# The Parallelization of a Knowledge Discovery System with Hypergraph Representation*

Jennifer Seitzer, James P. Buckley, Yi Pan, and Lee A. Adams

Department of Computer Science
University of Dayton, Dayton, OH 45469-2160

**Abstract.** Knowledge discovery is a time-consuming and space intensive endeavor. By distributing such an endeavor, we can diminish both time and space. System **INDED**(pronounced "indeed") is an inductive implementation that performs rule discovery using the techniques of inductive logic programming and accumulates and handles knowledge using a deductive nonmonotonic reasoning engine. We present four schemes of transforming this large serial inductive logic programming (ILP) knowledge-based discovery system into a distributed ILP discovery system running on a Beowulf cluster. We also present our data partitioning algorithm based on locality used to accomplish the data decomposition used in the scenarios.

## 1 Introduction

Knowledge discovery in databases has been defined as the non-trivial process of identifying valid, novel, potentially useful, and understandable patterns in data [PSF91]. Data mining is a commonly used knowledge discovery technique that attempts to reveal patterns within a database in order to exploit implicit information that was previously unknown [CHY96]. One of the more useful applications of data mining is to generate all significant associations between items in a data set [AIS93]. A discovered pattern is often denoted in the form of an IF-THEN rule (IF *antecedent* THEN *consequent*), where the antecedent and consequent are logical conjunctions of predicates (first order logic) or propositions (propositional logic) [Qui86]. Graphs and hypergraphs are used extensively as knowledge representation constructs because of their ability to depict causal chains or networks of implications by interconnecting the consequent of one rule to the antecedent of another.

In this work, using the language of logic programming, we use a hypergraph to represent the knowledge base from which rules are mined. Because the hypergraph gets inordinantly large in the serial version of our system [Sei99], we have devised a parallel implementation where, on each node, a smaller sub-hypergraph is created. Consequently, because there is a memory limit to the size of a storable hypergraph, by using this parallel version, we are able to grapple with problems

* This work is partially supported under Grant 9806184 of the National Science Foundation.

involving larger knowledge bases than those workable on the serial system. A great deal of work has been done in parallelizing unguided discovery of association rules originally in [ZPO97] and recently refined in [SSC99]. The novel aspects of this work include the parallelization of both a nonmonotonic reasoning system and an ILP learner. In this paper, we present the schemes we have explored and are currently exploring in this pursuit.

## 2 Serial System INDED

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.

### 2.1 Inductive Logic Programming

Inductive logic programming (ILP) is a new research area in artificial intelligence which 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. The following definitions are from [LD94].

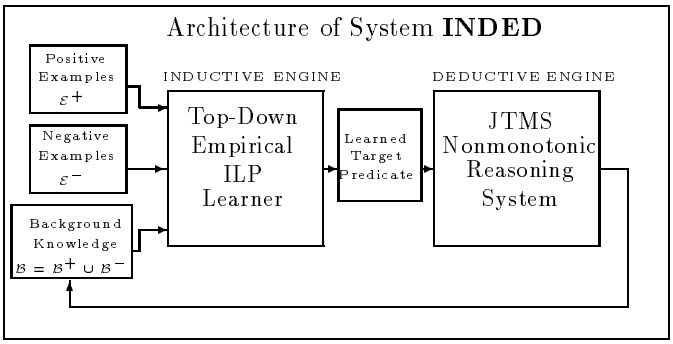
**Definition 2.1 (coverage, completeness, consistency)** *Given background knowledge $\mathcal{B}$, hypothesis $\mathcal{H}$, and example set $\mathcal{E}$, hypothesis $\mathcal{H}$* **covers** *example $e \in \mathcal{E}$ with respect to $\mathcal{B}$ if $\mathcal{B} \cup \mathcal{H} \models e$. A hypothesis $\mathcal{H}$ is* **complete** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if all positive examples are covered, i.e., if for all $e \in \mathcal{E}^+$ , $\mathcal{B} \cup \mathcal{H} \models e$. A hypothesis $\mathcal{H}$ is* **consistent** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if no negative examples are covered, i.e., if for all $e \in \mathcal{E}^-$ , $\mathcal{B} \cup \mathcal{H} \not\models e$.*

