Extending Inductive Generalization with Abduction

Takashi Kanai and **Susumu Kunifuji**
School of Information Science,
Japan Advanced Institute of Science and Technology
1-1 Asahidai Tatsunokuchi Noumi Ishikawa Japan
email:{kanai, kuni}@jaist.ac.jp

Abstract

In this paper, we propose a new integrated method of inductive generalization and abductive reasoning. This approach can deal with the situations when background knowledge is insufficient to induce definitions of the given examples. Most existing ILP systems implicitly assume that given background knowledge is enough to induce definitions of the given examples, but background knowledge is not usually supplied sufficiently in real-world situations. In order to solve this difficulty, this paper investigates an extended framework of Inductive Logic Programming which uses the methods in Abductive Logic Programming.

1 Introduction

Many researches on Inductive Logic Programming(ILP) [Muggleton and de Raedt, 1994] focus on inductive generalization, which generates a hypothesis that can be viewed as generalization of examples. For the sake of this purpose, many researches assume that background knowledge is sufficient to induce the rules which explains given examples. But this assumption is not usually satisfied because it is hard to provide background knowledge sufficiently in real-world situation.

On the other hand, Abductive Logic Programming(ALP) [Kakas *et al.*, 1992] is a hypothetical reasoning framework that hypothetical language is a set of ground atoms, and the representation language is a logic program. ALP has a remarkable property that ALP can be viewed as the inductive inference that is not categorized as inductive generalization. One of difficulties of ALP is that the explanation of a given example must be a set of ground atom, and it cannot deal with general Horn clause.

This paper proposes a new framework of inductive generalization, where background knowledge is not sufficiently given, background knowledge is supplemented dynamically by using abductive reasoning. We introduce an inductive generalization method with abduction, and compare our method to other unified frameworks of inductive generalization and abduction.

2 Induction with abductive reasoning

This section introduces the outline of our inductive method with abduction, and then shows how our method works by an example.

2.1 Problem setting

Both frameworks of ALP and ILP have their own problem setting. To construct an integrated framework of ALP and ILP, a proper common problem setting is needed. In this paper, we formalize the problem setting as follows.

Given:

- (insufficient) Background knowledge P
- Abducibles Ab
- A set of integrity constraint I
- Oracle Q
- A set of positive and negative examples E

Find:

A set of logic program H and a set of ground atom $\Delta \subseteq Ab$ which satisfies

- $P \cup H \cup \Delta \models E$
- $P \cup H \cup \Delta \cup I$ is consistent

2.2 Outline of our approach

The following is the outline of our method of induction with abduction.

- Top-down search strategy

  Our system adopts a top-down search method which uses FOIL[Quinlan, 1990]-like heuristics and greedy searching approach. The bottom-up approaches are not adequate for dealing with incomplete background knowledge. The bottom-up approaches generates the relative least general generalization(RLGG) using the given background knowledge. But in our problem setting, the given background knowledge are not given sufficiently in order to induce the rules to explain the given examples. Therefore RLGG cannot be created in our problem setting.

- Abduction as Example Generation

  If background knowledge is insufficient to check hypotheses whether it implies the given examples, usual inductive techniques do not work properly. In this situation, abduction is provoked to supplement the background knowledge, and the generated explanations are stored as examples. This behavior can be recognized as example generation.

- Insufficient background knowledge is declared as abducibles

  In our framework, insufficient part of background knowledge must be declared as abducibles[Kakas *et al.*, 1992]. This declaration means that insufficient part of background knowledge should be recognized in advance.

- Type and mode declarations

  In order to restrict the hypothesis space, We provide a restriction that every abducible predicate has mode and type declarations.

The following is a summary of our refinement operator. In the definition of the procedure, we assume that every abducible predicate has mode and type declarations and the abductive proof procedure in the refinement operator is a modified Kakas-Mancarella procedure which is extended to allow cost based abduction. Note that this procedure is an example of refinement operator, and other refinement operators can be extended in the same way.

Let C be a partial clause.

- If C is $\square$
  1. Choose two positive examples(F_1, F_2).
  2. Compute LGG of these two examples.Let h be the LGG atom.
  3. Translate all arguments of h to the MGU of arguments. MGU is defined as follows,
     - If T is a function,th MGU of T is TN.

       ```
       mgu(T,TN) :- functor(T,F,N),
            functor(TN,F,N).
       ```

     - Otherwise the MGU of T is T.

