Parallel Inductive Logic Programming System for Superlinear Speedup

Hiroyuki Nishiyama(✉) and Hayato Ohwada(✉)

Faculty of Science and Technology, Tokyo University of Science, 2641 Yamazaki, Noda-shi, Chiba 278-8510, Japan
hiroyuki@rs.noda.tus.ac.jp, ohwada@rs.tus.ac.jp

Abstract. In this study, we improve our parallel inductive logic programming (ILP) system to enable superlinear speedup. This improvement redesigns several features of our ILP learning system and parallel mechanism. The redesigned ILP learning system searches and gathers all rules that have the same evaluation. The redesigned parallel mechanism adds a communication protocol for sharing the evaluation of the identified rules, thereby realizing superlinear speedup.

1 Introduction

Inductive logic programming (ILP) is a superior supervised learning tool. However, learning for large problems usually requires a considerable amount of time. In addition, to generate good rules, continuous learning must be performed while changing the parameters of ILP learning. It is thus necessary to shorten the ILP learning time. To solve this problem, various studies have focused on speeding up ILP using parallel methods [1–3,7,8,12]. Although the problems were partially solved, the process speed was not sufficiently increased based on the number of processors provided, and the quality of the generated rules was not optimal, resulting in difficulty in using it as a practical tool. Skillicorn and Wang [10] succeeded in achieving superlinear speedup, but their method used only four or six CPUs, which performed poorly when the dataset was large.

In this study, we improved our parallel ILP system [7,8] to enable superlinear speedup. We redesigned several features of our ILP learning system [5] and parallel mechanism [7]. The redesigned ILP learning system searches and gathers all rules that have the same evaluation. The redesigned parallel mechanism adds a communication protocol for sharing the evaluation of the identified rules. To estimate the speedup, we used dairy cattle data (e.g. hormones, feed, and activity) to determine successful conditions for artificial insemination [4]. When 30 CPUs were used to solve a large problem (147,992 s for one CPU), we achieved a speedup of 46.85 times (3,159 s). In addition, we applied the parallel ILP system to a very large problem (162,768 s for 10 CPUs). However, the problem could not be solved using one CPU, because it was too large. When 30 and 174 CPUs were used to solve this problem, we achieved a speedup of 3.13 times (51,968 s) and 17.73 times (9,183 s) respectively, based on 10 CPUs. Using our parallel ILP

N. Lachiche and C. Vrain (Eds.): ILP 2017, LNAI 10759, pp. 112–123, 2018.
https://doi.org/10.1007/978-3-319-78090-0_8

system, we thus succeeded in superlinear speedup and demonstrated it to be more useful for large problems. In addition, we obtained identical rules using one CPU, 10 CPUs, 30 CPUs, and 174 CPUs.

2 Improved ILP System

An ILP system finds a hypothesis from a bounded hypothesis space. The ideal hypothesis covers as many positive examples and as few negative examples as possible. Let $p(h)$ and $n(h)$ be the numbers of positive and negative examples covered by hypothesis h. The number of literals in h is denoted by $c(h)$. We express this as follows, where $g(h)$ indicates the generality of h and $f(h)$ indicates the compression.

$$g(h) = p(h) - c(h),$$

$$f(h) = g(h) - n(h)$$

In the ILP system [5], the compression measure $f(h)$ is used to evaluate hypothesis h. If an evaluation (a value of $f(h)$) is the best in a bounded hypothesis space, hypothesis h is the best hypothesis in the space. To avoid noise, the ILP uses two thresholds: *pLimit* and *nLimit*.

$$minimize\ f(h)\ subject\ to\ p(h)\ >=\ pLimit$$

$$maximize\ f(h)\ subject\ to\ n(h)\ <=\ nLimit$$

In addition, the ILP system used a simple set-covering algorithm like that of the typical ILP algorithm [6]:

Step 1: Find a hypothesis using the method mentioned above.
Step 2: Remove the positive examples covered by the hypothesis from the entire set of positive examples.
Step 3: Add the hypothesis found to the set of hypotheses being built (also known as rules), which is initially empty.
Step 4: Repeat step 1 to step 3 until the entire set of positive examples are covered by the rules.
Step 5: Return the rules.