**Definition 2.2 (Formal Problem Statement)** *Let $\mathcal{E}$ be a set of training examples consisting of* **true** *$\mathcal{E}^+$ and* **false** *$\mathcal{E}^-$ ground facts of an unknown (target) predicate $\mathcal{T}$. Let $\mathcal{L}$ be a description language specifying syntactic restrictions on the definition of predicate $\mathcal{T}$. Let $\mathcal{B}$ be background knowledge defining predicates $q_i$ which may be used in the definition of $\mathcal{T}$ and which provide additional information about the arguments of the examples of predicate $\mathcal{T}$. The ILP problem is to produce a definition $\mathcal{H}$ for $\mathcal{T}$, expressed in $\mathcal{L}$, such that $\mathcal{H}$ is complete and consistent with respect to the examples $\mathcal{E}$ and background knowledge $\mathcal{B}$. [LD94]*

### 2.2 Serial Arichitecture

System **INDED** (pronounced "indeed") 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 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 internal ground instantiation represented internally as the deduction engine hypergraph.

## 3 Parallelizing INDED

Our main goals in parallelizing **INDED** are to obtain reasonably accurate rules faster and to decrease the size of the internal deduction hypergraph so that the

[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.

problem space is increased. The serial version is very limited in what problems it can solve because of memory limitations. Work has been done in the direct partitioning of a hypergraph [KAK99]. In our pursuit to parallelize **INDED**, however, we are exploring the following schemes where indirect hypergraph reductions are performed. Each of the following scenarios has been devised to be implemented on a Beowulf cluster [Buy99] using MPI [GLS99]:

1. large grained control parallel decomposition where one node runs the induction engine while another node runs the deduction engine
2. large grained control parallel decomposition where a pipeline of processors are established each operating on a different current state as created in previous (or subsequent) pipelined iterations
3. data parallel decomposition where each node runs the same program with smaller input files (hence smaller internal hypergraphs).
4. speculative parallel approach where each node attempts to learn the same rule using a different predicate ranking alogrithm in the induction engine.

## 4 Naive Decomposition

In this decomposition, we create a very coarse grain system in which two nodes share the execution. One node houses the deduction engine; the other houses the induction. Our strategy lets the induction engine initially discover a target predicate from positive and negative examples and an initial background knowledge base. Meanwhile, the deduction engine computes the current state using initial input files. This current state is sent to the induction engine as its background knowledge base in the subsequent iteration. The learned predicate from the induction engine from one iteration is then fed into the deductive engine to be used during the next iteration in its computation of the current state. This is then used as the background knowledge for the induction engine during the subsequent iteration.

In general, during iteration $i$, the induction engine computes new intensional rules for the deduction engine to use in its computation of the current state in iteration $i + 1$. Simultaneously, during iteration $i$, the deduction engine computes a new current state for the induction engine to use as its background knowledge in iteration $i + 1$. The above process is repeated until all target predicates specified have been discovered. As we extend this implementation, we expect to acquire a pipe-lined system where the deduction engine is computing state $S_{i+1}$ while the induction engine is using $S_i$ to induce new rules (where $i$ is the current iteration number).

## 5 Data Parallel Decomposition with Data Partitioning

In this method, each worker node runs **INDED** when invoked by a master MPI node [GLS99]; each worker executes by running a partial background knowledge

base which, as in the serial version, is spawned by its deduction engine. In particular, each worker receives the full serial intensional knowledge base (IDB) but only a partial extensional knowledge base (EDB). The use of a partial EDB creates a significantly smaller (and different) hypergraph on each Beowulf worker node. This decomposition led to a faster execution due to a significantly smaller internal hypergraph being built. The challenge was to determine the best way to decompose the serial large EDB into smaller EDB's so that the rules obtained were as accurate as those learned by the serial version.

