Efficient Inductive Logic Programming Based on Particle Swarm Optimization

Kyosuke Obara[1(✉)], Munehiro Takimoto[1], Tsutomu Kumazawa[2], and Yasushi Kambayashi[3]

[1] Department of Information Sciences, Tokyo University of Science, Chiba, Japan
6322510@ed.tus.ac.jp, mune@rs.tus.ac.jp
[2] Software Research Associates, Inc. Toshima-ku, Tokyo, Japan
kumazawa@sra.co.jp
[3] Department of Computer and Information Engineering, Nippon Institute of Technology, Saitama, Japan
yasushi@nit.ac.jp

Abstract. Inductive Logic Programming (ILP) is an inductive reasoning method based on the first-order predicative logic. This technology is widely used for data mining using symbolic artificial intelligence. ILP searches for a suitable hypothesis that covers positive examples and uncovers negative examples. The searching process requires a lot of execution cost to interpret many given examples for practical problems. In this paper, we propose a new hypothesis search method using particle swarm optimization (PSO). PSO is a meta-heuristic algorithm based on behaviors of particles. In our approach, each particle repeatedly moves from a hypothesis to another hypothesis within a hypothesis space. At that time, some hypotheses are refined based on the value returned by a predefined evaluation function. Since PSO just searches a part of the hypothesis space, it contributes to the speed up of the execution of ILP. In order to demonstrate the effectiveness of our method, we have implemented it on Progol that is one of the ILP systems [6], and then we conducted numerical experiments. The results showed that our method reduced the hypothesis search time compared to another conventional Progol.

Keywords: Inductive logic programming · Particle swarm optimization · Progol

1 Introduction

Finding a general theory from concreate examples has been the mainstream of scientific world. Data mining is a recent popular technique for this purpose. A scientist who is engaging in data mining is called a data scientist, and it is a popular occupation now. The spread of machine learning including deep learning has contributed to a lot of data mining through generating structured data

Y. Kambayashi et al. (Eds.): AICON 2022, LNICST 477, pp. 151–158, 2023.
https://doi.org/10.1007/978-3-031-29126-5_13

such as data bases from unstructured data such as images and sounds. On the other hand, data scientists have not sufficiently investigated the relations among structured data due to the nature of machine learning. To extract knowledge in the structured data from a single database or inter-related multiple databases, Inductive Logic Programing (ILP) [5] is known to be useful. Therefore, ILP is expected to be effective for data mining. ILP is an inductive reasoning methodology based on the first-order predicate logic. It tries to induce a general theory from specific examples. ILP searches for a simple hypothesis that covers positive examples and uncovers negative examples based on background knowledge. ILP generates the hypothesis candidates through generalizing positive examples, and then it verifies each candidate through covering positive examples and uncovering negative examples. ILP deals with the hypothesis as rules of the first-order predicate logic. A hypothesis is added to background knowledge, so that the background knowledge is extended. If all the examples with the same predicate as the hypothesis are deduced from the extended background knowledge, we can confirm that the induced hypothesis is valid. This validation process is a simple deductive reasoning practiced in the standard logic programming, and it shows that the examples are correct or incorrect.

Thus, when we pursue the explainable artificial intelligence, ILP is a strong data mining tool. It implements the explainable symbolic artificial intelligence. On the other hand, ILP tends to require a lot of execution time in practical cases. We can reduce the execution cost by parallelizing covering examples or searching hypotheses. However, parallelization requires other computational resources, and limits the effectiveness of such approaches. In this paper, we propose a method efficiently detecting a hypothesis using Particle Swarm Optimization (PSO) [4]. We call our method PSOProgol. It stands for a PSO based Progol.

PSO is a meta-heuristic algorithm inspired by the behavior of birds and fishes. The meta-heuristic algorithms can find semi-optimal solutions in the huge search space with less time and memory than exhaustive methods can do. As an approach similar to our method, i.e. ILP with meta-heuristic, Ant-FOIL has been proposed [12]. Ant-FOIL is an ILP system built on the top of FOIL [11] and it takes advantages of the ant colony optimization (ACO). FOIL is an ILP system based on a downward refinement operator. Refinement operators are frequently used in ILP. ACO is a meta-heuristic algorithm inspired by behaviors of ants [2]. Ant-FOIL contributes to improving predictive accuracy but not its performance. It has the same limitation as the original FOIL; it uses the base atoms as its available background knowledge as the original FOIL does. We have built a new ILP system on top of Progol. Progol can use any Horn clause as background knowledge [6]. In terms of meta-heuristic algorithms, PSO is often used to find the location of a target when only the distance to the target is known, while ACO is often used to find the shortest path when the target location and distance to it are known. Therefore, we adopt PSO to Progol. When we have applied PSOProgol to three public problems, we have observed that the bigger the problems we have to tackle, the more efficiency we can get to obtain hypotheses compared to the conventional Progol.