There is a possibility that some hypotheses have the same evaluation, that is, the best value in a bounded hypothesis space in step 1. For example, a hypothesis $h1$ was found and $f(h1)$ was 20; that is, the best value in a bounded hypothesis space ($p(h1)$ >= *pLimit* and $n(h1)$ <= *nLimit*). In this situation, if another hypothesis $h2$ is found and $f(h2)$ is 20, then the same evaluation is the best value. In addition, $h2$ is not identical to $h1$. In this case, the ILP system usually keeps the hypothesis ($h1$) that is found first (hypotheses after the second ($h2$) are dropped). However, the hypotheses after the second ($h2$) are potentially good hypotheses, because the evaluation is identical to the first ($h1$). If the order of finding hypotheses is changed, the identified rules may be changed in the same problem, because we used a simple set-covering algorithm. This algorithm removes positive examples covered by a hypothesis, identified from the entire set of positive examples (in step 2). Therefore, there is a possibility that covered positive examples may be different between when hypothesis $h1$ is identified first and when hypothesis $h2$ is identified first in the same hypothesis space.

Our new ILP system searches and gathers all hypotheses that have the same (best) evaluation. During the search, if our system finds a hypothesis with the same evaluation that is the best value, the system does not drop the hypothesis, but stores it. The system then finds some hypotheses in a bounded hypothesis space. Using this method, the identified rules are never changed in the same problem, even if the order of finding hypotheses is changed. In addition, there is a possibility that the system can reduce the learning time, because the system may remove more positive examples covered by hypotheses from the entire set of positive examples (in step 2), when some hypotheses are found in a bounded hypothesis space (in step 1). For example, we succeeded in reducing the learning time by approximately 10% using dairy cattle data (using a CPU).

3 Improvement of Parallel ILP System

3.1 Previous Parallel ILP System

Our previous research [7] designed and implemented a parallel-processing system for ILP. Figure 1 presents the system, which consists of a master module and worker modules (refer to [7] for details). In our system, the master module does not work for learning, but requests the first task to a worker module using the contract net negotiation protocol [11] and monitors all worker modules. The worker module itself has an autonomous function (unlike MapReduce). When a worker module has no task (e.g. immediately after starting up or completing a task), the worker module accepts a divided task from another worker module using the contract net negotiation protocol and starts the task. When the workload (the number of relationships in the information that the worker must search for) reaches a fixed quantity ($divideNum$), the worker module requests other worker modules to process the divided task (using the contract net negotiation protocol). After the request for the first task issued by a master module is implemented for one worker module, autonomous process distribution begins among worker modules, and all existing worker modules are engaged (saturation of the task). Because the divided task continues to be repeated among worker modules, all worker modules finally complete all processing at approximately the same time. Depending on the size of the fixed quantity ($divideNum$), it will take approximately several seconds. For example, when $divideNum$ is defined as 200 in our implementation, the deviation was less than 1 s, and the master module monitors and finds that all worker's tasks are finished, and receives the processing result (generated rules).

The flow of dividing a space among workers is as follows (step ② in Fig. 1).

- Workers use a branch-and-bound search and increase search *nodes*.
- If the $nodes > divideNum$, a worker requests a subtask (i.e., a part of the nodes: a part of the space) from others ($divideNum$ is a threshold for dividing a task).

