A Hypergraph Representation for Deductive Reasoning Systems *

Timothy E. Denehy and Jennifer Seitzer†
University of Dayton
Computer Science Department
denehyte@flyernet.udayton.edu
seitzer@cps.udayton.edu

Abstract

Hypergraphs are often used to represent knowledge bases because of their accurate depiction of causal links between facts. Deduction can be computed over such a knowledge base by complete hypergraph traversal. Efficient structuring of such hypergraphs can reduce time in deductive computation as well as space in vertex and hyperedge storage.

In this work, we transform a hypergraph storing a knowledge base into a more efficient representation. By representing only part of the ground instantiation of the internal knowledge base, we attain a great space reduction. This partial representation is accomplished by storing non-ground, intensional predicates and by performing individual instantiations on these. In this paper, we present the implementation of these techniques and compare them to those used in computer system **INDED** (pronounced "indeed").

Keywords: hypergraph, instantiation, logic programming.

Introduction

A logic program is a set of logic formulas grouped together to computationally solve a problem or perform a task. Logic programs serve as mechanisms of knowledge representation which are useful in expert system, deductive database, and truth maintenance system design. They are powerful vehicles of data augmentation that enable artificial intelligence computer systems to increase information using the techniques of logical deduction.

A semantics of a propositional logic program can be thought of as a function which produces the set of facts logically implied by the program, and therefore, states which propositions mentioned in the program are considered **true** and which are considered **false**. The stable semantics [GL90] and its three-valued approximation, the well-founded semantics [VRS91], are able to interpret logic programs with rules containing negative literals and (positive and negative) cycles of dependencies formed amongst its literals [1].

To depict causal dependencies among facts in a knowledge base, a hypergraph is often used [K.B95]. A hypergraph is a set of vertices and hyperedges, where a hyperedge is an ordered pair of subsets of vertices. In this representation, each fact is represented by a vertex; each rule body is represented by a hyperedge [Ull88].

In the original implementation of **INDED**, the hypergraph grew exponentially with the size of the knowledge base [Sei99]. This made the problem space quite limited. Only small problems could be solved using **INDED** due to memory limitations. In this work, we redesign the internal representation of the hypergraph and obtain an equivalent representation that uses much less memory.

Definitions and Terminology

A logic program is a set of logic formulas. Logic programs are powerful mechanisms of knowledge representation.

Definition 2.1 (atom) *An* atom *a is a construct of the form $\alpha(B)$ where α is a predicate symbol and B is an argument list $b_0, b_1, ..., b_n$ of either variables or constants.*

Definition 2.2 (rule) *A* rule *r is an expression of the form $\alpha_0(B_0) \leftarrow \alpha_1(B_1), \alpha_2(B_2), ..., \alpha_n(B_n)$. The first atom is the* head *of the rule, and the remaining atoms constitute the rule's* body. *Each atom may be preceded by a $\sim$ representing logical negation. The truth value of the head atom is determined by conjuncting the truth values of the rule's body atoms. An* intensional rule *is one in which all argument lists are made up of variables. A* ground instance *of an intensional rule is that rule with no variables (only constants).*

*Copyright @ 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

†This work is partially supported by grant 9806184 of the National Science Foundation.

[1] Although the formal definitions of these semantics are cited above, for this paper, we can intuitively accept stable and well-founded models as those sets of propositions that are generated by transitively applying modus ponens to rules.

Definition 2.3 (logic program) *A* logic program *P is a set of intensional rules. The* ground instantiation *of a logic program P_C is constructed by forming all possible ground instances of the rules of P with a set of constants C.*

Original Hypergraph Representation

A large part of this work was implemented by studying some of the shortcomings of the hypergraph representation of system **INDED**. As mentioned above, the complete structure was generated and traversed for input files that may only have required partial generation and traversal. Before discussing our reduction techniques, we first describe this knowledge based system.

System **INDED** is a knowledge discovery system that uses inductive logic programming (ILP) [LD94] as its discovery technique. To maintain a database of background knowledge, **INDED** houses a deduction engine that uses deductive logic programming to compute the current state (current set of true facts) as new rules and facts are procured.

