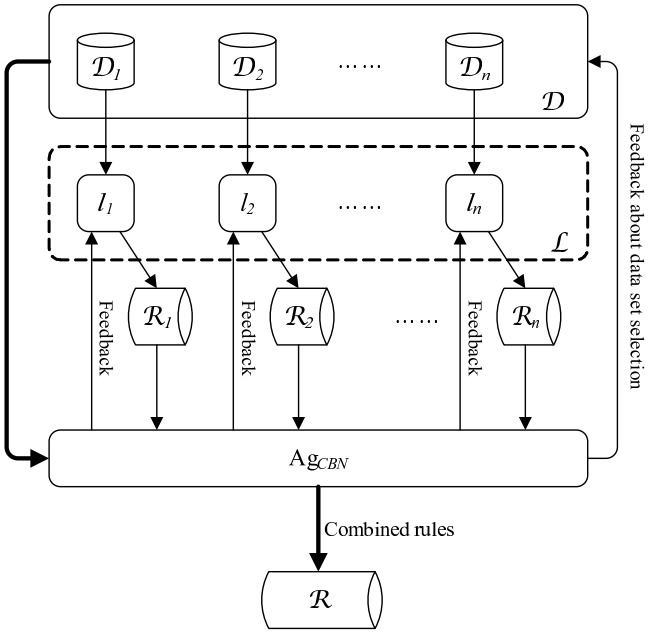

Fig. 1. *CoLe* model

In the above description, we propose a learn-and-feedback way of cooperation. To make use of the feedbacks, a learner should receive them before its learning work ends. In our model, this is achieved by dividing the whole learning process into iterations. In this way, the learners are synchronized to send their partial results after an iteration ends. $\mathrm{Ag_{CBN}}$ can then do its combination work and send feedback to help the learners' work in future iterations. The iterations continue

until some end condition is met (e.g. an iteration number limit, or some rule quality threshold). This workflow of *CoLe* is presented in Fig. 2.

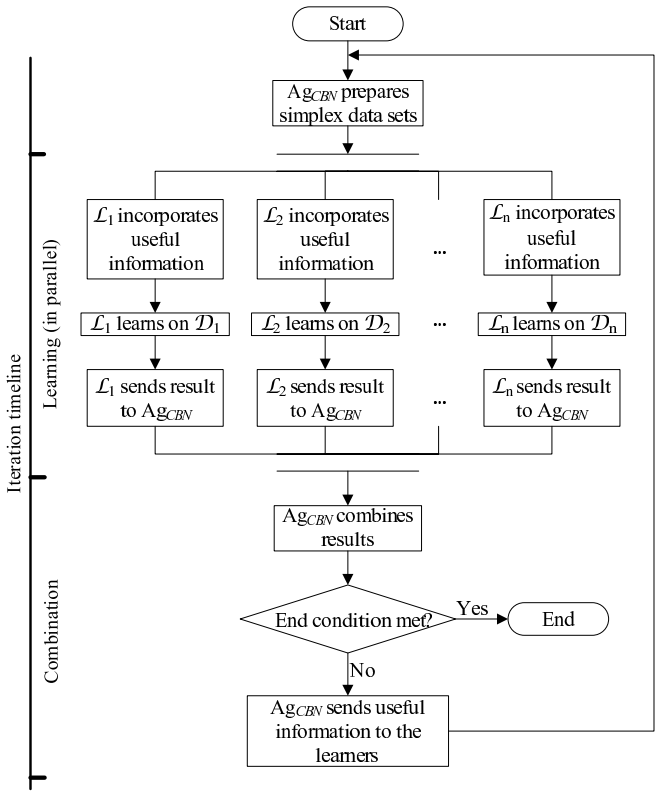

Fig. 2. *CoLe* work flow

3 Application Problem and *CoLe* Solution

As a proof of concept for the *CoLe* model, a cooperative learning system was developed and implemented for mining diabetes patient data from the Calgary Health Region. The application problem definition and the solution using the *CoLe* model are discussed here.

In the Canadian public health system, large amounts of data about individuals' medical services and diagnostic codes are collected. For now, the data is only used for public health insurance and billing purpose. As the data contains very complete and comprehensive medical records, we want to use data mining methods to the data, to discover knowledge that can help identify future diseases using a case history. To identify the real disease cases from such data,

the medical experts are not only interested in the diagnoses the individuals ever had, but, more over, they are also interested in the chronic development of the diseases. The *CoLe* learning model, with the ability to learn from different types of data, is very promising for their needs.

Here the particular disease we are focusing on is diabetes. In the diabetes health data, an instance that is really a diabetes patient is called a *case*, and an instance that is not a real diabetes patient is called a *control*.

3.1 Problem Definition

In Sect. 2, the general *CoLe* model has already been discussed. When we fit the application problem — discover knowledge from medical history for identifying future diabetics — into the *CoLe* model, we instantiate it as follows.

The *heterogeneous data set* in this problem is the public health data, which contains two different *simplex data sets*. One simplex data set $\mathcal{D}_s$ is the medical records of the individuals. These records come with timestamps and can be interpreted as discrete temporal sequences. The other simplex data set $\mathcal{D}_c$ is the non-temporal version of these medical records (with only boolean fields for each possible diagnosis) and their annual statistics of the number of medical services, together with the personal information (with an aggregated 10-year age group label).

For the temporal simplex data set, we need a *sequence learner* (*SL*) to work on it. *SL* is using a genetic algorithm to identify some key diagnosis sequences that are the indication of future diabetes diagnoses. These key diagnoses are called *events*. We do not care about the time when an event takes place but about the relative order of the events. *SL*'s results are actually rules that use temporal sequences as conditions and the conclusion is always "diabetes". Such rules are called *sequence rules* in the following.

For the non-temporal simplex data set, the learner is called *conjunctive learner* (*CL*) because its results are rules with conjunctions of predicates as conditions. The predicates are in the form "attribute *rel_op* value", where *rel_op* is a relation operator such as "=", ">","<", and the order of these predicates in a rule is of no consequence. These rules are called *conjunctive rules*.