The rest of this paper is organized as follows. Section 2 describes a brief introduction to Progol as a representation system of ILP systems, and Sect. 3 describes PSO. Section 4 gives a detailed description of our proposed method. Section 5 presents our experimental results, and Sect. 6 gives conclusions and future work.

2 Progol

Figure 1 shows the outline of Progol's learning algorithm [6]. B is a set of the background knowledge, E is a set of positive examples, and H is a set of hypotheses. Most Specific Hypothesis (MSH) is a hypothesis that has all possible literals, and all the literals of each generated hypothesis candidate construct a subset of ones of the MSH. MSH is generated by generalizing each positive examples (line 2 in Fig. 1). Progol generates hypothesis candidates by adding some literals in MSH to an empty clause. What hypotheses are generated depends on which literals are added to the basic candidate. Focusing on the relation between the base candidate and newly generated candidate, we can represent the hypothesis space as a directed acyclic graph (DAG), as shown in Fig. 2. The process of generating a more specific hypothesis candidate through adding some literals to the base candidate is called refinement. In the refinement, selecting which hypothesis to refine affects accuracy and efficiency of the search. Progol tries to find the simplest hypothesis, which covers all the positive examples and covers as few negative examples as possible, in the top-down direction in the DAG structure while checking each candidate with an evaluation function. Notice that hypothesis candidate is not as specific as MSH, which is the bottom of the DAG structure. Once a suitable hypothesis is found (line 3), it is added to the background knowledge (line 4), and the positive examples covered by it are eliminated from the positive examples to generalize the hypothesis (lines 5 and 6). Thus, lines 1–7 are repeated until the set of positive examples for generalization becomes an empty set (line 1). Finally, the set B returned in line 1 becomes the background knowledge with the hypothesis covering all the positive examples.

Progol's evaluation function f is expressed by the following equation:

$$f = p - (n + c + h) \tag{1}$$

Each variable is defined as follows. p is the number of positive examples covered by the candidate hypothesis and n is the number of negative examples covered by the candidate hypothesis. c is the number of literals in the hypothesis body part, and h is the minimum number of the literals that should be added to complete the input-output relationships of the candidate hypothesis. All the hypotheses with the complete input-output relationships have the corresponding input and output variables in the head literal. They have either the same input and output variables in the head or chained ones in the body of the hypothesis. We call the length of the chain the length of the hypothesis. Thus, h represents the distance to the target hypothesis.

1: if $E = \emptyset$, return B
2: Generate MSH from the beginning e of E.
3: A^*-like search algorithm generates H.
4: $B \leftarrow B \cup H$
5: $E' \leftarrow \{e' : e' \in \mathrm{E}$ and B implies $e'\}$
6: $E \leftarrow E \backslash E'$
7: go to 1

Fig. 1. Progol Algorithm

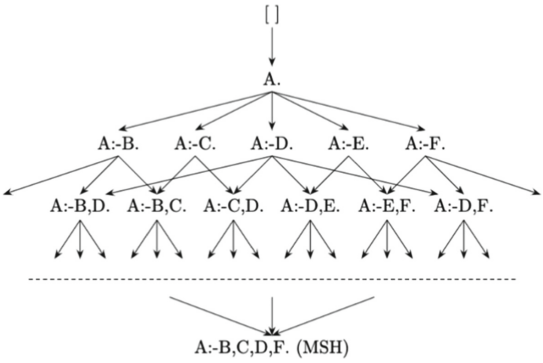

Fig. 2. Progol's Hypothesis Space

3 Particle Swarm Optimization

In PSO, multiple particles search for semi-optimal solutions while moving in a given search space. Each particle i has information on its position x_i, velocity v_i, and the local best position $pBest_i$ with the highest evaluation value for the particle. In addition, particles share the global best position $gBest$ with the highest evaluation value in the entire swarm. The i-th particle changes its velocity vector with the values of $pBest_i$ and $gBest$. As shown in Fig. 3, the position at time $(t + 1)$ is closer to the direction of $pBest_i$ and $gBest$ than the position at time t. PSO tries to avoid the local maxima through considering not only $gBest$ but also $pBest_i$. The velocity and position of each particle are determined by the following formulae:

$$v_i = w \cdot v_i + c_1 \cdot rand_1 \cdot (pBest_i - x_i) + c_2 \cdot rand_2 \cdot (gBest - x_i) \tag{2}$$

$$x_i = x_i + v_i \tag{3}$$

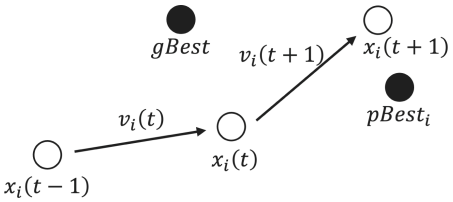

Fig. 3. Particle Movement

The velocity of each particle is updated through Eq. (2) at each step. Values $rand_1$ and $rand_2$ are randomly decided at each step respectively. Coefficients w, c_1 and c_2 are empirically determined. Once the velocity is updated, the position of the particle is updated through Eq. (3).

4 PSO Based Progol

Original Progol performs hypothesis search by A*-like algorithm. This algorithm mimics the A* algorithm but efficiently finds the optimal solution. It is a kind of heuristics sacrificing strict optimality in order to get efficiency. Thus, although the search strategy of Progol aims at efficiently finding a solution, it still takes long time in some practical cases. In order to mitigate this problem, PSOProgol uses PSO for its hypothetical search. PSO has an advantage for searching in a huge hypothesis space. In PSOProgol, a particle swarm moves through the hypotheses in the search space and refines the hypothesis with the best evaluation value. Since Progol tries to maximize the evaluation function (1), the ILP problem can be considered as the optimization problem that maximizes the value of (1). Thus, PSOProgol employs the function (1) as its fitness function and tries to maximize its value. Therefore, $pBest_i$ and $gBest$ can be related to Progol's evaluation function, and the velocity and position of the particles can be updated based on the evaluation of the function. The depth of MSH can be computed beforehand and PSOProgol assigns this depth as the dimension of position vectors. Note that the precomputation of the maximum depth is the main reason why PSO is suitable with ILP. Also, in order to relate the ILP hypothesis space to PSO search space, each hypothesis on a path from the root to MSH must be related to each component of a position vector in the PSO search space in accordance with the manner of [3]. The relation between paths in the hypothesis space and positions in the PSO search space is shown in Fig. 4. Each path on the DAG in Fig. 4 can be represented by a sequence of indices that are labeled on edges in the figure. The first component of the position vector represents the depth of the hypothesis, and the vector's dimension is "The depth of MSH + 1". The position vectors' components that do not indicate any edge indices are initialzed to 0. For example, the coordinate (2,1,2,0,...,0) corresponds to the hypothesis "A:-C" and the coordinate (3,1,1,2,0,... 0) corresponds to the hypothesis "A:-B,D".

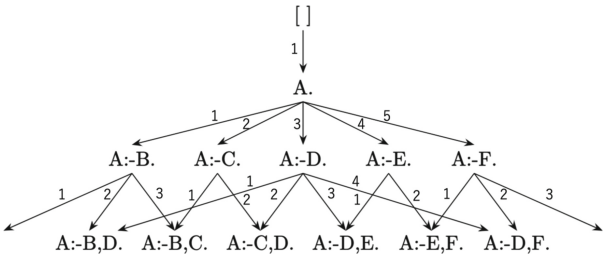

Fig. 4. The hypothesis space with paths

Figure 5 shows the algorithm of PSOProgol. First, initialize the particle swarm(line 1 in Fig. 5). The initial positions of all particles are the coordinates of the hypothesis "[]" and the initial velocity is 0. Next, the positions and velocities are updated according to Eq. (2) and (3) (line 2). The hypothesis corresponding to each particle's position is refined and the generated hypothesis is added to the hypothesis space(line 3). The $pBest_i$ and $gBest$ are updated based on the evaluated value of the hypothesis corresponding to the position of each particle after its movement(line 4 and 5). The above process is repeated until the termination condition is met. The termination condition is that the positive example set is empty, as in Progol.

1 Initialize the position and velocity of each particle.
2 Update the position and velocity of each particle.
3 Refine the hypothesis of the node to which each particle is moving and add it to the hypothesis space.
4 Update the pBest
5 Update the gBest
6 If termination conditions are met, output gBest and exit.
7 Repeat steps 3 through 6.