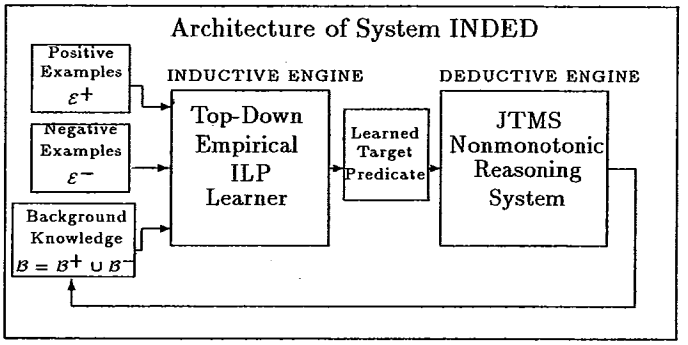
Inductive Logic Programming

Inductive logic programming (ILP) is a research area in artificial intelligence that attempts to attain some of the goals of machine learning while using the techniques, language, and methodologies of logic programming. Some of the areas to which ILP has been applied are data mining, knowledge acquisition, and scientific discovery [LD94]. The goal of an inductive logic programming system is to output a rule which *covers* (entails) an entire set of positive observations, or examples, and *excludes* or *does not cover* a set of negative examples [Mug92]. This rule is constructed using a set of known facts and rules, knowledge, called domain or *background* knowledge. In essence, the ILP objective is to synthesize a logic program, or at least part of a logic program using examples, background knowledge, and an entailment relation.

Architecture of INDED

System **INDED** is comprised of two main computation engines. The deduction engine is a bottom-up reasoning system that computes the current state by generating a stable model [2], if there is one, of the current ground instantiation represented internally as a hypergraph, and by generating the well-founded model [VRS91], if there is no stable model[GL90]. This deduction engine is, in essence, a justification truth maintenance system [Doy79] which accommodates non-monotonic updates in the forms of positive or negative facts.

The induction engine, using the current state created by the deduction engine as the background knowledge base, along with positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, induces a rule(s) which is then used to augment the deductive engine's hypergraph. We use a standard top-down hypothesis construction algorithm (learning algorithm) in **INDED**[LD94]. This algorithm uses two nested programming loops. The outer (covering) loop attempts to cover all positive examples, while the inner loop (specialization) attempts to exclude all negative examples. Termination is dictated by two user-input values to indicate sufficiency and necessity stopping criterea. The following diagram illustrates the discovery constituents of **INDED** and their symbiotic interaction.



The input files to **INDED** that initialize the system are the extensional database (EDB) and the intensional database (IDB). The EDB is made up of initial ground facts (facts with no variables, only constants). The IDB is made up of universally quantified rules with no constants, only variables. Together, these form the ground instantiation represented internally as the deduction engine hypergraph. We formally define *hypergraph* as follows.

Definition 3.4 (hypergraph representation) *Let P_C be a ground instantiation of a logic program with constant set C, and let A be the set of unique atoms within P_C. The* hypergraph representation *G of P_C has vertex set $G_V = A$ and edge set $G_E = \{ (a_0, S) \mid \exists\, r \in P_C$ with head atom a_0 and a set of body atoms $S \}$. Each edge part also contains the negation status of its respective body atom.*

Reduced Hypergraph Representation

The size of the original hypergraph representation grows exponentially with the number of constants. Because this representation is based on a set of ground instances of intensional rules, it can be seen that the hypergraph produced consists of several subgraphs with similar structure. These subgraphs contain vertices with identical predicate expressions and similar edge configurations. They differ, however, in the constant arguments making up each atom. This observation motivates the construction of a reduced hypergraph representation based solely on the intensional rules of a logic program (not on their ground instances). Such a reduction can be accomplished by simplifying the vertices and adding information to the edges of the hypergraph.

[2] Although the formal definitions of these semantics are cited above, for this paper, we can intuitively accept stable and well-founded models as those sets of facts that are generated by transitively applying modus ponens to rules.

The vertices of the new hypergraph will contain predicate expressions that can be instantiated with a list of constants to form a conventional atom.

Definition 4.5 (instantiation) *An* instantiation *I is a list $i_0, i_1, ..., i_n$ of either constants or variables.*

Traversing the edges in the hypergraph representation produces a set containing a rule's body atoms. In order to obtain the correct instantiation of each predicate expression in the body, we must define a translation from the head atom's constant arguments to those found in the body atom.