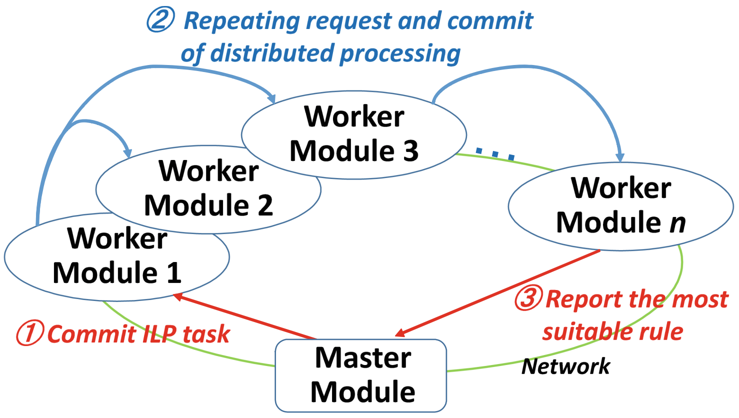

Fig. 1. System configuration of the ILP parallel-computation system and flow of distributed processing.

- A worker accepts the requested subtask if free (i.e., has no task).
- The requesting worker chooses and commits to the accepting worker that first sent the accepting message, and sends the subtask data. If the accepting worker does not have data or the background knowledge, then the requesting worker sends it with the subtask data.

This parallel-processing system has the following merits.

- 1. The master does not need to consider the division of the process in advance.
- 2. All workers work until the end (i.e. no free time) and finish at the same time.

The first merit indicates that the proposed system does not require the master to perform pre-division processing. The second merit means that the process speed can be increased by increasing the number of computers.

3.2 Improvement of the Parallel ILP System Communication Protocol

In the previous parallel ILP system, each worker module has the best evaluation of an identified hypothesis in its own module. Figure 2 illustrates one learning situation of the previous parallel ILP system. The value of the evaluation of the identified interim hypothesis (interim rule) of Worker Module 3 is 20. However, other worker modules have smaller values (i.e. the interim hypothesis is the best hypothesis that satisfies *pLimit* and *nLimit* in the searching process). In the branch-and-bound search, the other worker modules search for useless areas in the bounded hypothesis space. This system speed thus could not be sufficiently increased, even if all worker modules finished at the same time.

Our new parallel ILP system included a new communication protocol for sharing the best evaluation of the identified interim hypotheses (for only sharing

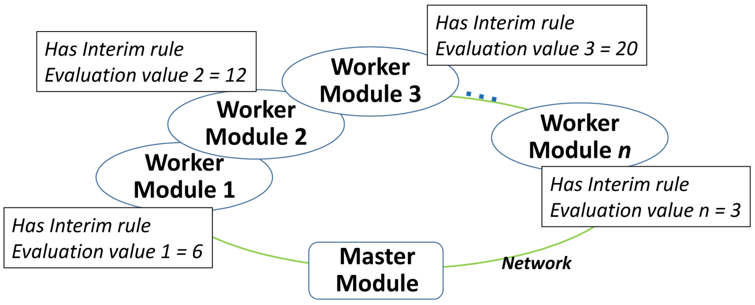

Fig. 2. One learning situation of the previous parallel ILP system. Each worker module has the best evaluation of an identified interim hypothesis.

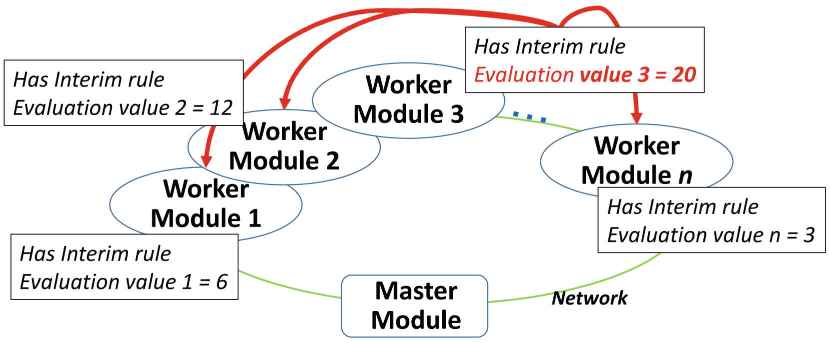

Fig. 3. Worker Module 3 finds an interim hypothesis, and the evaluation is 20, the best evaluation of the module and others. Worker Module 3 then sends the value to all modules to share the best value.

