Inductive Constraint Logic Programming: An Overview

Srinivas Padmanabhuni
Department of Computing Science
University of Alberta
Edmonton, Canada, T6G 2H1
srinivas@cs.ualberta.ca

Aditya K. Ghose
Decision Systems Lab
Dept. of Business Systems
University of Wollongong
Wollongong, NSW 2522 Australia
aditya@uow.edu.au

Abstract. This paper provides a brief introduction and overview of the emerging area of Inductive Constraint Logic Programming (ICLP). It discusses some of the existing work in the area and presents some of the research issues and open questions that need to be addressed.

1 Introduction

Inductive Logic Programming (ILP) refers to a class of machine learning algorithms where the agent learns a first-order theory from examples and background knowledge. The ILP framework in machine learning is perhaps the most general of all because of the complexity of the concepts learned. The use of first-order logic programs as the underlying representation makes ILP systems more powerful and useful than the conventional empirical machine learning systems. ILP systems have been successfully used in a variety of real life domains including mesh design, protein synthesis, games and fault diagnosis. Most of the first-order representations used in ILP systems are variants of horn-clause based clausal logic of Prolog.

ILP systems have been weak in handling numerical knowledge. Although some existing ILP systems are capable of handling numerical knowledge, the approach in most cases is ad-hoc. Given the obvious utility of extending the power of the logic programming framework to computational domains other than than Herbrand terms (such as sets, strings, integers, reals etc.) and the well-known success of various constraint logic programming languages such as languages like CLP(R), CHIP etc. in various real life applications [1], there is a clear need for developing an inductive framework similar to that of ILP but based on constraint logic programming schemes.

2 Existing work in ICLP

Given that this is a relatively new area, there only a small number of frameworks that attempt a solution to the ICLP problem.

2.1 Kawamura and Furukawa

Furukawa and Kawamura [2] adopted the dominant paradigm in ILP, namely the paradigm of *inverse resolution* for generalizing constraints. As is well known in ILP literature, inverse resolution methods are essentially based on the inversion of the process involved in deduction using resolution. In resolution, given a pair of clauses, called resolvents, a third clause is deduced. In contrast, the process of inverse resolution seeks to invent one of the resolvents, given one of the resolvents and the solved clause. The process basically revolves around three operations, truncation, absorption and intra-construction of two logic program rules.

Truncation Let P1 and P2 be two clauses whose bodies are empty. Then the truncation of P1 and P2 yields a clause which is more general than both P1 and P2.

Absorption Suppose two clauses R1 and R2 on resolution yield the clause R3. Absorption refers to the operation of guessing R2 given R3 and R1.

Intra-Construction Given two clauses C1 and C2, intra-construction refers to the process of generating three clauses R1, R2, and R3 such that R1 and R2 on resolution yield C1 and R2, R3 yield C2 respectively.

In this sense the process of inverse resolution inverts the deduction involved in resolution process.

The ILP frameworks are based on algorithms which are variants of the above-mentioned basic inverse resolution algorithm. The model of θ-subsumption used in ILP cannot be extended to ICLP because of the universal generalization rule adopted in ILP systems, namely, of replacing constants by variables, to yield a generalized formula. But this method is bound to fail with constraint logic programs because of the constraints involved. For instance, the generalization of two equations $\{x = 7, x = 2\}$ is $x = y$ by θ-subsumption. This type of generalization is meaningless, and necessitates the development of domain specific mechanisms to generalize constraint logic programs based on the domain of the constraint s involved.

Kawamura and Furukawa generalize the concepts of least general generalization for general logic programs to constraint logic program generalization, by considering the logic program components and generalization of constraints components separately and merging them. This raises the question of whether there always exists a least general generalization for a given set of constraints. Even if on exists there should be a method to compute the greatest lower bound of all possible constraint generalizations.

In their framework, Kawamura and Furukawa devise methods to compute the inverse resolution based generalizations for linear algebra based constraints.

Consider the development of an absorption algorithm for linear algebra. If c and c2 are two linear algebra constraints, and we need to devise a c1 such that $c1 \cup c2 \vdash c$. Given $c = \{x = 1\}$, and $c1 = \{y = 2, z \geq 2\}$, we get the equation of c_2 as $\{a_1x + b_1y = a_1 + 2b_1, a_2x + b_2x \geq a_2 + 2b_2\}$. This satisfies $c1 \cup c2 \vdash c$.