     Let h^* be the translated atom.
  4. Retain F_1 and F_2.
  5. Return the clause "$h^* : -true$"
- Otherwise(C is not $\square$)

  Let $C = h : -l_1, l_2, \ldots, l_n$
  1. For all predicate name, do the following:
     (a) Get type and mode declaration. Let L be the MGU of the predicate.
     (b) Collect all of the terms that are occurred in C.
     (c) Substitute input arguments of L to collected terms.
     (d) Verify type and mode constraint of L.
     (e) If the literal is a recursive literal,then check whether the variables which is occurred in the input arguments of the literal are occurred at same position of the head of the clause.
     (f) Compute the gain of $h : -l_1, l_2, \ldots, l_n, L$ using the abductive proof procedure without oracle(discussed later).
  2. Choose a literal which has a maximum gain. Let L^* be the literal which has a maximum gain.
  3. Check that F_1 and F_2 satisfy $h : -l_1, l_2, \ldots, l_n, L^*$ using the abductive proof procedure using oracle.
  4. Return $h : -l_1, l_2, \ldots, l_n, L^*$

2.3 Evaluation of hypotheses using abduction

Heuristics is used to estimate the goodness of generated hypotheses. Usually, the heuristics is implicitly assumed to use in the situation that background knowledge is sufficient to explain the given examples. If this assumption is not satisfied, the common heuristics cannot estimate the goodness of a hypothesis, because ILP systems need background knowledge to check a hypothesis whether it implies the given examples. In fact, heuristics may lead to a wrong hypothesis. In addition, the truth of unknown atoms should not be asked to the user indiscriminately because the number of oracles becomes too large. Therefore, the heuristics should cope with insufficient background knowledge in our problem setting.

Our system adopts an improved FOIL's evaluation of a hypothesis which can be used in abductive framework. FOIL adds the most gainful literal one by one to partial clause. Information heuristics is used in this step to find the most gainful literal in FOIL. Literals are evaluated as follows. Let the number of + (-) bindings of a partial clause be $n^+(n^-)$. The average information that one of the bindings has label + is

$$I(n^+, n^-) = -\log_2 n^+/(n^+ + n^-)\ bits$$

If a literal L is added,let k be the n^+ bindings which are not excluded by L , the numbers of bindings of the new partial clause be $m^+(m^-)$. Then the total information gained by adding L can be represented as

$$k \times (I(n^+, n^-) - I(m^+, m^-))\ bits$$

But FOIL's heuristics uses the number of positive (negative) examples which are explained by a current hypothesis. However in the insufficient background knowledge settings, the number of positive(negative) examples cannot be computed correctly if the added literal L requires the incomplete part of background knowledge to check whether L holds or not.

Our improvement of FOIL's heuristics is to count the sum of minimum cost of examples instead of the number of positive(negative) examples. The following function is

used instead of the number of positive(negative) literals.

$$n^+ = \sum_{e \in E^+} 2^{-cost(e)}$$

where $cost(x)$ is the cost function that returns the minimum cost to explain x. The $cost(x)$ equals to zero iff an explanation of x is empty set, and $cost(x)$ is ∞ iff x cannot be explained. Other part of evaluation process remains unchanged. This evaluation is compatible to FOIL's evaluation method when e has an explanation which is an empty set, therefore this modification is an extension to induction based on abductive framework. This modification is also applied to the evaluation of negative assumptions, because our abductive procedure deals with negative literals as abduction [Eshghi and A., 1989].

In the above definition, the cost of explanation (which is generated by an abductive procedure) is used to calculate the minimum cost of the explanation of given examples. In ALP, proof procedure does not take account of the cost of an explanations.

The procedure which finds the minimum cost explanation can be constructed by using iterative deepening search method. In order to implement an iterative deepening search method for finding the minimum cost explanation, representing the cost of the proof and limited cost search method is needed. The implementation of abductive procedure which limits the cost is trivial. Due to the limitation of space, we omit the detail of the procedure.

In order to represent the cost of explanation, a built-in predicate $costly/1$ is introduced. The procedural meaning of $costly/1$ is that $costly(X)$ is true iff '$current\ cost' - X$ is less than 0. $costly/1$ is usually used for the addition of cost to the current proof in integrity constraints. For example,the integrity constraint $\leftarrow abd^*, costly(3)$ means that the abducibles abd^* cost 3.

2.4 Example: lamp diagnosis problem

Consider the following fault diagnosis problem of lamp which is modified from [Ade and Denecker, 1995].