### 5.1 Data Partitioning and Locality

In this data parallel method, our attention centered on decomposition of the input files to reduce the size of any node's deduction hypergraph. We found that in many cases data transactions exhibited a form of locality of reference. Locality of reference is a phenomenon ardently exploited by cache systems where the general area of memory referenced by sequential instructions tends to be repeatedly accessed. Locality of reference in the context of knowledge discovery also exists and should be exploited to increase the efficiency of rule mining. A precept of knowledge discovery is that data in a knowledge base system are nonrandom and tend to cluster in a somewhat predictable manner. This tendency mimics locality of reference. There are three types of locality of reference which may coexist in a knowledge base system: spatial, temporal, and functional. In spatial locality of reference, certain data items appear together in a physical section of a database. In temporal locality of reference, the data items that are used in the recent past appear in the near future. For example, if there is a sale in a supermarket for a particular brand of toothpaste on Monday, we will see a lot of sales for this brand of toothpaste on that day. In functional locality of reference, we appeal to a semantic relationship between entities that have a strong semantic tie that affects data transactions relating to them. For example, cereal and milk are two semantically related objects. Although they are typically located in different areas of a store, many purchase transactions of one, include the other. All three of these localities can be exploited in distributed knowledge mining, and help justify the schemes adpoted in our implementations discussed in the following sections.

### 5.2 Partitioning Algorithm

To retain all global dependencies among the predicates in the current state, all Beowulf nodes receive a full copy of the serial IDB. The serial EDB, the initial large set of facts, therefore, is decomposed and partitioned among the nodes. The following algorithm transforms a large serial extensional database (EDB) into $p$ smaller EDB's to be placed on $p$ Beowulf nodes. It systematically creates sets based on constants appearing in the positive example set $\mathcal{E}^+$ . Some facts from the serial EDB could appear on more than one processor. The algorithm is of linear complexity requiring only one scan through the serial EDB and positive example set E+.

**Algorithm 5.1 (EDB Partitioning Algorithm)** This algorithm is $O(n)$, where $n$ is the number of facts in the EDB.

**Input:** Number of processors $p$ in Beowulf
Serial extensional database (EDB)
Positive and negative example set $\mathcal{E}^+$ , $\mathcal{E}^-$
**Output:** $p$ individual worker node EDB's

*BEGIN ALGORITHM 5.1*
For each example $e \in \mathcal{E}^+ \cup \mathcal{E}^-$ Do
For each constant $c \in e$ Do
create an initially empty set $S_c$ of facts
Create one (initially empty) set $S_{none}$ for facts that have
no constants in any example $e \in \mathcal{E}^+ \cup \mathcal{E}^-$
For each fact $f \in$ EDB Do
For each constant $c' \in f$ Do
$S_{c'} = S_{c'} \cup f$
If no set exists for $c$ then
$S_{none} = S_{none} \cup f$
Distribute the contents of $S_{none}$ among all constant sets
Determine load balance by summing all set cardinalities
to reflect total parallel EDB entries $K$
Define $min\_local\_load = \lceil K/p \rceil$
Distribute all sets $S_{c_i}$ evenly among the processors
so that each processor has an EDB of roughly
equal cardinality such that each node has EDB of
cardinality $\geq min\_local\_load$ as defined above.
*END ALGORITHM 5.1*

## 6 Global Hypergraph using Speculative Parallelism

In this parallelization, each Beowulf node searches the space of all possible rules independently and differently. All input files are the same on all machines. Therefore, each worker is discovering from the same background knowledge base. Every rule discovered by **INDED** is constructed by systematically appending chosen predicate expressions to an originally empty rule body. The predicate expressions are ranked by employing various algorithms, each of which designates a different search strategy. The highest ranked expressions are chosen to constitute the rule body under construction. In this parallelization of **INDED**, each node of the Beowulf employs a different ranking procedure, and hence, may construct very different rules.