2.2 Mizoguchi and Ohwada

Mizoguchi and Ohwada [5][6], extend ideas from ILP based on Plotkin's framework [8] of Relative Least General Generalization(RLGG) to induce constraint logic programs. They consider the spatial layout problem, and devise methods to automatically acquire geometric constraints. The objective in RLGG is to construct a minimum clause that covers a set of positive examples yet maintaining consistency with the negative examples in the presence of background knowledge. RLGG is essentially the minimal clause which in conjunction with background knowledge yields the above satisfiability criterion (with respect to positive and negative examples).

To construct an RLGG, the concepts of constraint subsumption, and constrained LGG need to be explained in detail. A constraint C_1 is said to $C - subsume$ C_2 if for the two sets of constraints C_1 and C_2, there exists a constraint C such that the solution set of $C_1 \cup C$ is same as the solution set of C_2. Let $C_1 = \{p(X) \leftarrow \{X + 3 \leq Y\}, p(Y)\}$ and $C_2 = \{p(X) \leftarrow X + 4 \leq Y, p(Y)\}$. Here C_1 C-subsumes C_2.

Consider two constrained clauses CC_1 and CC_2. The notion of C-subsumption is extended to constrained clauses on the following lines. Let the two constrained clauses CC_1 and CC_2 be:

$CC_1 = B_0 \leftarrow C_1, B_1, \ldots, B_n$
and
$CC_2 = A_0 \leftarrow C_2, A_1, \ldots, A_m$

where C_1 and C_2 are conjunctions of constraints and each A_i and B_i is a non-constraint atom. After renaming variables such that A_0 is exactly identical to B_0, and $\{B_1, \ldots, B_n\} \subseteq \{A_1, \ldots, A_n\}$, CC_1 C-subsumes CC_2 if and only if C_1 C-subsumes C_2.

Based on this notion of $C - subsumption$, the *least general constraint set* D of a set of constraints $C_i (i = 1, n)$, is defined such that D C-subsumes all constraints $C_1, \ldots, C_n$. Moreover any constraint set R which C-subsumes all members $C_1, \ldots, C_n$, C-subsumes D too because it is the least general constraint set subsuming the set of constraints $C_1 \ldots C_n$.

A generalization of the concept of the least general generalization of a set of constraints, is the concept of *relative least general generalization* of a set of constraint logic programs in presence of background knowledge. Formally the RLGG for constrained clauses can be defined as the least general constrained clause which in combination with the background knowledge subsumes the set of all input constrained clauses.

Mizoguchi et al. [6] apply the framework of constrained RLGG to the problem of floor planning. The non-overlap conditions of any two rooms in a floor layout

is encoded as a set of constrained clauses. Ultimately the solution desired here is the set of position and size values of the rooms.

In practice there is no finite or tractable RLGG as it contains an intractably large number of atoms and constraints. But suitable practical language restrictions have been shown to yield a computationally feasible RLGG algorithm which can be the basis of learning clauses in ICLP. These restrictions are collectively referred to as *bias*.

A variety of types of bias have been proposed by the authors to regulate the size of RLGG. Some of the common types of biases are based on semantic notions such as *mode* and *type* of constraint. Other types are based on functional dependencies between variables of a constrained clause.

2.3 Page and Frisch

Page and Frisch [7] extend the concepts involved in the generalization of atoms, to more general forms of atoms, especially atoms with constraints attached to them. The constraint is a logical expression built of specialized predicates called *constraint predicates* and represents a form of restriction on the atom to which the constraint is involved. Some examples are:

$eats(x, y)/COW(x) \wedge GRASS(y) \wedge GREEN(y)$,
$eats(x, onland(y))/COW(x) \wedge GREEN(y) \wedge TASTY(onland(y))$

The restriction imposed on the generalization of these atoms in Frisch's paper was that the term, present in the constraint part had to be present in the atom part of the constrained atom.

In their work, Frisch and Page identified generalization in terms of background knowledge. The background theory is termed as *constraint theory* and it is a first-order theory whose only predicates are constraint predicates. In addition to the constrained generalization, they also describe sorted generalization, as a special case where the following restrictions apply in addition to the normal constraint restrictions: Only monadic constraint predicates may be used and only variables are constrained and a given variable may occur in only one atomic formula of the constraint. Thus *sorted generalization* is a specialized case of *constrained generalizations*.

For generalization of constrained atoms, the three factors taken into account are the background theory, the language of atoms and the constraints on the atoms. In general a constrained atom e_1 is considered to be more general than another constrained atom e_2 iff there is a substitution θ such that θ maps the head of e_1 to the head of e_2 and θ respects the constraints.