Fig. 5. Algorithm of PSOProgol

5 Experiments

In order to demonstrate the effectiveness of our approach, we have implemented PSOProgol in OCaml 5.0.0. We then conducted numerical experiments and evaluated it's execution time and compared the corresponding values to the conventional Progol's for three datasets, Grammar, Animals and Append [8]. These are public-domain benchmark datasets provided by the Progol developers. Grammar checks whether a given English sentence is grammatically correct or not, Animals classifies a given animal, and Append concatenates two lists. We took the average of 30 experiments for each dataset because both Progol and PSOProgol employ randomized approaches. In Progol, negative examples are automatically generated based on stochastic logic programs [7]. In PSOProgol, Formula (2)

contains random numbers. The parameters of the PSO were set as per Table 1. These values were determined through several preliminary experiments. Since the initial position of all particles is set to the most general and empty hypothesis, the value of w was set to a larger value of 1.5 so that they disperse more quickly. We have conducted experiments on Apple M1 with MacOS Monterey version 12.4 and 8 GB memory.

As shown in the Fig. 6, conventional Progol was a little faster than PSOProgol in Append, but in Grammar and Animals, PSOProgol was faster than conventional one.

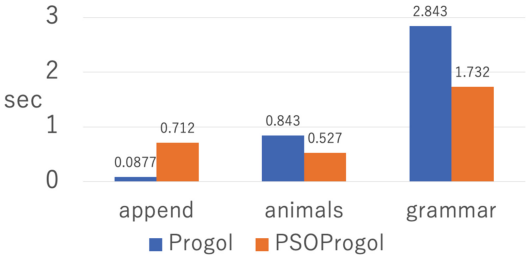

Fig. 6. Execution time for each problem

Table 1. Parameters of PSOProgol

Parameter	Value
w	1.5
c_1	1
c_2	1
Maximum Depth	Depth of MSH
Number of Particles	45

While PSOProgol is less efficient for small hypothesis spaces such as Append because of the time it takes to move multiple particles and the random numbers involved, it is more efficient in huge hypothesis spaces such as Grammar and Animals as well as PSO. Regarding accuracy, the hypotheses generated by Progol and PSOProgol were identical. The experiments show that PSOProgol becomes faster than Progol in hypothesis search as the size of the hypothesis space increases and has an advantage in practical cases.

6 Conclusions and Future Work

We have proposed a new ILP system PSOProgol that combines a heuristic algorithm PSO and a broadly descriptive algorithm Progol. Through the numerical experiments, we have observed that PSOProgol is faster than conventional

Progol when the hypothesis space enlarges. Therefore, the proposed method is effective in data mining, where the data may be huge and faster execution speed is required. As future work, we plan to extend PSOProgol using parallel and distributed processing [1,9,10]. Specifically, further speed-up can be expected by dividing the search space and separating particles into multiple groups.

References

1. Algahtani, E., Kazakov, D.: GPU-accelerated hypothesis cover set testing for learning in logic. In: Riguzzi, F., Bellodi, E., Zese, R. (eds.) ILP Up-and-Coming/Short Papers. CEUR Workshop Proceedings, vol. 2206, pp. 6–20. CEUR-WS.org (2018)
2. Dorigo, M., Stuetzle, T.: Ant Colony Optimization. The MIT Press, Cambridge (2004)
3. Ferreira, M., Chicano, F., Alba, E., Gómez-Pulido, J.A.: Detecting protocol errors using particle swarm optimization with Java pathfinder. In: Smari, W.W. (ed.) Proceedings of the High Performance Computing & Simulation Conference (2008)
4. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995). https://doi.org/10.1109/ICNN.1995.488968
5. Muggleton, S.: Inductive logic programming. New Gener. Comput. **8**, 295–318 (1991)
6. Muggleton, S.: Inverse entailment and progol. New Gener. Comput. Spec. Issue Induct. Logic Program. **13**(3–4), 245–286 (1995). https://citeseer.nj.nec.com/muggleton95inverse.html
7. Muggleton, S.: Stochastic logic programs (1996)
8. Muggleton, S.: Progol (2003). https://www.doc.ic.ac.uk/shm/progol.html
9. Nishiyama, H., Ohwada, H.: Parallel inductive logic programming system for superlinear speedup. In: Lachiche, N., Vrain, C. (eds.) ILP 2017. LNCS (LNAI), vol. 10759, pp. 112–123. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78090-0_8
10. 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
11. Quinlan, J.R.: Learning logical definitions from relations. Mach. Learn. **5**, 239–266 (2004)
12. Yan, C.: Ant-FOIL: integrating ant colony system and FOIL. In: 2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics, vol. 1, pp. 559–562 (2015). https://doi.org/10.1109/IHMSC.2015.20