We use a single combination agent Ag_{CBN} to combine the rules from *CL* and *SL* respectively. The combined rules may contain both sequence conditions and conjunctive conditions. Thus we call them *hybrid rules*. A sequence rule or conjunctive rule can also be taken as a hybrid rule when we take its sequence condition or conjunctive condition as empty (or always *true*).

In summary, in this problem of mining diabetes health data, we have the data set $\mathcal{D} = \mathcal{D}_s \bowtie \mathcal{D}_c$, three types of rules: sequence rules, conjunctive rules and hybrid rules, the learner set $\mathcal{L} = \{SL, CL\}$, and the combination agent Ag_{CBN}.

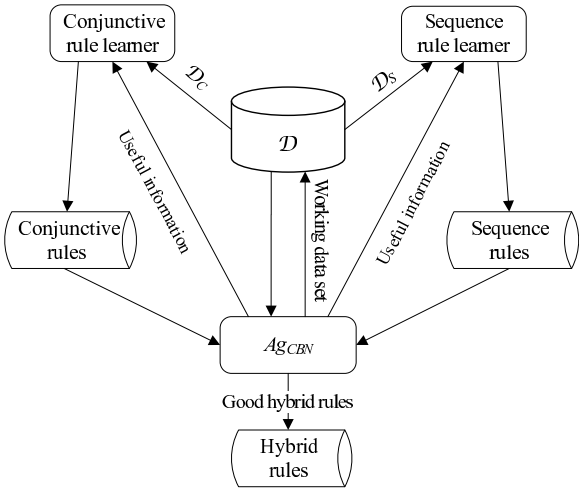

Fig. 3. Learning system model

3.2 System Design

As already stated, there are three major components in our system, namely *SL*, *CL* and Ag_{CBN} (see Fig. 3). The system works in iterations, presented in Fig. 4: Each learner receives its simplex data set, and learns rules from it. Both types of rules are sent to Ag_{CBN}, which combines them into hybrid rules and then validates them against the heterogeneous data set. The hybrid rules with higher quality than a pre-defined threshold are put into the hybrid rule set as the result of the combination. At the same time, Ag_{CBN} also gives help to the two learners — generate the simplex data sets for the next iteration and send useful information extracted from *CL*'s results to *SL* to help its future work. Such mining iterations will continue until our predefined iteration number limit has been reached.

Data Sets The data set our system is going to work on is considerably large. To make the learners run efficiently, we do not let the learners run directly on the whole data set. Instead, we introduce *working data sets* with a smaller size for the learners to work on. In each of our mining iteration, Ag_{CBN} generates working data sets for *SL* and *CL*. The working data sets contain the same set of instances but different types of data, fitting the two learners respectively. The details of generating the working data sets are presented in Ag_{CBM}'s design.

Sequence Learner The sequence learner (*SL*) is the agent that does mining on the temporal simplex data set, and finds the sequence rules that can reflect the temporal relation of the diagnoses that can predict diabetes. *SL* first uses a genetic algorithm to discover interesting sequence rules from the data. It then

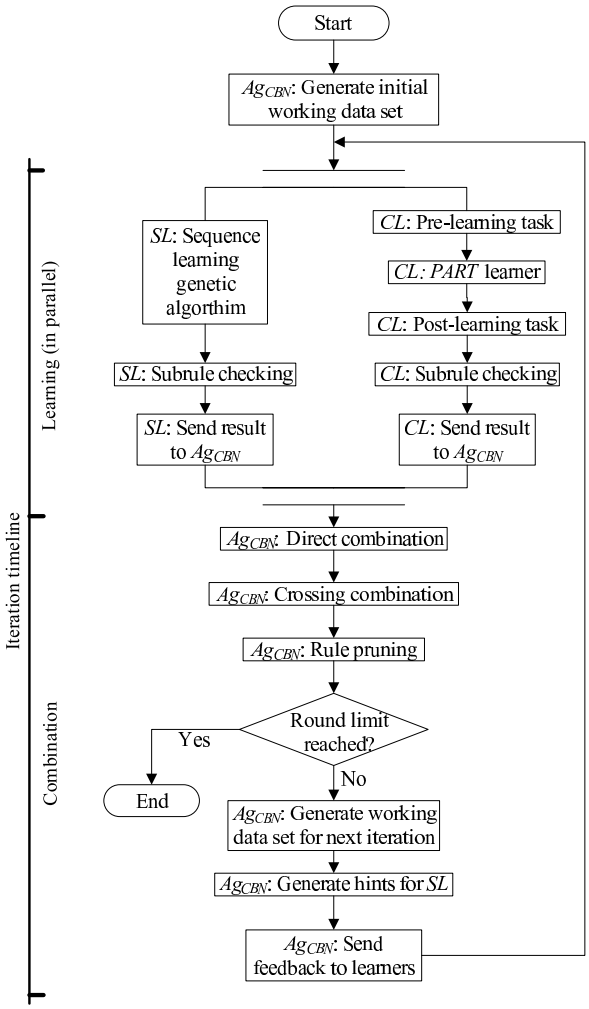

Fig. 4. Learning system work flow

checks the subrules of the discovered sequence rules. Finally the top sequence rules and good subrules are sent to the combination agent.

SL uses a genetic algorithm to achieve temporal rule learning. We designed a Michigan-like approach (see [2]) with sequence rules as individuals. A Michigan-like approach has a smaller granularity and its average run time is shorter than that of Pittsburgh approaches. This is suitable for dividing the whole genetic algorithm search process into many short runs — more suitable for cooperation as required by *CoLe*. In the representation of individuals, these sequence rules have a common conclusion: "diabetes", i.e., *SL* only learns positive rules from the data set. This is suitable when we are interested a knowledge discovery instead

of a classification problem. And we can omit the conclusion part since they are all the same in *SL*. So finally, the actual individuals are sequences that indicate diabetes cases.

Besides the commonly used genetic operators crossover and mutation, we designed an "intelligent" genetic operator called *intellicut*, using similar ideas from [3], to make the evolutionary process more targeted. In *intellicut*, each event in the middle of an individual is checked to see if the quality of the individual can be increased by cutting off the events after this point, i.e., it will cut off a bad "tail" in the sequence. In this way, even if an individual is not very good, good parts of it are preserved.