Based on this notion of generalization, the least general constrained atom is defined as the constrained atom that is more general than a given set of constrained atoms and is less general than any other constrained atom which is more general than any member of the given set of constraints. However in many cases, it is not possible to define least generalization exactly. In such a case a set of minimal non-comparable generalization of the given set of constrained atoms is found. Such a set of generalizations is called a *complete set of incomparable generalizations* (CIG).

It is shown that *a set of constrained atoms built from the same ordinary predicate has a singleton CIG.* This conjecture is true because constrained atoms allow for use of generalized terms rather than just sorts in the constrained atoms.

Consider the following example to illustrate computation of the CIG of a set of constrained atoms.

Example 1. Consider the following set of constraints forming the background theory:

$\{LARGER(succ(brown), succ(tom)), LARGER(succ(mary), succ(lisa)),$
$LARGER(brown, tom), LARGER(mary, lisa), MALE(brown),$
$MALE(tom), FEMALE(lisa), FEMALE(mary),$
$\forall x \forall y MALE(x) \wedge FEMALE(y) \rightarrow LARGER(x, y)\}.$

With reference to the above background knowledge, if we were to compute the CIG of the set of constrained atoms $\{BULLIES(tom, x)/FEMALE(x),$ $BULLIES(y, mary)/MALE(y)\}$. the answer would be:
$\{BULLIES(x, y)/LARGER(x, y) \wedge MALE(x) \wedge FEMALE(y)\}$.

With reference to the same background knowledge, if we we were to compute the CIG of the set of constrained atoms $\{BULLIES(succ(brown), succ(tom)),$ $BULLIES(succ(mary), succ(lisa))\}$, then the answer would be
$\{BULLIES(succ(x), succ(y))/LARGER(x, y) \wedge LARGER(succ(x), succ(y))\}$.

It is thus clear that background knowledge had a role in deriving the CIG of the input constrained atoms.

2.4 Sebag et al.

Sebag et al. (see [9] [10] and their chapter in this volume) propose a framework for learning clauses which can discriminate between positive and negative examples expressed as constrained clauses. They conjecture that only a subset of the entire set of discriminating clauses need to be determined fully in order to explicitly represent the learned concept. The remaining clauses of the concept can be derived from these basic set of clauses. Thus they use a two step induction process in learning constrained clauses. The first step called small induction gives a computational characterization of the sufficient discriminating clauses. In the second step of exhaustive induction, the entire set of discriminating clauses is constructed.

2.5 Martin and Vrain

In their study of induction of constraint logic programs, Martin and Vrain [4] conjecture that instead of interpreting function symbols in constraints symbolically, if we interpret them by more semantic means, there is scope for development of better algorithms for generalizing and inducing constraint logic programs. They propose a method to learn logic programs containing function symbols other than mere constants. They are able to use the interpretation of functions not as

mere symbols but as semantic entities based on consideration of their domains. They rely not on a syntactic notion of function terms as is usually the case in ILP, but consider the semantics of functions in terms of the domain values and interpretations. They use the domain values of the functions, and develop a model for learning constraint logic programs based on the domain values of a function in the program and its role in discriminating between positive and negative examples. In their work they do not concentrate on generating complete and consistent logic programs from the given positive and negative examples. Instead they develop a notion of coverage of the input positive and negative examples based on a concept called *D-coverage*[4]. The notion of *D-coverage* is not shown to be sufficient for learning complete and consistent logic programs but has been shown to be successfully employable for useful classes of constraint logic programs.

D-coverage: A *D-atom* refers to a predicate of form $p(d_1, d_2, .., d_n)$ where p belongs to the set of allowed predicates in the CLP signature. and each of the d_i's belong to the domain of CLP. A D-atom e is covered by a constrained clause, with respect to the background knowledge (BK^+) and the set of positive examples (E^+) if there is a valuation which makes the head of the clause to be e and makes the body of the constrained clause to be true including the constraint part of the clause. Thus, the notion of D-coverage is based on a semantic notion of functions and is not a symbolic notion.

Example 2. Consider the example specification, where $D = N$, the set of positive integers. Let *pred* be the function involved, and let it be interpreted as predecessor function of natural numbers. The basic predicate *even* is defined by $BK^+ = \{even(0), even(2), even(4)\}$ and the target predicate *odd* is given by $E^+ = \{odd(1), odd(3), odd(5)\}$, and $E^- = \{odd(2), odd(4)\}$. In this example a possible solution is then:

$\{odd(X) \leftarrow X = 1.odd(X) \leftarrow pred(pred(X)), odd(Y).\}$

The idea is to build constrained clauses by adding iteratively to the body of the clause either a constrained atom or a constraint, until no negative example is *D-covered*.