> Let l_1, l_2, l_3, l_5 be lamps,l_4 be not a lamp. Each device $\{l_1, \ldots, l_5\}$ is connected to battery $\{b_1, \ldots, b_5\}$. Each battery is empty except for l_2 . It is known that the code of l_3 is not likely to break.
>
> Problem: Generate the programs to explain the trouble of the lamp $l_1, \ldots, l_5$.

Mode and Type declaration:

```
faulty_lamp(+atomic). lamp(-atomic).
empty(+atomic). break_code(+atomic).
conn_to_battery(+atomic,-atomic).
conn_to_empty_battery(+atomic,-atomic).
```

Background Knowledge:

```
lamp(l1). lamp(l2). lamp(l3). lamp(l5).
conn_to_battery(l1,b1). conn_to_battery(l2,b2).
conn_to_battery(l3,b3). conn_to_battery(l4,b4).
conn_to_battery(l5,b5).

empty(b1). empty(b3). empty(b4). empty(b5).
```

Integrity Constraints:

```
<-- break_code(l3),costly(3).
% every abduced atom costs 1
<-- _, costly(1).
```

Observations(examples):

```
faulty_lamp(l1). -faulty_lamp(l2).
faulty_lamp(l3). -faulty_lamp(l4).
faulty_lamp(l5).
```

Note that the abducible 'break_code(l3)' costs 4 because if 'break_code(l3)' is assumed to be true, costly(3) and costly(1) are invoked simultaneously. Other abducibles(including negative literals) costs 1.

The structure of the correct programs are shown in Figure 1 .

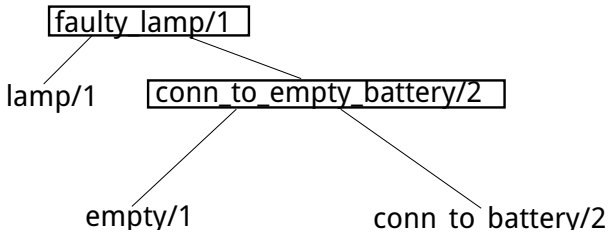

Figure 1: structure of expected programs

Our system works as follows.

Phase 1: Inducing faulty_lamp/1

1. Make initialization and a user controls the system to induce the definition of faulty_lamp/1.
   clause: faulty_lamp(A) :- true
2. ' lamp(A) ' is added to the partial clause, because the addition of lamp(A) to the partial clause excludes the negative example faulty_lamp(l4) only. Therefore the literal lamp(A) is the most gainful literal.
   clause: faulty_lamp(A) :- true,lamp(A)
3. At this time, conn_to_empty_battery(A,B) is most gainful literal, because the cost of assuming break_code(l3) is 4, but the cost of assuming conn_to_empty_battery(l3, B) is 1, and the literal empty(A) excludes all examples, and the gain of adding the literal conn_to_battery(A, B) is 0.
   The system selects the literal and asks the value of conn_to_empty_battery(l1,B) and conn_to_empty_battery(l3,B).
   The user answers conn_to_empty_battery(l1,b1) and conn_to_empty_battery(l3,b3) are true. Then conn_to_empty_battery(A,B) is added to partial clause.
   clause: faulty_lamp(A) :- true, lamp(A), conn_to_empty_battery(A, B).

4. The system asks the correctness of the generated clause to the user. The system also asks the value of conn_to_empty_battery(l2,B), the user indicates that conn_to_empty_battery(l2,b2) is false.

Note that the system generates two different hypotheses, one is the definition of faulty_lamp/1 and the other is the example of conn_to_empty_battery/2. The generated examples are used in the next phase.

Phase 2: Inducing conn_to_empty_battery/2

1. The user indicates the system to create the definition of conn_to_empty_battery/2
2. The system finds that conn_to_empty_battery(A,B) is most gainful literal. Then add the literal to the partial clause.
   clause: conn_to_empty_battery(A, B) :- true, conn_to_battery(A, B)
3. The system finds that empty(B) is the gainful literal. Then add the literal to the partial clause.
   clause: conn_to_empty_battery(A, B) :- true, conn_to_battery(A, B), empty(B).
4. The system asks the user whether the generated clause is true or not.
5. The generated clause is a correct definition, and the user answers 'yes' , then all examples are explained by generated hypotheses.

In this phase, the system creates the definition of conn_to_empty_battery/2, which is the insufficient part of the given background knowledge.

3 Comparison to other researches

In this section, we compare our method to other integrated frameworks of ALP and ILP such as SLDNFAI, inverse resolution and abductive concept learning.

