Inductive Logic Programming Using a MaxSAT Solver

Noriaki Chikara[1], Miyuki Koshimura[2], Hiroshi Fujita[2], and Ryuzo Hasegawa[2]

[1] National Institute of Technology, Tokuyama College, Gakuendai, Shunan, Yamaguchi 745-8585, Japan
tikara@tokuyama.ac.jp
[2] Kyushu University, Motooka 744, Nishi-ku, Fukuoka 819-0395, Japan
{koshi, fujita, hasegawa}@inf.kyushu-u.ac.jp

Abstract. Inductive Logic Programming (ILP) is a method of machine learning which is based on predicate logic. In this work, we propose a new method that transforms a problem of ILP into that of MaxSAT. MaxSAT is an optimization version of SAT. It finds an variable assignment that maximizes the number of satisfied clauses.

1 Introduction

Inductive Logic Programming (ILP) is a method of inductive learning which uses logic programming. It is able to learn directly from complex data. ILP has a wide variety of applications, for example, toxicity prediction of chemical, natural language processing, web mining, and so on.

MaxSAT is an extension of Satisfiability Testing (SAT) so as to satisfy clauses as many as possible. Recently, there has been a lot of progress in SAT/MaxSAT solvers and they have been applied, in many cases with remarkable success, to a number of applications.

In this study, we propose a method which transforms a problem of ILP into that of MaxSAT in order to utilize state-of-the-art MaxSAT solver. We give a syntactical restriction on the ILP problem while mimic the search in Progol and Aleph which are used for many practical applications. In addition, each ILP problem is preprocessed before the transformation in order to prevent the size of the transformed problem growing up. Kondo and Yamamoto[1] proposed a method which solves ILP with SAT solver. Their method is based on a declarative view on ILP while our method is based on an operational view on practical ILP systems.

2 Inductive Logic Programming

This section outlines Inductive Logic Programming (ILP) and a search method used in ILP systems.

2.1 Framework of Inductive Logic Programming

Inductive Logic Programming (ILP)[2] is a method of inductive learning based on predicate logic. ILP uses logic programming as an uniform representation of background knowledge and hypotheses. Given an encoding of the known background knowledge B, a set of positive examples E^+, and a set of negative examples E^- which satisfy the following relations:

$$B \not\models E^+ \tag{1}$$

$$B \cup E^- \not\models \Box \tag{2}$$

An ILP system will derive a hypothesized logic program which satisfies the following relations:

$$H \cup B \models E^+ \tag{3}$$

$$H \cup B \not\models \ E^- \tag{4}$$

The relations from (1) to (4) indicate that (1) E^+ cannot be derived from B only, (2) B and E^- don't conflict, (3) E^+ can be derived from B plus H, and (4) E^- can't be derived from B plus H

2.2 A Search Method of Inductive Logic Programming

Many ILP systems extract multiple hypotheses with a cover set algorithm so as to cover all positive examples. The algorithm of the typical ILP systems such as Aleph[3] and Progol[4] is as follows, where B is the background knowledge, H is hypotheses, and E is a set of positive examples. H is initialized to $\emptyset$.

```
(1) If $E = \emptyset$, then output $H$.
(2) Let $e$ be an example in $E$.
(3) Generate a MSH from $e$ and $B$.
(4) Generate the best hypothesis $H'$ with a top-down search.
(5) $H := H \cup H'$.
(6) $E' := \{e' \mid e' \in E \ and \ B \cup H' \models e'\}$.
(7) Goto (1).
```

In (3), MSH refers the most specific hypotheses that are used to restrict the following top-down search to hypotheses related to MSH. MSH are the most weakest hypotheses to explain the example. Here, the weaker a hypothesis is, the less it explains. Generally, a hypothesis consists of infinite literals. In order to keep only finite literals, those ILP systems use mode declarations.

The top-down search in (4) is performed as follows: First, we start a most general shortest hypothesis. Second, we add literals in MSH to the hypothesis using a refinement operation under the restriction in which the literals are connected by common variables. We generate the "best" hypothesis in (4). We consider the hypothesis, which satisfies positive examples as many as possible, does not satisfy negative examples and has literals in its body as less as possible, is good one.

In this study, we use a MaxSAT solver for the top-down search in (4) and the same refinement operation as that of Aleph and Progol.

2.3 Restricting ILP

In this study, we restrict ILP problems as follows: Arguments in the predicate do not have structure, i.e. we consider only Datalog. All predicates are required to having mode declarations. We also do not deal with negated atoms.