We are considering two possibilities for handling the rules generated by each worker. In the first, as soon as a process converges (finds a valid set of rules), it broadcasts a message to announce the end of the procedure. When the message is received by other processes, they are terminated. The other possibility we

are considering is to combine the rules of each worker. Different processes may generate different rules due to the use of different ranking algorithms. These rules may be combined after all the processes are terminated, and only good rules are retained. In this way, not only can we speed up the mining process, but we can also achieve a better and richer quality of solutions.

## 7 Current Status and Results

The current status of our work in these parallelization schemes is as follows. We have implemented the naive decomposition and enjoyed a 50 per cent reduction in execution time. Thus far, however, the bulk of our efforts have centered on implementing and testing the data parallel implementation on an eleven node Beowulf cluster. Here, we also experienced a great reduction in execution time. Figure 1 illustrates the consistent reduction of time as the number of nodes increased. The problem domain with which we are currently experimenting relates to the diagnosis of diabetes. The accuracy of the discovered rules by the cluster has varied. The rule learned by serial **INDED** is

```
inject_insulin(A) <-- insulin_test4(A) .
inject_insulin(A) <-- iddm(A) .
```

We attribute the variance of rule accuracy by the clusters to our partitioning algorithm. We anticipate extensive refinement of this algorithm as we continue this work.

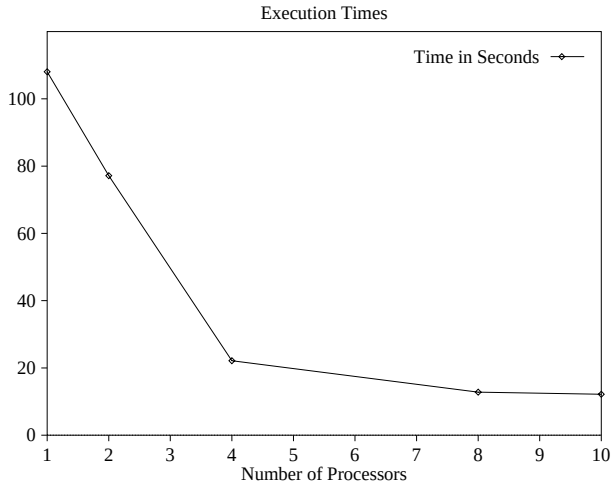



**Fig. 1.** Performance of Rule Mining on a Cluster.

## 8 Current and Future Work

We have delineated four parallelization schemes of transforming a large serial reasoning system that both deduces new facts as well as discovers new rules. In successfully implementing two of these schemes,we have found that one of the most interesting problems of parallel rule discovery is effective partitioning of the data. We have performed experimentation with one algorithm and anticipate extensive experimentation with new partitioning algorithms of the EDB and background knowlege. Additionally, we are currently implementing the speculative parallelization scheme discussed above, and are enhancing and devising new predicate ranking algorithms used by the induction engine.

## References


[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Bulletin*, pages 207–216, May 1993.

[Buy99] Rajkumar Buyya. *High Performance Cluster Computing Programming and Applications*. Prentice-Hall, Inc, 1999.

[CHY96] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 1996.

[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.

[GLS99] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI; Portable Parallel Programming with Message Passing Interface*. The MIT Press, 1999.

[KAK99] G Karypis,R. Agrawal, V. Kumar. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transactions of VLSI Systems*, 7(1),pages 69–79, Mar 1999.

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

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

[PSF91] Piatetsky-Shapiro and Frawley, editors. *Knowledge Discovery in Databases*, chapter Knowledge Discovery in Databases: An Overview. AAAI Press/ The MIT Press, 1991.

[Qui86] J. R. Quindlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[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.

[SSC99] L. Shen, H. Shen, and L. Chen. New algorithms for efficient mining of association rules. In *7th IEEE Symp. on the Frontiers of Massively Parallel Computation, Annapolis, Maryland*, pages 234–241, Feb 1999.

[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.

[ZPO97] M. J. Zaki, S. Parthasarathy, and M. Ogihara. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1:5–35, 1997.