3.1 Comparison to SLDNFAI

SLDNFAI[Ade and Denecker, 1995] is an abductive and inductive procedure which is used in the abductive proof procedure SLDNFA[Denecker and Danny, 1992]. SLDNFAI first produces an explanation of the given goal, then generates a set of clauses which is induced the explanation as examples.

The differences between our framework and SLDNFAI are:

- Generation of examples

  SLDNFAI firstly produces the explanation of an abductive goal, then executes their inductive generalization using generated explanation as its example.

  In our approach, the explanations are firstly produced using the given examples, then inductive generalization is invoked. Therefore our approach can retain more examples(which is generated as explanations) than SLDNFAI approach.

- Plausibility of explanation

  Our approach uses cost based abduction in the process of specializing hypotheses, this leads to the plausible hypothesis correctly. SLDNFAI cannot deal with the plausibility of explanation.

- Relation of abduction and induction

  SLDNFAI calls induction procedure in abductive procedure, but our approach executes the process of abduction in induction process.

3.2 Comparison to inverse resolution

Inverse resolution[Muggleton and Buntine, 1988] is an inductive method which is an inverted version of resolution rule. There are two types of inductive rules, one is called 'V' operator, the other is 'W' operator. The 'V' operator are defined as follows.

$$D = (C \backslash \{l\} \cup C' \backslash \{l'\})\theta\theta'$$

$$l\theta = \neg l'\theta'$$

where C, C' and D are clauses, l(resp. l') is a positive (negative) literal. Generating C' from C and D is called absorption, and generating C from C' and D is called identification. Most specific solution for absorption and identification can be uniquely identified. The 'W' operator can be viewed as combination of two 'V' operators. Assume C_1 and C_2 resolve on common literal L within A, and each resolvant of C_1(resp. C_2) and A is B_1(resp. B_2). The 'W' operator generates C_1, C_2 and A for given B_1 and B_2. If L is negative(resp. positive) literal, the 'W' operator is called intra-construction(resp. inter-construction).

Inverse resolution can be thought to contain the process of abduction. The identification operator can be viewed as a generalized operation of abductive inference. Therefore, ALP will be a special case of inverse resolution. In fact, ALP is the ILP setting that the hypothesis language is a set of ground atom(called abducibles), an example is the one positive atom(it is called an observation or a goal) and biases are represented as integrity constraints. Therefore, it is worthy to compare inverse resolution and integrated frameworks of abductive and inductive inferences to identify the relationship between abduction and induction.

The difference between inverse resolution and our methods are:

- Generating hypotheses

  the 'V' operator generates a clause given a resolvant and a clause which is used to generate the resolvant. Our method can generate both clauses, given resolvant and type and mode declaration. One clause, which can be considered as generalization of a given clause, is systematically enumerated under the restriction of type and mode declaration, and the other clause, which is assumed to be a ground atom, is generated by using abduction. Figure2 illustrates the relationship between inverse resolution and our approach.

- Plausibility of hypotheses

  In our approach, the hypotheses are systematically enumerated. Therefore the hypotheses should be

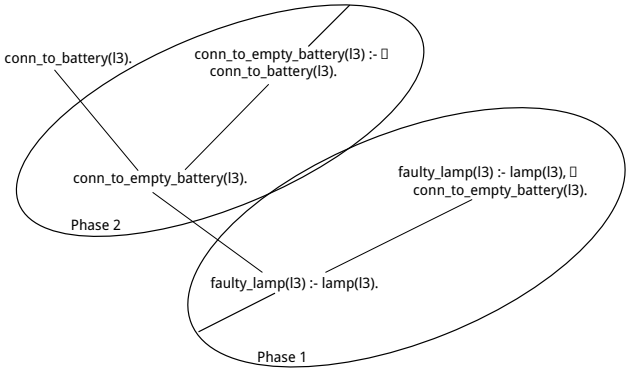

Figure 2: Diagram of Inverse resolution

evaluated in terms of the plausibility. In the evaluation process, our method can select the plausible clause by considering the costs of abductive explanation. In our approach, the costs of explanation can be changed flexibly, because the cost of abducibles are described in integrity constraints.

- Predicate Invention

  The 'W' operator generates a new predicate automatically, but in our problem setting, atoms are specified as abducibles. In fact, abducibles indicate the invented literal, which is automatically discovered by the 'W' operators.

  The ILP systems which use 'W' operators select the plausible hypothesis by heuristics, therefore the form of an argument in new predicates are chosen by given heuristics. In our approach, type and mode declaration is used to restrict the form of an argument in new predicates.

3.3 Comparison to abductive concept learning

In [Dimopoulos and Kakas, 1996], an integrated framework of abduction and machine learning, called abductive concept learning, is introduced. This integration can be summarized as follows.