The fitness of an individual is evaluated according to its *accuracy* — the ratio of true positives to all instances it matches, and *coverage* — the ratio of true positives to all cases in the data set. These two factors are both considered to make sure the individuals contain valid knowledge and they will not overfit the data. To balance the accuracy and coverage, we propose the following equation to calculate the fitness:

$$fitness = 10 \times \left(\frac{tp}{tp + fp}\right)^x \times \frac{\ln(tp)}{\ln(case_num)} \tag{1}$$

In (1), tp and fp are the true positives (*cases* matched by this individual) and the false positives (*controls* matched by this individual) respectively, and $case_num$ is the number of all *cases* in the data set. Here in the coverage calculation, we do not directly divide tp by $case_num$. Instead, we calculate their logarithm value before the division. In this way, the fitness is less sensitive to coverage when the coverage is big enough. The fitness is the product of the two factors so that neither the accuracy nor the coverage can be low. The exponent x of the accuracy controls the weight of the two factors in the fitness. And we multiply it by 10 so that the resulting range will have better readability. In fact, (1) is used as a global assessment of rule quality throughout the system. Different x values are chosen for specific situations by experimenting with various x values in these situations.

Conjunctive Learner The conjunctive learner (*CL*) is the agent that mines on the non-temporal simplex data set to discover conjunctive rules.

The base learning algorithm used for *CL* is the *PART* algorithm (see [4]). It forms rules from pruned partial decision trees built using *C4.5*'s learning method (for *C4.5*, see [5]). The resulting rules are in the form of conjunctive rules. *PART* has no global optimization and therefore can return a rule immediately after discovering it. This gives us the ability to interrupt it halfway with partial results, allowing run time limits for *CL* and easy synchronization with other learners. In our system we use an implementation of *PART* directly from the machine learning package *WEKA* (see [6]).

As *PART* is designed for creating rule sets for classification problems, it is necessary to do some pre- and post-learning work to make the result more suitable for our cooperative mining problem.

The pre-learning tasks are mainly used for data reduction. The original data set contains so many attributes (possible diagnostic codes) that the *PART* algorithm would generate rules with a large amount of "junk" predicates and it would take a long time to run. Therefore, it is necessary to reduce the number of distinct attributes.

In the pre-learning task, we use a *relevance factor* to calculate how much an attribute can discriminate the *cases* and *controls* (inspired by [7]). The equation is:

$$RF(A) = \Pr(A) \times \log\left(\frac{\Pr(A|case)}{\Pr(A|control)}\right) \tag{2}$$

In (2), A is the attribute being calculated, $\Pr(A)$ is the probability of A having value *true* in the data set, and $\Pr(A|case)$ and $\Pr(A|control)$ are the probabilities of A having value *true* given the instance is a *case* or a *control* respectively. This *relevance factor* will have a high absolute value if an attribute is frequent enough and it is relevant for discriminating the *cases* and *controls*. In the actual calculation, these probability values are estimated by the corresponding frequencies in the non-temporal data set. respectively. A relevance factor threshold is set to get those relevant attributes into *CL*'s data input.

The post-learning task is used for generalizing the results from *PART*. The *PART* algorithm favours the accuracy more than we should do in this mining problem and consequently makes the coverage of the rules very small — too specialized for combination. We need to relax the condition of the rules and make them cover more instances, at the cost of lowering down their accuracy a bit. We do not harm the mining result by doing this, because in our specific application, having some false positives is acceptable as long as we have large coverage on the disease instances. The fitness measure in (1) is used to balance the coverage and accuracy.

The post-learning task is done in two stages:

1. Rule merging: every pair of rules in the result is checked by the fitness measure to see if we can get a better rule by using the intersection of the two rules' conditions as the condition of the new rule. We do this as a hill-climbing process in iterations. In each iteration, the pair that can lead the biggest fitness boost is chosen and one (both) of them is (are) replaced by the new rule if the new fitness is higher. The iterations continue until no replacement can be done.
2. Predicate pruning: For each rule, we test each predicate to see if we can gain a higher fitness by removing this predicate. This is also a hill-climbing process and done in iterations. In each iteration, the predicate resulting in the biggest fitness boost if removed is chosen for deletion. The iterations also continue until no deletion can result in a fitness increase.

These post-learning stages can simplify the conditions of the rules, making them more generic. This is better for the combination.

Subrule Checking in Both Learners Subrule checking is done in both *SL* and *CL* before sending their results to Ag_{CBN} to prepare more candidates for combination. A subrule is one whose condition is a subset of the original condition (and for sequence rules, the events should also appear in the same relative order as the original rule). For each rule, all the subrules of this rule will be checked against a fitness threshold over the entire data set $\mathcal{D}$. All qualified subrules are sent together with the original result set to Ag_{CBN} as the result of an individual learner.

Although this subrule checking process seems to be similar to some earlier tasks in the learners, e.g. the post-learning task in *CL* and the deletion of an event in the mutation genetic operator in *SL*, its purpose is different from them. Tasks like post-learning and genetic operators are mainly used for replacing a rule with a new one. They can help increase individual rule quality. But the subrule checking process here is aimed at producing as many good rules as possible for the combination. All subrules, whether better or worse than the original ones, are put to the rule set if their fitnesses are higher than the threshold. This brings more "materials" for the combination while the earlier tasks aim at increasing the overall rule quality.

Combination Agent The cooperation in the *CoLe* model is mainly achieved by Ag_{CBN} combining the results from the learners. In our specific problem, the combination has to produce hybrid rules from conjunctive and sequence rules.

In Ag_{CBN}, the combination inputs are *srules* and *crules*, which are the result rule sets of *SL* and *CL*, respectively. The combination is done in several stages:

1. Direct combination: Combine a sequence rule with a conjunctive rule (including subrules)
2. Crossing combination: Covert a predicate to an event (or vise versa) and combine with existing hybrid rules
3. Rule pruning: Remove duplicates and unnecessary parts
4. Working data set generation: Generate the working data set for the next iteration to the learners
5. Hints to *SL*: Give hints to *SL* according to *CL*'s results.