3 MaxSAT Encoding of ILP Problem

Boolean satisfiability testing (SAT) is the first problem of which NP completeness has been proven. It is a basis of many problems in computer engineering and artificail intelligence. MaxSAT[5] is an extension of SAT to solve an optimization problem. A MaxSAT solver tries to find a variable assignment which satisfies clauses as many as possible while a SAT solver finds the assignment which satisfies all the clauses.

In this study, we make use of weighted partial MaxSAT (WPMS) in which a problem is represented as a set of hard clauses and weighted soft clauses. A WPMS solver tries to find a variable assignment which satisfies all the hard clauses and maximizes the sum of weights of satisfied soft clauses. Before we encode the problem with MaxSAT, we perform a preprocess that transforms causality among literals in the MSH into a tree structure.

3.1 Preprocessing

A straightforward MaxSAT-encoding of ILP problems suffers from generating exponential number of propositional variables and clauses with respect to the number of MSH literals. To remedy this, we first construct a tree structure that represent causality between MSH literals in the given problem.

First, let consider the case of generating a hypothesis H' from a hypothesis H by applying a refinement operation with a predicate `p4/3` as follows:

H: `head(A) :- p1(A,B), p2(A,C), p3(A,D)`

$\Downarrow$ *Refinement operation*

H': `head(A) :- p1(A,B), p2(A,C), p3(A,D), p4(B,E,F)`

where the predicates `p1(A,B)` and `p4(B,E,F)` have a shared variable `B`. Such refinement operations are performed during the search of ILP. Unifiability between `p1(A,B)` and a ground unit clause in the background knowledge B depends on which ground unit clause in B is unified with `p4(B,E,F)`. Fig. 1 shows a graph which represents a causality of literals in H'.

We should notice that the graph has a tree structure. When we consider the unifiabiliy between a node literal in such a tree, we have only to take care of its descendant nodes locally. However, the refinement operation may violate tree structures. For example, applying the refinement operation to H' with a predicate `p5(C,D)` makes a structure shown in the left graph of Fig. 2. The structure is not a tree anymore.

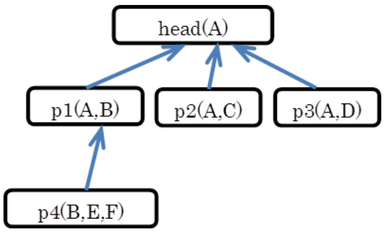

Fig. 1. Causality graph of literals in the hypothesis

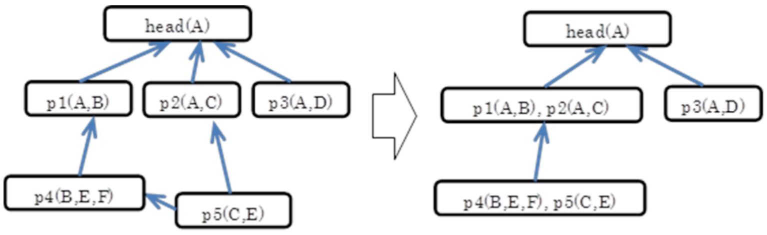

Fig. 2. The transformation to tree structure in the case of causality loop exists

We group literals violating the tree structure. The literals are grouped according to their distance from the head. We treat each group as a node in the graph. The right tree of Fig. 2 is the result of such grouping. Thus, we keep tree structure.

3.2 MaxSAT encoding

After the preprocessing, we obtain the most specific hypothesis MSH and the corresponding tree structure. We make use of MaxSAT to generate the best hypothesis H' from MSH. We consider the following two criteria on a hypothesis: the number of positive examples covered by the hypothesis, and the number of literals in the hypothesis. In this study, we exclude hypothesis that covers negative examples.

In order to take the criteria into account, we introduce two predicates $r(i,j)$ and $p(ex)$:

$r(i,j)$ means that the j-th literal of the i-th literal group in MSH appears in H'.

$p(ex)$ means that H' covers the ex-th positive example.

As we prefer the hypothesis having less literals in its body, we declare a negative unit clause $\neg r(i,j)$ as a soft clause having weight 1. We also prefer the hypothesis covering more positive examples. So, we declare a positive unit clause $p(ex)$ as a soft clause having weight $n+1$ where n is the number of literals in MSH. In this way, we respect the number of covered positive examples than the number of literals in the hypothesis.

We introduce other several predicates for the MaxSAT encoding. We enumerate the primary predicates as follows:

$a(i, bki)$ means that the i-th literal group in MSH is satisfied. Through bki, we can find out ground unit clauses in the background knowledge which are used for the satisfaction.