Definition 4.6 (translation) *A* translation *T is a list $t_0, t_1, ..., t_n$ of integers representing instantiation positions. The application of T to instantiation I yields a new instantiation $T(I) = J$ with $j_0 = i_{t_0}, j_1 = i_{t_1}, ..., j_n = i_{t_n}$.*

The lists of arguments in a rule's body atoms may contain constants that do not appear in the head atom's arguments. Therefore, we must define an augmented instantiation that includes these additional constants. This new instantiation will be applied to the head predicate expressions in the hypergraph to form a head atom, and it will serve as the basis for the translations to the body atoms.

Definition 4.7 (augmented instantiation) *For a rule r of the form $\alpha_0(B_0) \leftarrow \alpha_1(B_1), \alpha_2(B_2), ..., \alpha_n(B_n)$, let B'_0 be the list of unique variables appearing in $B_1, B_2, ..., B_n$ but not in B_0. The* augmented instantiation *B_A is defined such that $b_{A0} = b_{00}$, $b_{A1} = b_{01}$, ..., $b_{Ai} = b_{0i}$, $b_{A(i+1)} = b'_{00}$, ..., $b_{A(i+j+2)} = b'_{0j}$.*

Since each translation corresponds to a specific edge part, we form associations between rules, predicate expressions in the rule's body, and the translation used to traverse from the head of the rule to that predicate expression.

Definition 4.8 (association) *For an intensional rule r with head atom $\alpha_0(B_0)$, a body atom $\alpha_i(B_i)$, and the augmented instantiation B_A, an* association *is a 3-tuple (r, α_i, T) such that $T(B_A) = B_i$.*

Finally, we can describe the reduced hypergraph representation of a logic program in terms of our previous definitions.

Definition 4.9 (reduced hypergraph) *Let P be a logic program with a set of unique predicate expressions A. The* reduced hypergraph representation *H of P has vertex set $H_V = A$ and edge set $H_E = \{ (\alpha_0, S, U) \mid \exists\, r \in P$ with head predicate expression α_0, a set of body predicate expressions S, and the set U of all associations $(r, \alpha_i, T_i) \}$. Each edge part also contains the negation status of its respective body predicate expression.*

The important aspect of this work is the realization that the reduced hypergraph representation maintains the same functionality as the original hypergraph representation.

Theorem 4.10 *Let P be a logic program and C a set of constants. The reduced hypergraph representation H of P along with a set of instantiations of constants from C is equivalent to the original hypergraph representation G of P_C.*

Proof: *For each rule r of the form $\alpha_0(B_0) \leftarrow \alpha_1(B_1), \alpha_2(B_2), ..., \alpha_n(B_n)$ in P_C, there exist atom vertices $a_0, a_1, ..., a_n$ in G_V and predicate expression vertices α_0, α_1, ..., α_n in H_V. G_E contains an edge $(a_0, \{a_1, ..., a_n\})$, and H_E contains an edge $(\alpha_0, \{\alpha_1, ..., \alpha_n\}, U)$. Let B_A be the augmented instantiation derived from B_0, B_1, ..., B_n. Applying instantiation B_A to α_0 yields a vertex equivalent to a_0. Utilizing the transitions T from the associations in U yields the set of atoms $\{\alpha_1(T_1(B_A)) = a_1, ..., \alpha_n(T_n(B_A)) = a_n\}$. Thus, the application of an instantiation from C yields a vertex and an edge in H equivalent to those found in G.*

Example

Consider the logic program P

$$propersubset(X, Y) \leftarrow subset(X, Y), \sim subset(Y, X).$$

This program determines if X is a proper subset of Y based on whether or not X is a subset of Y and Y is a subset of X. Consider the relation between the set of vehicles and the set of cars. We will describe these sets with the set of constants $C = \{v, c\}$. Figure 1 shows the hypergraph representation P_C. This hypergraph demonstrates every possible proper subset relation between vehicles and cars, and traversing the edges leads to the basis facts upon which each relation may be true. It can be seen that the subgraph containing *propersubset* (v, v) is very similar to the subgraph containing *propersubset* (c, c). This is also true for the subgraphs containing *propersubset* (v, c) and *propersubset* (c, v).