In these stages, a rule's quality is also measured by the fitness calculated by (1). And a fitness threshold ft is set before hand. The combined rules with fitness greater than ft will be put into the hybrid rule set *hrules*. We assume all these rule sets contain only rules with a conclusion "diabetes", because the knowledge we intend to discover is "what a diabetes patient should be like" instead of "what a diabetes patient should not be like".

In direct combination, if a rule in *srules* or *crules* already has greater fitness than ft, it will be put to *hrules* directly. Then each possible direct combination of a sequence rule in *srules* and a conjunctive rule in *crules* will be checked. If this new hybrid rule has a higher fitness than ft, it is also put into *hrules*.

The second stage, crossing combination, uses the hybrid rules in *hrules* after the first two stages, together with *srules* and *crules*, as the base. This is a deeper

combination than the first two stages. The basic idea of crossing combination is to convert some diagnostic predicates in conjunctive rules to events and put them into the sequence parts of a hybrid rule to see if the hybrid rule's fitness increases, or vise versa. The conversion can be made because both the diagnostic predicates and events are on the same set of possible diagnoses. For example, we can convert a predicate "$Diagnosis_A = true$" to an event "$[Diagnosis_A]$". In this way, the use of one learner's result to help the other is maximized. A hill-climbing method is used in the iterations to find the best results for each hybrid rule's combination. The detailed algorithm is given in Fig. 5.

After the combination stages, the resulting rule set *hrules* is pruned to eliminate duplicate rules and useless conditions according to the following criteria:

- Single-event sequences are converted to a predicate since a single event cannot indicate any temporal order.
- A predicate that has a counterpart in the sequence part is erased because the condition in the sequence part is stronger than the conjunctive part.
- Duplicate rules are erased.

Ag_{CBN} then generates the working data sets for the two learners. The first working data set is generated randomly from the whole data set before the mining work starts. Later ones are generated according to the combination results, based on the previous working data set. We denote the previous working data set and the new (currently generating) one as W_{o} and W_{n} respectively. $\mathrm{Ag_{CBN}}$ first eliminates the correctly covered instances (true positives), as well as the true negatives covered by conjunctive rules predicting *controls*, from W_{o}. The remaining instances can take only up to 80% in W_{n}. The rest of pending instances in W_{n} are randomly picked from the whole data set. By doing this, $\mathrm{Ag_{CBN}}$ can guide the miners to focus on the cases that have not been covered by existing rules without driving the miners into smaller and smaller corners.

The predicates of the conjunctive rules from *CL* may contain some key indicators of diabetes. So they can act as topic specific knowledge to *SL*. After the combination, Ag_{CBN} will extract the predicate attributes from *CL*'s result, and make some candidate sequence segments from them to help *SL*'s work. These candidate segments can be used by the mutation operator in *SL*'s genetic algorithm. For example (shown in Fig. 6), we have conjunctive rules cr_1 and cr_2 coming from the conjunctive rule learner. Let $evt_1, \cdots, evt_5$ correspond to $pre_1, \cdots, pre_5$. We have sequence segments $ss_1, ss_2, \cdots, ss_8$ to be used as candidate individual parts in *SL*.

We do not have a counterpart for giving hints to *CL*, because *CL* is using an existing implementation and we have few ways to influence the *PART* algorithm directly.

4 Experimental Evaluation

Our cooperative learning system has been tested with different experiments to show the advantage of cooperation, as well as the various improvements in the system.

```
Algorithm CrossingCombination
input: D  //Diabetes census data set
       R  //Hybrid rule set after the first two combination stages
       S  //Result of SL, the sequence rule set
       C  //Result of CL, the conjunctive rule set
output: H  //Hybrid rule set
E: list of events converted from predicates
P: list of predicates converted from events
h1: new hybrid rule from combination
R0, R1: hybrid rule set scratch for loops
01 begin
02  //Predicates to events
03  foreach conjunctive_rule c in C do
04    foreach predicate p in c do begin
05      covert p to event e;
06      insert e into E;
07    end
08  R0=R;
09  while not R0 empty do begin
10    foreach hybrid_rule h in R0 do begin
11      h1=h;
12      foreach event e in E do begin
13        append e to h1's sequence part;
14        fitness_calculation(h1,D);
15        if h1.fitness>h.fitness then begin
16          insert h1 to R1;
17          insert h1 to H;
18        end
19      end
20    end
21    R0=R1;
22    R1.clear;
23  end
24  //Events to predicates
25 foreach sequence_rule s in S do
26    foreach event e in s do begin
27      covert e to predicate p;
28      insert p into P;
29    end
30  R0=R;
31  while not R0 empty do begin
32    foreach hybrid_rule h in R0 do begin
33      h1=h;
34      foreach predicate p in P do begin
35        insert p to h1's conjunctive part;
36        remove all diagnostic codes indicated by p from h1's sequence part;
37        fitness_calculation(h1,D);
38        if h1.fitness>h.fitness then begin
39          insert h1 to R1;
40          insert h1 to H;
41        end
42      end
43    end
44    R0=R1;
45    R1.clear;
46  end
47  return H;
48 end
```

Fig. 5. Crossing combination algorithm

4.1 Data Preparation

The diabetes medical control data for our learning system comes from the Calgary Health Region. The data contains population born before 1954 and have been living in Calgary continuously since 1994. We want to analyze the medical records 5 years prior to the identification of diabetes. So we keep only the individuals who have no diagnoses of diabetes before 2000 but have at least one diabetes diagnosis in 2000 (April 1, 2000 to March 31, 2001), i.e., first diagnosed as diabetes patients in 2000. They are the *cases* in our data. For each of the

$$cr_1: \quad pre_1 \wedge pre_3 \Longrightarrow disease$$
$$cr_2: pre_2 \wedge pre_4 \wedge pre_5 \Longrightarrow disease$$

can generate sequence segments:

$$ss_1: \quad evt_1 \rightarrow evt_3$$
$$ss_2: \quad evt_3 \rightarrow evt_1$$
$$ss_3: evt_2 \rightarrow evt_4 \rightarrow evt_5$$
$$ss_4: evt_2 \rightarrow evt_5 \rightarrow evt_4$$
$$ss_5: evt_4 \rightarrow evt_2 \rightarrow evt_5$$
$$ss_6: evt_4 \rightarrow evt_5 \rightarrow evt_2$$
$$ss_7: evt_5 \rightarrow evt_2 \rightarrow evt_4$$
$$ss_3: evt_5 \rightarrow evt_4 \rightarrow evt_2$$

Fig. 6. Hints from *CL* to *SL* example

cases, we also give 2 *controls* who are in the same sex and similar age but have no diabetes diagnoses at all.