Input:

- E: an example for the concept to be learned
- O: a set of relevant observations

Procedure:

1. Explain the observations for each example
2. Construct the compact explanation sets
3. Choose a subset of the compact explanations
4. Find the (skeptical) abductive conclusions under the chosen subset of explanation sets
5. Apply an inductive generalization algorithm with background knowledge to the one generated in step (4)

In this approach, abduction and induction is invoked sequentially. Step 1-3 represent abductive process which explain the given observations O. Step 4 and 5 express inductive generalization using the chosen subset of explanation sets and background knowledge. Unlike [Dimopoulos and Kakas, 1996], Our approach uses abduction in the process of inductive generalization to complement the given background knowledge.

This difference is caused by the difference of problem setting, especially the given input. In our problem setting, the relevant observation O is not given as input. Therefore the explanations which explains the observations O, cannot be generated, and the skeptical abductive conclusions cannot be calculated too. In our problem setting, abductive concept learning needs to be modified to cope with the lack of observations.

The problem of finding the relevant observations is pointed out in [Dimopoulos and Kakas, 1996], but no solution of this problem is described. Our approach can be viewed as searching the relevant observations of examples in the process of refining a hypothesis. Our approach specializes a hypothesis by adding the most gainful literal. The literals, which can be added to a hypothesis, are evaluated by the heuristics which can deal with the cost of explanations. This literal evaluation process is similar to the step 1-3 in abductive concept learning. In abductive concept learning, the observations are true statements and are given as input. On the other hand, in our approach, the observations are automatically enumerated in the process of evaluating literals.

4 Relation between abduction and induction

Our approach have some relations to the following issues of abduction and induction:

- Distinguish abduction and induction

  As described earlier, abduction can be viewed as inverse resolution. But in our view, ALP differs from ILP in terms of the policy that what is needed to learn. ILP focuses on generating clauses to explain some given examples, and ALP generates missing facts that explain the given examples. ILP generates some knowledge while ALP generates some facts. Therefore it is natural to divide the two processes which are generalization of examples and finding missing facts.

  This viewpoint is reflected in our approach. The process between generalizing examples and non-generalized method are distinguished. The former uses ILP approach and the latter uses ALP approach to produce examples.

- Induction using abduction

  In our approach, using abductive procedure to hypotheses confirmation process enables to generate plausible hypothesis under the situation that background knowledge is not sufficiently provided. This is done by supplementing the given background

knowledge dynamically. Furthermore, the rules which explains the abduced atoms are generated by iterating the generalization process to the obtained explanations.

5 Conclusion

In this paper, we proposed an integrated framework of abductive logic programming and inductive logic programming.

The proposed approach can deal with hypotheses even if background knowledge is described incompletely. This is achieved by supplementing insufficient background knowledge using abductive reasoning. We also extend the framework to search hypotheses space effectively by considering the cost of explanation. The heuristics is an extended FOIL's heuristics which can cope with insufficient background knowledge when evaluating hypotheses. We have implemented a prototype of the abductive and inductive procedure in Prolog, and confirmed that our system can deal with the above example and several examples. We compared our method to SLDNFAI, inverse resolution and abductive concept learning. We showed that our method can deal with the plausibility of hypotheses by considering the costs of the explanations.

The similarity and difference among our method and other integrated frameworks of abduction and induction should be studied further, because it is expected that each approach has different preferable problem domains. In addition to the further comparison, our method should be applied to more complicated learning task to confirm whether our approach can be used more practical situations.

References

[Ade and Denecker, 1995] Hilde Ade and Marc Denecker. AILP: Abductive inductive logic programming. In *Proceedings of 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1201–1207, 1995.

[Denecker and Danny, 1992] Marc Denecker and De Schreye Danny. SLDNFA: An abductive procedure for normal abductive programs. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 686–700, Washington, USA, 1992. The MIT Press.

[Dimopoulos and Kakas, 1996] Yannis Dimopoulos and Antonis Kakas. *Abduction and Learning. In Advances in ILP. Luc De Raedt(ed.).* IOS Press, 1996.

[Eshghi and A., 1989] K. Eshghi and Kowalski R. A. Abduction compared with negation by failure. In Giorgio Levi and Maurizio Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 234–254, Lisbon, 1989. The MIT Press.

[Kakas *et al.*, 1992] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductiive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[Muggleton and Buntine, 1988] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[Muggleton and de Raedt, 1994] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[Quinlan, 1990] R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.