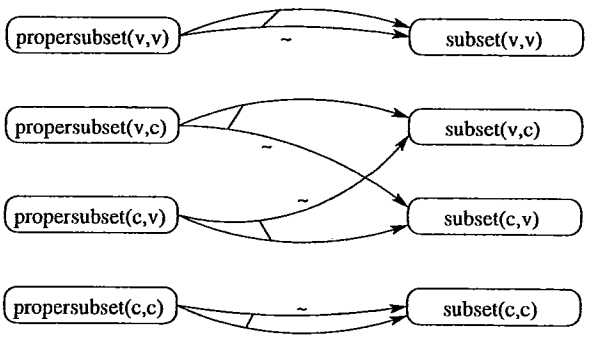


Figure 1: Original Hypergraph Representation of P_C

The reduced hypergraph representation of P and the set of possible constant instantiations from C is shown in Figure 2. Applying each of the instantiations to the *propersubset* vertex produces a

subgraph of P_C. For example, applying the instantiation $[v, c]$ to the *propersubset* vertex produces atom $propersubset(v, c)$. Traversing the positive edge part and applying the translation $T(0, 1)$ produces $subset(v, c)$. Traversing the negative edge part and applying the translation $T(1, 0)$ produces $subset(c, v)$.

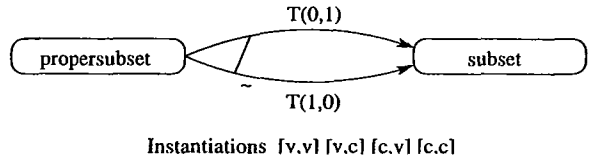


Figure 2: Reduced Hypergraph Representation of P

Algorithm for Reduced Hypergraph Construction

The method for constructing the reduced hypergraph representation of a logic program is given in Algorithm 6.11.

Algorithm 6.11 *Reduced Hypergraph Construction*

```
for each rule r of the form
α0(B0) ← α1(B1), α2(B2), ..., αn(Bn) in P {
  set the augmented instantiation size for r to 0
  if α0 is not in the predicate expression set {
    add α0 to the predicate expression set
    create a new vertex v for α0
  }
  else {
    retrieve vertex v for α0
  }
  create a new edge e emanating from v
  for each variable bi in B0 {
    if bi is not in the variable list of r {
      add bi to the variable list of r
      increment augmented instantiation size for r
    }
  }
  for each ai in the body of r {
    if αi is not in the predicate expression set {
      add αi to the predicate expression set
      create a new vertex vi for αi
    }
    else {
      retrieve vertex vi for αi
    }
    create a new translation T
    for each variable bj in Bi {
      if bj is not in the variable list of r {
        add bj to the variable list of r
        increment augmented instantiation size for r
      }
      retrieve the index k of bj in the variable list of r
      add position k to translation T
    }
    add subedge (v, vi) to e with translation T and
      optional negation from ai
  }
}
```

Current Implementation and Future Work

The reduced hypergraph representation for logic programs has been implemented using C++ and STL. This representation has been used to perform deduction by way of the Bilattice algorithm [Sei97] for finding a well-founded model, and the results have been verified using the previous implementation of **INDED**. In the future, the hypergraph representation will be augmented to allow for induction.

The reduced hypergraph representation also provides a firm basis for the parallelization of logic programming algorithms. Because of its reduced size, the hypergraph can be constructed on each of the slave nodes in a multiprocessor or distributed system. The master node can then distribute a set of constant instantiations to each of the slave nodes to constitute their work load for processing. Our future work will include the parallelization of these hypergraph algorithms using MPI on a Beowulf cluster.

References

[Doy79] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[GL90] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, 1990.

[K.B95] K.Berman, et. al. Computing the well-founded semantics faster. *Lecture Notes in Artificial Intelligence*, 928, 1995.

[LD94] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming.* Ellis Horwood, Inc., 1994.

[Mug92] Stephen Muggleton, editor. *Inductive Logic Programming.* Academic Press, Inc, 1992.

[Sei97] Jennifer Seitzer. *A study of the well-founded and stable logic programming semantics.* PhD thesis, University of Cincinnati, 1997.

[Sei99] Jennifer Seitzer. **INDED**: A symbiotic system of induction and deduction. In *MAICS-99 Proceedings Tenth Midwest Artificial Intelligence and Cognitive Science Conference*, pages 93–99. AAAI, 1999.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol I.* Computer Science Press, Inc., 1988.

[VRS91] A. VanGelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.