In the original data set, there are three tables. One is the registration table containing ID, age, gender, and class (*case* or *control*). The other two are medical records, one for hospital (HOSP table, see Table 3) and the other for clinical services (MD table, see Table 2). The medical records are mainly the diagnostic codes given by the doctors, together with the date of service. In Tables 1, 2 and 3 the sample data tables are shown (some unused fields are omitted).

Table 1. Sample data table REG

ALT_ID	CC	GENDER	YEAR
2	0	M	1921
3	0	F	1922
5	0	F	1946
6	0	F	1930
19	1	M	1940
	...	...	

We put the clinical and hospital medical records together and only extract the records from 1995 to 1999 (April 1, 1995 to March 31, 2000) for both the *cases* and *controls* to find knowledge in the 5-year period before diabetes diagnoses.

The first step of preparation is to aggregate the diagnostic codes. The codes are defined by the International Classification of Diseases, 9^{th} revision, or ICD-9 for short (see [8]). A full-length ICD-9 code is often 5–6 digits long and there are over 17000 possible codes. These are too many for learning because it will result

Table 2. Sample data table MD

...	SERV_SDATE	DIAG1	DIAG2	DIAG3	ALT_ID
...	2000-03-06				2
...	2000-03-25	595			2
...	2000-03-14	V70.0			2
...	1997-08-11	706			3
...	1997-06-27	594.9	788.0		3
...	1995-12-07	466			5
...	1999-12-14	733			5
...	1999-10-08	174			5
...	2000-02-25	780.5			6
...	2000-02-25	780.5			6
...	1999-09-15	785			6
...	1997-05-06	783	717.8	719.4	7
...	1997-10-27	723.1			7
		...	...		

Table 3. Sample data table HOSP

ADMIT	DX_1	DX_2	DX_3	DX_4	DX_5	...	ALT_ID
1996-12-21	9975	5968	E8788	7140	36610	...	3
1997-06-19	57420					...	3
1997-06-30	57410	V1301				...	3
1998-04-08	9962	E8781				...	7
1999-04-16	99677	72709	2851	E8781		...	7
1999-08-12	5409					...	7
1998-04-27	9962	99813	E8781			...	7
			...	...			

in a very sparse data set. We use a higher-level abstraction, the ICD-9 Basic Tabulation List[9], to aggregate the diagnostic codes. In this list, the diagnostic codes are abstracted to 307 disease types in about 70 categories. And we also gain a clearer logical relation in the diagnostic codes.

The second step is to split the data set into sequence and conjunctive simplex data sets. For the sequence simplex data set, we order each instance's diagnoses, both hospital and clinical, by the date of service, and use this diagnosis sequence as the instance's sequence data. The conjunctive simplex data set contains two parts. One part is the basic information of the instances: age, gender and statistics including the number of medical services each year and average number of medical services per year. The other part is a boolean table for the diagnoses. If an instance ever had a diagnosis, the corresponding field has the value *true*, and vice versa.

There were two data sets prepared for the testings. A small data set contains 1800 instances, with 600 *cases* and 1200 *controls*. The corresponding working data set for the learners contains 900 instances. This small data set is used for

most of the tests and comparisons in Sections 4.2 and 4.3. We also prepared a large data set with all valid instances we have, containing 9450 instances (3150 *cases* and 6300 *controls*). The corresponding working data set size is 3150 instances. Test runs on this larger data set were made to get rules for evaluation of the knowledge discoveries in Sect. 4.4. In Table 4 some detailed statistics about the two data sets are presented.

Table 4. Data sets

Data set	Small	Large(full)
Size	1800	9450
Cases	600	3150
Controls	1200	6300
Number of aggregated diagnostic codes	308	
Number of events	81087	436776

4.2 The Effect of Combination

Our first evaluation is the comparison of the single learners and the combinations so that we can show the effect of combination in our system. We had five test runs with different fitness thresholds in Ag_{CBN}. These fitness thresholds control how a rule is qualified (see Sect. 3.2). Table 5 shows these fitness threshold values. Other parameter values were decided by some experiments before hand so that their values used in the tests were reasonable and the same for all our tests here.

The resulting hybrid rules are focused on in our evaluation. The first comparison is the number of rules with different origins. If a rule is the contribution of a single learner, the origin is the corresponding learner. This includes qualified original rules from a learner, as well as the qualified subrules in the subrule checking stage in Ag_{CBN}. The rules with contributions from both learners take "Combination" as their origins. When our system records the origins, single-learner origins have higher priority, i.e. when a rule discovered by a single learner happens to be re-discovered by Ag_{CBN}, we give credit to the single learner instead of Ag_{CBN}. In Table 5 we can clearly see that the combined rules are much more than the ones with origins *SL* or *CL*. This is the first indication that the combination can produce more potentially good rules according to our fitness measure. Additionally, these numbers also show that *SL* can discover more qualified rules than *CL*. This is because:

1. Sequence rules are naturally more complex than conjunctive rules, and thus better for expressing complex knowledge about diabetes
2. *CL*'s base algorithm is intended for classification problems instead of getting descriptive knowledge (although our post-learning work improves this)

Table 5. Tests with different thresholds

Test No.		1	2	3	4	5
Fitness threshold		3.8	3.9	4.0	4.1	4.2
Rule origins	*SL*	263	104	25	13	5
	CL	0	0	3	0	0
	Combination	5745	1852	743	287	72
	Total	6008	1956	771	300	77
Fitness averages	*SL*	3.973	4.089	4.162	4.282	4.251
	CL	N/A	N/A	4.183	N/A	N/A
	Combination	4.054	4.117	4.258	4.312	4.344
	Overall	4.051	4.116	4.254	4.311	4.338
True positive averages	*SL*	67.18	61.73	65.36	26.62	33.00
	CL	N/A	N/A	46.333	N/A	N/A
	Combination	56.55	55.66	45.92	26.49	25.25
	Overall	57.02	55.98	46.55	26.49	25.75

However, this does not mean *CL* has no contribution. Every combined rule contains parts from *CL*'s results, either predicates or events converted from conjunctive rules' predicates (otherwise the origin will be *SL* instead of Combination).