the best evaluation value). In this protocol, when a worker module found an interim hypothesis with a better evaluation, then the worker sent the value to all other workers. Next, when another worker module received a value better than its own, then the worker updates its own value to the better one and drops its identified interim hypotheses. For example, Figs. 3 and 4 depict one learning situation of our new parallel ILP system. Worker Module 3 identifies an interim hypothesis and the evaluation is 20, the best evaluation of the module and others. Worker Module 3 then sends the value to all modules to share the best value (Fig. 3). Then, Worker Module 1, 2 and n receive the value that is better than their own, they update their own value to the better one and drops their own identified interim hypotheses (Fig. 4). This means that the new ILP system can avoid searching useless areas in the bounded hypothesis space. The new system can therefore speed up sufficiently, by increasing the number of computers used.

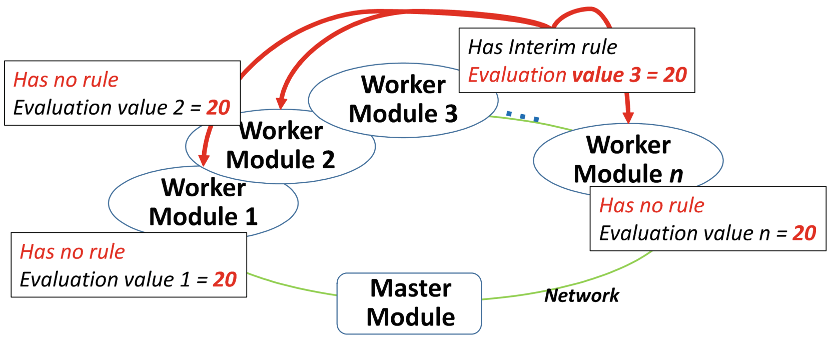

Fig. 4. When a worker module receives a value that is better than its own, its value is updated to the better value. Each worker module then shares the best evaluation, and all worker modules therefore have the same value.

4 Experiment and Results

We implemented the improved parallel ILP system using Java. A total of 30 computers (CPU Core i7 5820 K 6core/12thread 3.3 GHz 64 GBRAM) were used, as seen in Fig. 5, in an experiment to measure speedup using the implemented system. To estimate the speedup, we used data (e.g. hormone, feed, and activity) of dairy cattle to determine the successful conditions for artificial insemination [4].

4.1 Experiment Problems

We used two problems concerning dairy cattle to determine the successful conditions for artificial insemination [4] while estimating the speedup. The first was a large-scale problem that required approximately two days using one CPU. The second was a very large problem that required approximately two days using 10 CPUs. This problem could not be completed using one CPU, because it was too large.

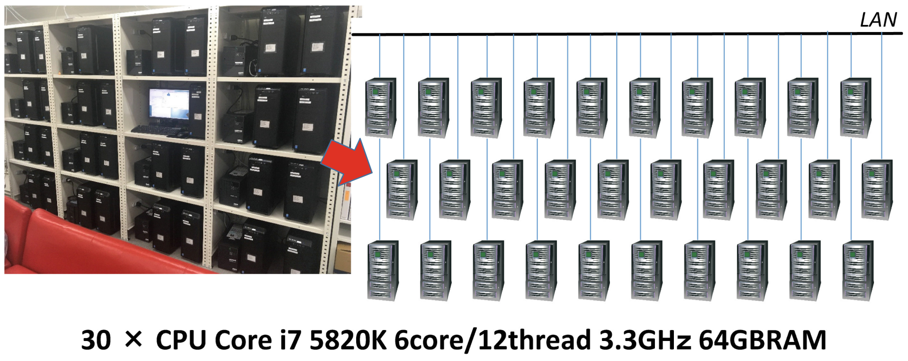

Fig. 5. Experiment environment for the parallel ILP system. The Bio-oriented Technology Research Advancement Institution typically uses this environment for a daily cow project.

Problem 1-Large Problem. The first problem is a large one, consisting of the following learning data:

- Positive examples: 105
- Negative examples: 279
- Kinds of predicates: 6
- Background knowledge size: 49,757 lines

Table 1 lists the predicates and their mode declarations in the background knowledge of this problem. We also define several parameters. *pLimit* is 5, *nLimit* is 0, and max literals in learning is 8. After learning, we obtained 68 rules from this problem.