$e(i_a, bki_a, i_b)$ means that the necessary condition for $a(i_a, bki_a)$ is satisfied in the decendant nodes of the i_b-th literal group in MSH.

$n(ey)$ means that H' covers the ey-th negative example.

We also introduce several hard clauses that represent the relationship among the predicates. In order to exclude the hypothesis that satisfies negative examples, we declare a negative unit clause $\neg n(ey)$ as a hard clause.

4 Experiment

We used Connect-4, Audiology and Molecular Biology (splice-junction gene sequences) data sets of UCI[6] for the experiment. Connect-4 is a two-player connection game. Each data of Connect-4 has 42 attributes. We took the first 50 examples from 67,557 examples in the data set. They consist of 38 win examples, 5 draw examples, and 7 loss examples. We have succeeded in extracting seven rules from 38 win examples, one rule from 5 draw examples, and one rule from 7 loss examples.

Each data of Audiology has 69 attributes. We randomly selected 100 examples from 226 examples in the data set.The data set contain 20 cochlear age examples, 10 cochlear age and noise examples, 22 cochlear unknown examples, 2 mixed cochlear age otitis media examples, and 5 possible menieres examples.

Molecular Biology data set is a database of primate splice-junction gene sequences (DNA) with associated imperfect domain theory. Each data of Molecular Biology has 60 attributes. We randomly selected 200 examples from 3,190 examples in the data set. The data set contain 47 EI examples and 105 Neither examples.

The experiment were carried out on Windows 7 (64bit). The processor is 2.6GHz Intel i7-2620M with 8 GB RAM. We use QMaxSAT[7] 14.04 (on cygwin 1.7.28) as a MaxSAT solver. We also used Java 1.8.0_40 for pre-processing and post-processing and JavaCC as a parser.

Table 1 shows the experimental result. We used Aleph with YAProlog[8] for comparison. With the default setting, Aleph sometimes fail to extract rules because of the limit number of search nodes. In order to extract the same rule as our method does, we adjusted the setting. Our method runs much faster than Aleph for the draw examples. Our method seems to have advantage when the length of the extracted rule is relatively long. On the other hand, when we repeatedly extract short rules, the time for I/O of MaxSAT instances becomes dominant factor for the whole processing time.

Table 1. The execution time of the experiment

Problem	Target of positive examples	Num. of extracted rules	Max. length of extracted rules	Our method (using MaxSAT)	Aleph+ YAProlog
Connect-4	win	7	5 literals	1.501s	9.360s
	draw	1	7 literals	0.877s	56.566s
	loss	1	4 literals	0.970s	0.640s
Audiology (Standardized)	cochlear_age	2	6 literals	2.624s	1m39.169s
	cochlear_age_and_noise	1	4 literals	2.530s	0.218s
	cochlear_unknown	4	6 literals	3.934s	10m47.807s
	mixed_cochlear_age_otitis_media	1	3 literals	1.672s	0.187s
	possible_menieres	1	2 literals	1.438s	0.172s
Molecular Biology (Splice-junction Gene Sequences)	EI	4	5 literals	4.418s	0.328s
	Neither	14	4 literals	1m4.166s	0.905s

5 Conclusion

We proposed a new method that transforms a problem of ILP into that of MaxSAT. We also provided a preprocess that converts an ILP problem to a tree structure, thereby suppressing the size of MaxSAT encoding for it. Experimental results show that our method works faily well on a state-of-the-art MaxSAT solver with reasonable additional cost for problem conversion. Future works are to apply our method to other ILP problems and to evaluate the performance, and to speed up I/O of MaxSAT instances.

Acknowledgments. This work was supported by KAKENHI(25330085).

References

1. Kondo, S., Yamamoto, A.: Inductive Logic Programming Using a SAT Solver, The Special Interest Group Notes of the Japanese Society for Artificial Intelligence, SIG-FPAI-B104, JSAI, pp.75–80 (2012) (in Japanese)
2. Furukawa, K., Ozaki, T., Ueno, K.: Inductive logic programming, Kyoritsu publishing, Tokyo (2001) (in Japanese)
3. Srinivasan, A.: The Aleph Manual, `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/` (1999)
4. Muggleton, S.: Inverse Entailment and Progol: New Generation Computing Journal, Vol. 13, pp. 245–286, (1995)
5. Chu, M., Felip, M.: MaxSAT, Handbook of Satisfiability, pp.613–631 (2009)
6. UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml/`
7. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: A Partial Max-SAT Solver, JSAT, Vol.8 pp. 95–100 (2012)
8. Yet Another Prolog, `http://www.dcc.fc.up.pt/~vsc/Yap/`