The average fitness of the rules by origin is calculated as well. The averages differ as the thresholds are different. The average fitness of the combined rules is a bit higher than the ones from single learners. This again indicates that combination increases the quality of the discovered knowledge, or in another word, produce better knowledge from two types of materials.

In each test run, the combined rules have comparable true positive coverage with the ones from single learners. This shows that our combination strategies are not overfitting the given data. Although the true positive coverages of combined rules are a bit lower than the ones of individual miners, this shows that the combined rules have higher accuracy, because our fitness measure in (1) is balanced between coverage and accuracy and the combined rules have higher fitness.

In addition to the summaries for the whole rule sets, summaries for the top 10 (by fitness) rules in each test are also presented here in Table 6. From this table we can see that most of the top 10 rules are combined rules. And in the only test with rules from *SL* in the top 10, the combined rules are obviously better than the ones from *SL*. For the same reason indicated above, in Test No. 2 the true positive average of rules from *SL* is a bit higher than that of the combined rules, indicating combined rules have higher accuracy.

In summary, the results of the five test runs shown in Table 5 indicate that our combination/cooperation model successfully makes much more better rules out of the raw discoveries from the individual learners.

Table 6. Tests with different thresholds (top 10 rules)

Test No.		1	2	3	4	5
Fitness threshold		3.8	3.9	4.0	4.1	4.2
Rule origins	*SL*	0	2	0	0	0
	CL	0	0	0	0	0
	Combination	10	8	10	10	10
	Total	Top 10				
Fitness averages	*SL*	N/A	4.594	N/A	N/A	N/A
	CL	N/A	N/A	N/A	N/A	N/A
	Combination	4.769	4.684	4.821	4.696	4.491
	Overall	4.769	4.666	4.821	4.696	4.491
True positive averages	*SL*	N/A	24.50	N/A	N/A	N/A
	CL	N/A	N/A	N/A	N/A	N/A
	Combination	23.80	23.38	24.40	23.20	25.60
	Overall	23.80	23.60	24.40	23.20	25.60

4.3 Particular Experiments

In addition to the comparisons of the rules from different origins in the previous section, some particular experiments are also made to show the contribution of various knowledge-based designs in our system.

Hints in Sequence Learner In Section 3.2 we presented how Ag_{CBN} generates hints from *CL*'s results and gives them to *SL* to make more cooperation possible. And an example was also given in Figure 6. This mechanism was specially tested and evaluated to prove its contribution.

The tests are to run *SL*'s genetic algorithms with and without hints, to see if the results will be different in quality. To get some valid hints for the tests, the run log in test No. 3 in Table 5 was used and we extract the hints and working data set in the tenth iteration from it. The tests made here all ran on this working data set, and the ones with hints used this hint set. To make sure we have a general result without the interference of random numbers, we repeat each test individually for five times. So totally we have 5 runs with hints, and 5 without hints. In each run, we use a generation size of 300, evolve the individuals for 50 generations, and finally filter out those individuals with fitness greater than 2.8.

Table 7 shows the qualified individual numbers for each test run. From the numbers and the averages, an obvious conclusion is that the runs with hints generate more qualified individuals. More detailed observations are made on the evolutionary process. In Fig. 7, average fitnesses of the first 10 generations in each test run's evolutionary process are drawn in lines. We can see the average fitnessses for test runs with hints increase much faster than the ones without hints.

Table 7. Number of qualified individuals

Generation size	300					
Generation limit	50					
Fitness threshold	2.8					
Testing No.	1	2	3	4	5	Average
With hints	13	33	33	51	63	38.60
Without hints	7	9	12	15	28	14.20

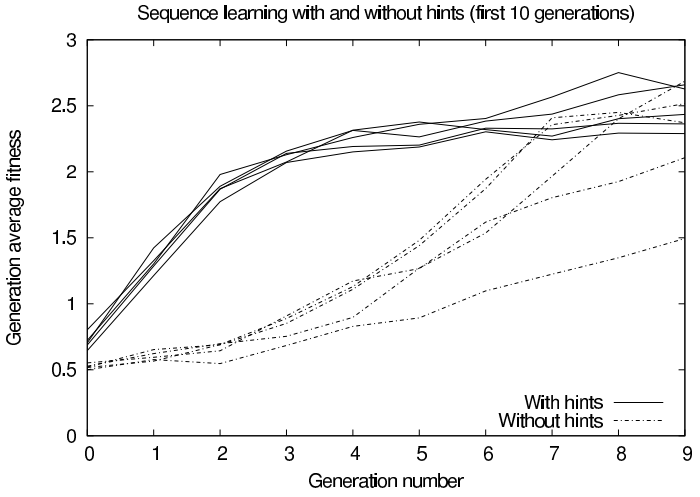

Fig. 7. Average fitness with and without hints (first 10 generations)

From the results shown in Table 7 and Fig. 7, we can have the conclusion that our *SL* does benefit from hints so that it can start faster and get more potentials from the data.

***Intellicut* in Genetic Algorithm** The special genetic operator *intellicut* is another attempt to speed up our genetic algorithm in *SL* and improve the result quality (see Sect. 3.2). The *intellicut* run logs were extracted from the five test runs presented in Table 5 and were analyzed to see the contribution of *intellicut* in our genetic algorithm in *SL*.

Table 8 shows our analysis. The total number of *intellicut* operations that takes place is 1350000 in the five test runs with 20 iterations each and 3 repetitions of the evolutionary process in each iteration. In all these occurrences of *intellicut*, about 40% successfully increase the individual fitness. And the overall average fitness boost in the 1350000 occurrences is 0.5098, from 2.8532 to 3.3630. This shows the introduction of *intellicut* in our sequence learning genetic algorithm is a success.