Problem 2-Very-Large-Scale Problem. The second problem is a very-large-scale one, consisting of the following learning data:

- Positive examples: 101
- Negative examples: 101

Table 1. Predicates and their mode declarations in the background knowledge of the large problem. Mode `+` indicates an input variable, `-` an output variable, and `#` a constant.

Predicates
`progesterone(+cowID, +-preg_datetime, #val),`
`progesterone_diff(+cowID, +-preg_datetime, +-prog_datetime, #time, #val),`
`feed(+cowID, +-feed_datetime, #val),`
`feed_diff(+cowID, +-feed_datetime, +-feed_datetime, #time, #val),`
`sametime(+cowID, +-prog_datetime, +-feed_datetime),`
`birth_num(+cowID, #birth_number)`

Table 2. Predicates and their mode declarations in the background knowledge of the very-large-scale problem. Mode `+` indicates an input variable, `-` an output variable, and `#` a constant.

Predicates
`progesterone(+cowID, +-preg_datetime, #val),`
`progesterone_diff(+cowID, +-preg_datetime, +-prog_datetime, #time, #val),`
`preg_datetime_definition(+cowID, +-preg_datetime, #val),`
`feed(+cowID, +-feed_datetime, #val),`
`feed_diff(+cowID, +-feed_datetime, +-feed_datetime, #time, #val),`
`feed_datetime_definition(+cowID, +-feed_datetime, #val),`
`birth_num(+cowID, #birth_number),`
`activity(+cowID, +-activity_datetime, #val),`
`activity_diff(+cowID,+-activity_datetime,+-activity_datetime,#time,#val),`
`activity_datetime_definition(+cowID, +-activity_datetime, #val)`

– Kinds of predicates: 10
– Background knowledge size: 48,049 lines

Table 2 lists the predicates and their mode declarations in the background knowledge of this problem. In addition, we define several parameters. *pLimit* is 10, *nLimit* is 5, and max literals in learning is 8. After learning, we obtained 168 rules from this problem.

4.2 Experiment Results

Results of Problem 1. Table 3 and Fig. 6 present the results of problem 1, demonstrating that the parallel ILP system accomplished superlinear speedup. In addition, we obtained identical rules using 1 to 30 CPUs.

Table 3. Results of parallel experiments of problem 1.

The Number of CPUs	Execution time (sec.)	Speedup
1	147,992	1.000
2	52,123	2.839
3	32,849	4.505
5	18,859	7.848
10	9,183	16.116
15	6,157	24.036
30	3,159	46.848

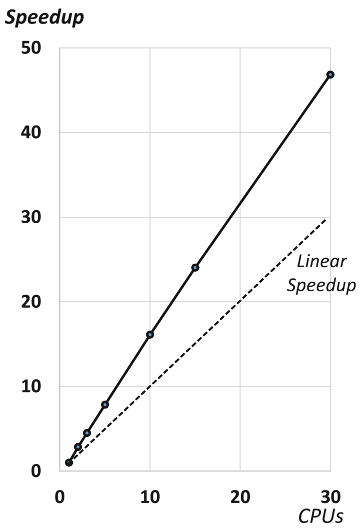

Fig. 6. Graph of parallel experiments of problem 1. The solid line is the result of our parallel ILP system, and the dashed line shows a linear speedup. This graph thus demonstrates that our system accomplished superlinear speedup.

Results of Problem 2. Table 4 and Fig. 7 present the results of problem 2. Our parallel ILP system accomplished a slight superlinear speedup based on 10 CPUs. This means we can apply our system to larger-scale problems by using more CPUs. In addition, we obtained identical rules using 10 to 174 CPUs. The 174 CPUs represent 29 computers with six CPUs each. One computer serves as the master module.

Table 4. Results of parallel experiments of problem 2. The 174 CPUs represent 29 computers six CPUs each. One computer is the master module.