The different steps involved in building *D*-linked constrained clauses are as follows:

1. Choose a random example e belonging to the set of target examples belonging to E^+.
2. Build a relevant clause which *D*-covers e.
3. Build a set of possible constraints or constrained atoms which can be added to the body of the clause.
4. Use an entropy measure to choose the best constraint which is added to the body of the clause so that as many positive *D*-uncovered examples are covered.

Because of the complexity of the problem of finding constrained clause due to the infinite number of the possible terms, the generalization needs to be

controlled by use of a suitable form of *bias*. Different restrictions on the structure of the possible generalizations are imposed. Some of the biases used here are:

1. Limitation on the depth of terms
2. Limitation on the number of constraints
3. Limiting use of other functions inside a top level function. e.g. pred is not allowed inside the succ function, otherwise it can lead to an infinite computation.

This work in the direction of generalization of constraint logic programs illustrates some important factors which need to be taken into consideration when generalizing constraints and constraint logic programs.

3 Research issues and open questions

It is clear that the ICLP area is well-motivated, with possible applications including robot motion planning,spatial reasoning, temporal reasoning, graphical layout problems and computer aided publishing, amongst others. It is also clear that several outstanding research issues remain. We shall consider, now, some of the basic questions that need to be addressed by ICLP researchers.

1. *Constraint induction:* Much needs to be done in defining techniques for inducing constraints in the well-known constraint domains. Of interest as well is the issue of integrating domain-specific constraint induction methods to obtain techniques for inducing constraint logic programs which operate on multiple domains. One interesting approach to this problem is the work of Padmanabhuni, You and Ghose that appears in this volume.
2. *Adaptation of ILP techniques to ICLP:* It is important to consider which of the existing ILP techniques may be adapted for ICLP. In the previous section we saw the ICLP version of inverse resolution and RLGG frameworks. It would be interesting to observe if inverse unification, inverse entailment, generalized subsumption and other models of generalization which have been used in ILP can be extended to ICLP. The work of Martin and Vrain [4] mentioned in the previous section stresses the need to interpret functions and predicates in a more semantic sense than is done in the usual ILP case. In our opinion the biggest stumbling block in the direction of adaptation of ILP techniques to ICLP based systems is the syntactic interpretation of the functions and predicates involved in the CLP. The most commonly used technique in the process of induction in ILP, the replacing of a constant by a variable, is inapplicable to generalize in CLP's because it is based on a syntactic interpretation of functions and predicates.
3. *Semantics:* Even though the study of ICLP is driven more by applications, a comprehensive and accessible semantic basis is still needed to foster growth of the field.

4. *Bias:* A major area of research motivated by all the ICLP systems discussed in the previous section involves the question of developing appropriate notions of bias and language restrictions as a means of managing the time and space complexity inherent in the problem.
5. *Novel applications:* New application domains where automatic acquisition of constraints may be helpful, such as computer aided publishing [3], need to be studied.
6. *Learnability studies:* The other aspect of the interest involves studies on learnability issues from computational learning theory.

References

1. J. Jaffar and M.J.Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, pages 503–581, 1994.
2. T. Kawamura and K. Furukawa. Towards inductive generalization in constraint logic programs. In *Proceedings of the IJCAI-93 workshop on inductive logic programming*, pages 93–104, Chamberry, France, August 1993. Academic Press.
3. F. Jacquenet M. Bernard and C. Nicolini. Induction of constraint logic programs for computer-aided publishing. In *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*, Banff, Canada, July 1997.
4. Lionel Martin and Christel Vrain. Induction of constraint logic programs. In *Proceedings of Algorithms and Learning Theory (ALT) - 1996*, Sydney, Australia, October 1996. Springer Verlag.
5. F. Mizoguchi and H. Ohwada. Constraint-directed generalization for learning spatial relations. In *Proc. Second international workshop on ILP*, Tokyo, Japan, 1992.
6. F. Mizoguchi and H. Ohwada. Constrained relative least general generalization for inducing constraint logic programs". *New Generation Computing*, 13:335–368, 1995.
7. C.D. Page and A.M. Frisch. Generalizing atoms in constraint logic. In *Proc. Second international conference on knowledge representaion and reasoning*, pages 429–440, 1991.
8. G.D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:101–124, 1971.
9. M. Sebag and C. Rouveirol. *Constraint inductive logic programming*, pages 277–294. Advances in ILP. IOS Press, 1996.
10. Michele Sebag and Celine Rouveirol. Induction of maximally general clauses compatible with integrity constraints. In Stefan Wrobel, editor, *Proc. of Fourth International workshop on Inductive Logic Programming*, 1994.

This article was processed using the LaTeX macro package with LLNCS style