Table 8. Statistics about *intellicut*

Total number	1350000
Resulting fitness increases	546516
Fitness increase ratio	40.48%
Average fitness before *intellicut*	2.8532
Average fitness after *intellicut*	3.3630

Pre- and Post-learning in the Conjunctive Learner In *CL*, we have pre- and post-learning tasks in addition to the *PART* algorithm (see Sect. 3.2). These tasks were also tested and evaluated.

The pre-learning task is mainly used to reduce the running time of the *PART* algorithm, and to reduce junk predicates in the results. Test runs with and without the pre-learning process were made to see the effects. In our experiments, a *PART* run without pre-learning is only done once because *PART* is an algorithm without any randomness. But we need to run *PART* with pre-learning multiple times because randomness is introduced in our pre-learning task.

Table 9. *PART* tests with and without pre-learning processing

Test	Run time	Average fitness	Average condition size
Without pre-learning	265.811s	1.765	26.854
Pre-learning Average	3.373s	1.655	4.390
With pre-learning 1	3.190s	1.628	4.381
With pre-learning 2	3.096s	1.628	4.381
With pre-learning 3	4.832s	1.710	4.409

In the results of the shown in Table 9, the most obvious difference is the running time. The test run without pre-learning takes much longer than tests with pre-learning. On average, the rules from a *PART* run without pre-learning contain much more predicates. While the average fitness shows not much difference, we can tell many of the predicates in the results without pre-learning are junk predicates. A direct consequence of this is a much longer running time for the post-learning task.

The post-learning task is to increase the coverage of the rules from *PART* and to generalize them. The run logs in the test runs shown in Table 5 were extracted and are presented in Fig. 8. There are 100 pairs of points in the figure, corresponding to the 100 iterations in our five tests, to show the boost of fitnesses in post-learning procedures. We can clearly see that post-learning increases the quality of the rules according to our fitness measure in (1).

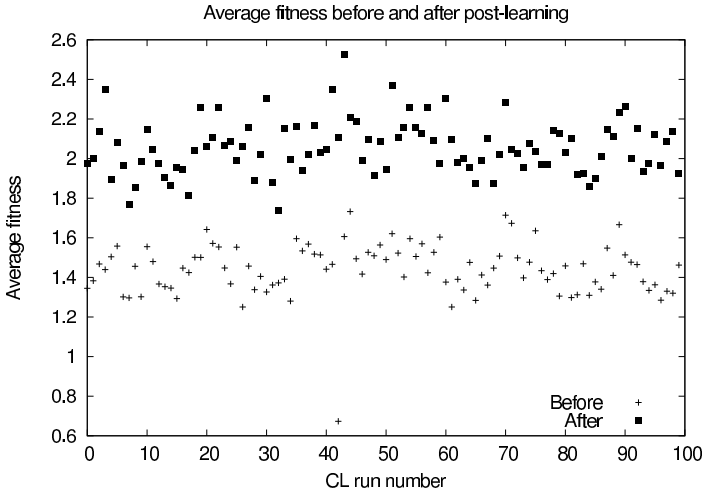

Fig. 8. Post-learning: Increased fitness

4.4 Knowledge Discovered

In this section the discoveries of our test runs on the full data set (with 9450 instances, see Table 4) are examined and analyzed. We find in our results there are not only rules matching known facts about diabetes, but also promising and interesting discoveries that are new to the medical experts.

The most important discovery of our learner is the relation of hypertension and diabetes. In the medical field, it is already known that diabetes has a tight relation with hypertensive diseases. There is a high chance that a diabetes patient also has hypertensive diseases. In our test runs, the diagnoses for hypertensive diseases were identified in many stages. *CL*'s run logs show that the pre-learning task picked "hypertensive diseases" as one of the most discriminating diagnoses. In *SL*, the hypertensive diseases diagnoses appear in over 90% of the best individuals in each generation. And in the final hybrid rule sets 100% of the rules have either "hypertensive diseases" in the sequence part as an indicative event, or "hypertensive diseases=*true*" in the conjunctive part. With such high occurrences of hypertensive diseases diagnoses in our results, it is a very exciting result to the medical experts.

Another discovery is about a general diagnose "signs, symptoms and ill-defined conditions". This diagnosis also has high occurrences in our results. However, unlike the hypertension diagnoses, this diagnose cannot tell us about any specific diseases or disorders, but only some indication in general that the patient does not feel well. In particular, many rules come in the form like Fig. 9, where the "signs, symptoms and ill-defined conditions" diagnoses appear repeatedly in the sequence part. This is an indication that the patients may have been feeling uncomfortable for long before diabetes related diagnoses are made.

And this is not a rare phenomenon. Among all our *cases* there are about 25% who match the rule in Fig. 9.

Part	Condition (ICD-9)	Description
Conjunctive	{466,480-519}=1	Other diseases of the respiratory system
Sequence	{780-799}	Signs, symptoms and ill-defined conditions
	{780-799}	Signs, symptoms and ill-defined conditions
	{780-799}	Signs, symptoms and ill-defined conditions
	{401-405}	Hypertensive disease

Fig. 9. A hybrid rule

Another frequent diagnosis, "other diseases of the respiratory system", has an average of over 80% to appear in all the final hybrid rules (the rule in Fig. 9 is one of them). In medical doctors' eyes, this discovery does not have an obvious explanation (according to the experts we have talked to so far). This should be an interesting topic for the medical experts.

5 Related Work

According to the categories for cooperative search problems discussed in [10] (knowledge-based search is essentially the core in data mining methods), the *CoLe* model has characteristics from both *dividing the problem into subproblems* — split of the heterogeneous data set into simplex data sets and use of multiple miners — and *improving on the competition approach* — result segments from different miners are competing to appear in the combined rules. Depending on the way the heterogeneous data set is split and the strategies for combination in $\mathrm{Ag_{CBN}}$, the *CoLe* model's similarity to *dividing the problem into subproblems* and *improving on the competition approach* may vary.

More importantly, the *CoLe* model emphasizes the combination of the different types of rules, which is not a key characteristic in any of the existing cooperative distributed methods. The combination is not only an effort to make the cooperation more thorough but it also produces more, respectively better knowledge. This can lead to greater synergetic effects.