The Number of CPUs	Execution time (sec.)	Speedup (base: 10 CPUs)
10	162,768	1.000
15	105,107	1.549
30	51,968	3.132
174	9,183	17.725

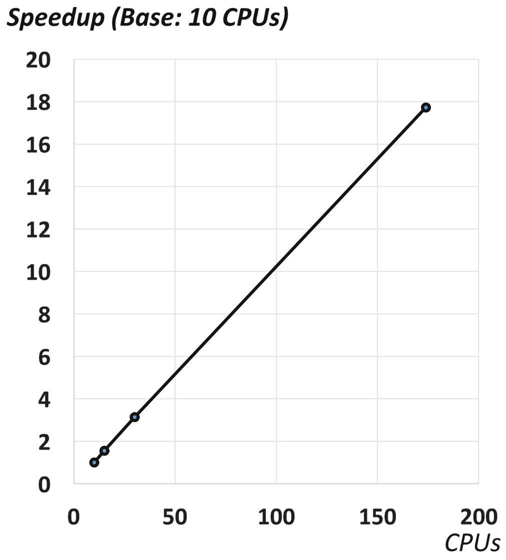

Fig. 7. Graph of parallel experiments of problem 2. The results indicate that our system can be applied to very large problems by using more CPUs.

5 Discussion

5.1 Discussion of Speedup

Our original purpose was to attain linear speedup. However, we succeeded in attaining superlinear speedup and demonstrated it to be more useful in large problems using our parallel ILP system. This demonstrates that all worker modules can divide a learning task and work ideally without searching useless areas in the bounded hypothesis space. In addition, we succeeded in obtaining identical rules by using one to 174 CPUs. This indicates that the rules identified are unchanged in a given problem, even if the order of finding hypotheses is changed.

Figure 8 provides the reason for the speedup of our parallel ILP system. The left side of Fig. 8 depicts the usual parallel method for ILP. This method only divides a space as subtasks among workers. Each worker searches its space only as in Fig. 8. Here, worker C worked much more than other workers. Thus, it is not effective. The right side of Fig. 8 depicts our parallel method, where each worker helps other workers when it completes searching its own space. Using our method, the workers communicate with each other and request or accept a portion of the subtasks. Finally, each worker simultaneously completes all tasks. In addition, our parallel mechanism adds a communication protocol for sharing the evaluation of rules identified. This means that our system can avoid searching useless areas in the bounded hypothesis space. Our system can therefore be speed up sufficiently by increasing the number of computers used.

5.2 Discussion of Speedup Stability

We checked the stability and robustness of our parallel ILP system in another experimental environment, using 48 computers (2vCPU:2.4 GH, 8 GBRAM) on an ATLUS Cloud with 95 CPUs for workers and 1 CPU for the master module. In this experiment, we repeatedly (16 times) executed parallel learning using our system on problem 2. Table 5 presents the results of this experiment. The average time is 20,741.69 s, but the standard deviation is only 29.16 s. This means that our system is very stable for parallel ILP execution.

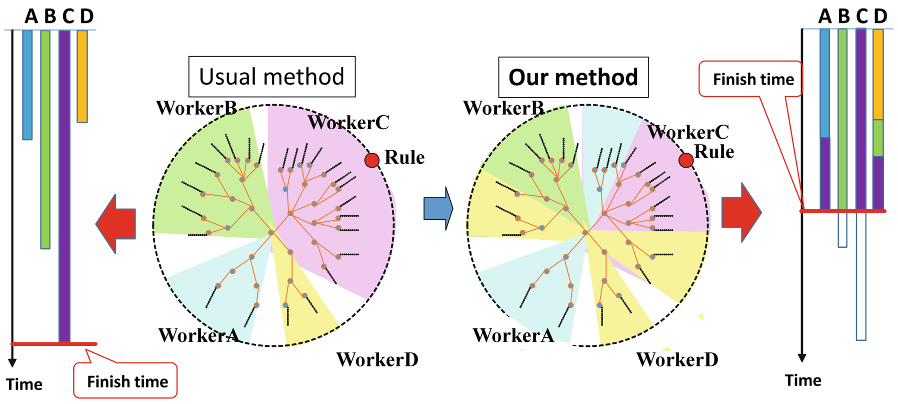

Fig. 8. Comparison between the usual method and our method. The left side depicts the usual method, and the right side depicts our method. Using our method, each worker completes all tasks at the same time.

Table 5. Result of repeated application of the parallel ILP system to problem 2. The result indicates that the speedup is stable.

Number of times	Execution time (sec.)
1	20773
2	20734
3	20747
4	20695
5	20758
6	20774
7	20787
8	20711
9	20755
10	20733
11	20705
12	20736
13	20715
14	20717
15	20738
16	20789
Average	20741.69
Standard deviation	29.16

6 Conclusions

In this study, we improved our parallel ILP system to enable superlinear speedup. This improvement redesigned several features of our ILP learning system and parallel mechanism. The redesigned ILP learning system searches and gathers all rules that have the same evaluation. The redesigned parallel mechanism included a communication protocol for sharing the evaluations of identified rules. To estimate the speedup, we used data from dairy cattle indicating the successful conditions for artificial insemination. Finally, we succeeded in superlinear speedup and demonstrated that it was more useful in large problems. We succeeded in obtaining identical rules using one to 174 CPUs. In addition, we applied our system to the problem of anomaly detection based on the log analysis of a company's server, and succeeded in obtaining valid rules in a realistic time [9]. Currently, we are designing an intelligent grid-search system using this parallel ILP to acquire better rules for dairy-cattle problems.

Acknowledgments. This research was supported by grants from the Project of the Bio-oriented Technology Research Advancement Institution, NARO (the special scheme project on advanced research and development for next-generation technology).

References

1. Fidjeland, A., Luk, W., Muggleton, S.: Customisable multi-processor acceleration of inductive logic programming. In: Latest Advances in Inductive Logic Programming, pp. 123–141 (2014)
2. Fonseca, N.A., Silva, F.M.A., Camacho, R.: Strategies to parallelize ILP Systems. In: ILP 2005, pp. 136–153 (2005)
3. Katzouris, N., Artikis, A., Paliouras, G.: Distributed Online Learning of Event Definitions. Computing Research Repository (CoRR), abs/1705.02175 (2017)
4. Matsumoto, A., Kubota, C., Ohwada, H.: Extracting rules for successful conditions for artificial insemination in dairy cattle using inductive logic programming. In: Proceedings of the 9th International Conference on Machine Learning and Computing, pp. 6–10 (2017)
5. Mizoguchi, F., Ohwada, H.: Constrained relative least general generalization for inducing constraint logic programs. New Gener. Comput. **13**, 335–368 (1995)
6. Mugglenton, S.: Inverse entailment and progol. New Gener. Comput. **13**(3, 4), 245–286 (1995)
7. Nishiyama, H., Ohwada, H.: Yet another parallel hypothesis search for inverse entailment. In: CEUR Workshop Proceedings of the Late Breaking Papers of the 25th International Conference on ILP, vol. 1636, pp. 86–94 (2016)
8. Ohwada, H., Nishiyama, H., Mizoguchi, F.: Concurrent execution of optimal hypothesis search for inverse entailment. In: Cussens, J., Frisch, A. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 165–173. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44960-4_10
9. Shimada, T., Hatano, R., Nishiyama, H.: Server failure detection system based on inductive logic programming and random forest. In: Proceedings of 33rd International Conference on Computers and Their Applications, CATA 2018, March 2018
10. Skillicorn, D.B., Wang, Y.: Parallel and sequential algorithms for data mining using inductive logic. Knowl. Inf. Syst. **3**(4), 405–421 (2001)
11. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. **C-29**(12), 1104–1113 (1980)
12. Fonseca, N.A., Srinivasan, A., Silva, F., Camacho, R.: Parallel ILP for distributed-memory architectures. Mach. Learn. J. **74**(3), 257–279 (2009)