Compared to some existing applications, the *CoLe* model also has differences and improvements to them.

In [11] a cooperative heterogeneous theorem proving system is presented. This system utilizes two types of provers, namely universal provers and specialized provers respectively. In the proving process, the provers exchange selected clauses periodically with each other and integrate received clauses into their own search states. This system is referred to as the *TECHS* (**TE**ames for **C**ooperative **H**eterogeneous **S**earch) model. Our *CoLe* model has some design similarities with *TECHS*. However, the major difference is that *CoLe*'s combination of rules is

done in a central agent Ag_{CBN}, while in *TECHS* each agent is responsible for its own integration of others' results. The central combination agent in *CoLe* can have a global view on all the different types of results and thus ha more potential to gain good rules through combination.

Viktor et al. developed a cooperative learning framework named Cooperative Inductive Learning Team (*CILT*) in [12]. In this framework, several agents form the cooperative mining team. The agents are either *machine learners* — implementations of machine learning algorithms — or *human learners* — interfaces to human experts. The machine learners each employ a different data mining technique to discover knowledge from data; and the human learners obtain knowledge from human experts. The learning is done in iterations and the whole team will finally come to a knowledge fusion phase to combine the learners' results and form the team rule set. Although in *CILT* different algorithms and different types of learners are used, they all produce the same (compatible) type of results. This leads to very few potentials of knowledge reproduction. In *CoLe*, we consider the results from the learners to be heterogeneous. Combining these heterogeneous results can gain hybrid results that can not be achieved by any single learner.

In both comparisons above, we have a common difference that was not mentioned: *CoLe* proposes the split of a heterogeneous data set into simplex data sets, while in *TECHS* and *CILT*, the data sets are not split. This emphasizes the concept that we should take the characteristics of the data into account and use suitable algorithms for different parts of the data. In this data set definition we can also let the combination agent split the heterogeneous data set dynamically, depending on the specific instantiation of our *CoLe* model.

6 Conclusion and Future Work

We proposed a cooperative multi-agent mining model — *CoLe*. In *CoLe*, multiple learners use different algorithms to mine a heterogeneous data set. These results are sent to a combination agent to create hybrid results and extract useful information as feedbacks to the learners. *CoLe* highlights the interaction and cooperation of different algorithms and the combination of different-typed results from different algorithms, resulting in synergetic improvements against the single algorithms.

We implemented a mining system using our *CoLe* model, aimed at mining knowledge from diabetes data with both temporal and non-temporal information. We use a sequence learner for temporal knowledge mining. This learner contains our genetic algorithm for sequential pattern discovery, with a new knowledge-based genetic operator *intellicut* and targeted beaviours to the mutation genetic operator. Non-temporal knowledge is mined by our conjunctive learner, using an existing implementation of *PART* algorithm, with our pre- and post-learning improvements. The combination agent in this system contains our strategies for combination of the two types of knowledge, as well as feedback generation for cooperation.

Experiments showed clearly how our combined hybrid results enhance and strengthen the raw results from single miners. Detailed particular tests were also made to show that our various knowledge-based targeted strategies played important roles to improve the individual learners' results. Additionally, in the medical field, these results not only confirmed known knowledge about diabetes, namely that hypertension has a tight relation with diabetes, but also found some observations new to the medical experts.

The implemented learning system is the first proof of concept for our *CoLe* model. While it shows good results in the experiments, we have a lot of additional work to do.

First, more implementations of cooperative learning systems using *CoLe* should be made to show that our *CoLe* model is indeed a generic paradigm for various learning/mining problems. In our presented implementation, we only have two learner agents. In future work, we should apply our *CoLe* model to a more complex heterogeneous situation where we have more different algorithms to be introduced into the learning systems. Besides complex heterogeneous data sets, the *CoLe* model is also useful for the situation where the data calls for different levels of abstraction or different levels of emphasis. We can let the learner agents use processed data with different abstraction, or the same algorithm with different parameters to fit such situation.

On the conceptual side, more ways are needed to provide feedback from one learner (or the combination agent) to others. This will make the cooperation more powerful and thus result in greater synergetic effects.

On the medical data mining side, future work should include: using different levels of abstractions on the data; considering both the temporal order and relative time for the diagnoses; making use of other information together with diagnoses; learning on data sets where the distribution of disease cases is more close to general population. The application s of data mining to data on more diseases is also promising.

References

1. Fayyad, U., Uthurusamy, R.: Evolving data into mining solutions for insights. Communications of the ACM **45** (2002) 28–31
2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional (1989)
3. Chan, B., Denzinger, J., Gates, D., Loose, K., Buchanan, J.: Evolutionary behavior testing of commercial computer games. In: Proceedings of the 2004 Congress on Evolutionary Computation, Portland, Oregan, IEEE Press (2004) 125–132
4. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In: Proceedings of the 15^{th} International Conference on Machine Learning, Morgan Kaufmann (1998) 144–151
5. Quinlan, J.R.: C4.5: Programs for Machine Learning. The Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann (1993)
6. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (1999)

7. Liu, H., Lu, H., Yao, J.: Toward multidatabase mining: Identifying relevant databases. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 541–553
8. Karaffa, M.C., ed.: International Classification of Diseases, 9^{th} Revision, 4^{th} Edition, Clinical Modification. Practice Management Information Corp., Los Angeles (1992)
9. World Health Organization: International Classification of Diseases, 9^{th} Revision: Basic Tabulation List with Alphabetical Index. World Health Organization, Geneva (1978)
10. Denzinger, J.: Conflict handling in collaborative search. In Tessier, Chaudron, Müller, eds.: Conflicting Agents: Conflict Management in Multi-agent Systems, Kluwer Academic Publishers (2000) 251–278
11. Denzinger, J., Fuchs, D.: Cooperation of heterogeneous provers. In: Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence (IJCAI'99), Stockholm, Sweden, Morgan Kaufmann (1999)
12. Viktor, H.L., Arndt, H.: Data mining in practice: From data to knowledge using a hybrid mining approach. The International Journal of Computers, Systems and Signals **1** (2000) 139–153