# Learning Model Transformation Rules from Examples: The GAILP System

**Hamdi A. Al-Jamimi and Moataz A. Ahmed**,
aljamimi@kfupm.edu.sa moataz@kfupm.edu.sa
King Fahd University of Petroleum and Minerals Dhahran, 31261, Kingdom of Saudi Arabia

**Summary**
Learning by examples refers to acquiring knowledge and experience to generalize theory from existing examples. Inductive logic programming (ILP) uses inductive inference to generate hypotheses from examples given with a background knowledge. ILP systems have been successfully applied in a number of real-world domains. Several ILP systems were introduced in the literature. Each system uses different search strategies and heuristics; however, most systems employed a single predicate learning approach, which is not applicable in many learning problems. In this paper, we present GAILP, an ILP system that overcomes this limitation. GAILP employs genetic algorithms to discover various aspects of combinations to induce a set of hypotheses. It appraises such combinations in different ways to extract the most generic ones. The paper presents a thorough evaluation of the foundational aspects of the learning capability of GAILP. Two experiments were conducted to learn software model transformation rules. Experimental results reveal that GAILP is superior to a prominent ILP system, namely ALEPH, in different aspects; and specifically in learning multi-predicates incrementally. We used a case study of tasks from the automated software engineering domain. The results obtained for the "class packaging" task showed that the accuracy of GAILP was 0.88 comparing with 0.83 achieved by ALEPH. Similarly, for "introducing Façade interface" task, the accuracy obtained using GAILP and ALEPH were 0.90 and 0.66 respectively.

***Key words:***
*Inductive logic programming; inductive learning; model transformation; generalization.*

## 1. Introduction

Learning by examples refers to acquiring knowledge and experience to generalize theory from existing examples, i.e., observations. The examples represent a significant source of knowledge as they reflect the experience acquired by experts and practitioners. Inductive learning is a model building process where the data are analyzed to discover hidden patterns and draw conclusions from the available data. Thus, the induced patterns do not only describe the given examples, but also similar situations. Accordingly, they can be used to make estimation about unseen instances. The inductive learning programs have substantial applications in several science and engineering domains [1]–[3]. ILP has been employed successfully in extracting comprehensible and accurate rules in both scientific and industrial problems such as bioinformatics [4], medicine [5]–[7] as well as in other areas like web semantic [8], [9] and software engineering [10], [11] . They can be utilized in two basic modes: (i) as interactive tools for knowledge acquisition from explicit examples, and (ii) as part of an expert system in which other components provide the required learning examples for the inductive program.

Inductive logic programming (ILP) uses inductive inference to generate hypotheses from examples presented with the background knowledge. ILP systems can automatically construct first-order rules from background knowledge and training examples. ILP is defined as a subfield of machine learning (ML) which utilizes logic programming to represent the examples, background knowledge and induced hypotheses [12], [13]. Different ILP systems have been introduced in the literature. They differ in the way to learn from the given examples. However, some limitations were reported about the current ILP systems. For instance, ALEPH [14] and some other ILP systems [15] require manual declarations of the modes in advance to the learning process. *Mode declaration* means identifying in details the predicates that are expected to appear in the induced hypothesis. However, it is not practical in all cases to predefine such modes [16].

The ILP learning problem can be viewed as a search problem for rules that deduce the training examples [13]. ILP represents the given data and learning problem in first-order logic. Input data for learning can be represented by sets of positive, and negative examples in addition to background knowledge describing the given examples. The power of using the background knowledge is that it gives the domain experts the opportunity to select and integrate the appropriate knowledge that describes the given examples and the undertaken domain. The general ILP learning problem can be defined as follows:

***Given*** *a background knowledge B containing definitions of predicates and a set of positive* $E^+$ *and negative* $E^-$ *examples where* $E = E^+ \cup E^-$ .
***Find*** *a hypothesis H such that* $\forall e^+ \in E^+$ *s.t.* $B \cup H \models e$ *and* $\forall e^+ \in E^-$ *s.t.* $B \cup H \not\models e^-$ *i.e., the background theory together with the hypothesis entails all positive examples and none of the negative examples.*

Manuscript received April 5, 2019
Manuscript revised April 20, 2019

The generalization induces a generic hypothesis that covers the positive examples with sidestep the negative ones. That is, ILP attempts to find a hypothesis which is consistent with the positive and negative examples of given facts (background knowledge). In ILP terminology, a rule covers the set of facts if it is consistent with every positive and negative example. H is a complete w.r.t. the background knowledge and the given examples in case it covers all instances in E+ set. In another way, H is defined as a consistent w.r.t. the background knowledge and the given examples if it does not cover any negative example. We have recently presented a brief proposal of new genetic algorithms (GA) based ILP system, namely GAILP, to overcome some of the limitations of previous systems such as mode declaration, and requirement of negative examples [17]. In this paper, we discuss the technical details and the validity of GAILP. We present the capability of GAILP and compare its performance against the most prominent system currently available in the literature namely ALEPH. The results confirmed the incremental learning ability of GAILP to detect the suitable order of predicates and to revise the knowledge theory through utilizing the positive examples. GAILP was also shown to support the learning of multiple predicates. This ability distinguishes GAILP from other systems which only consider an iterative procedure for learning the rules one by one where the order is provided manually.

This paper is organized as follows: Section 2 presents a motivational example illustrating the main ILP concepts and focusing on the limitation of most ILP systems. It also presents the fundamental aspects of GAILP system. Section 3 introduces the experiment setup including the rules generation methodology, the data and evaluation measurements. The induction results are discussed in Section 4, while Section 5 concludes the paper and gives some directions for further research.

## 2. GAILP: the Challenge and Solution

### 2.1 A Motivational Example

In the following, we present a real-world scenario that illustrates the concept of our model. Consider a task of learning concepts of family relationships namely aunt(X, Y) and uncle-in-law(X, Y). The former states that person X is an aunt of person Y, while the latter states that person X is uncle-in-law of person Y. The training examples consist of positive examples and negative examples. Each positive example is a fact of the predicate aunt or uncle-in-law that is known to be true, while each negative example indicates a pair of people are not connected by the aforementioned relationships. This task will be used as a running example throughout this paper. Fig. 1 shows an example of family relations, while Table 1 demonstrates the representation of the relations in first-order logic form. The background knowledge includes parent, brother, and husband relationships, which are represented by two predicates. The output of an ILP algorithm could be the following "learnt" rules expressed as a Horn clause.

***aunt (X, Y) :- brother (Z, X ), parent (Z, Y), female(X)***
***uncle-in-law(X,Y):- aunt(Z,Y), husband(X,Z)***

The first declarative rule simply states that if a person Z is a brother of X, Z is a parent of Y, and X is female, then X is an aunt of Y. On the other hand, the second rule states that if a person X is a husband of Z, and Z is an aunt of Y then X is an uncle-in-law of Y.

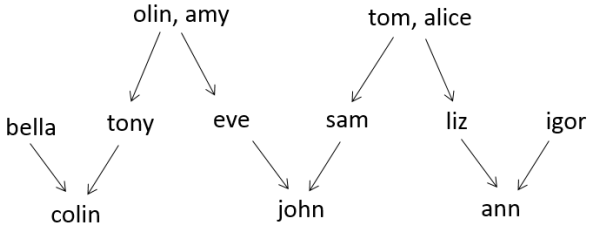


Fig. 1 Example of family relations

Table 1. The first-order logic form of relationship concepts learning problem

| | |
|---|---|
| ***Background Facts*** | brother(sam,liz),brother(tony,eve)<br>husband(olin,amy),husband(tom,alice)<br>husband(sam,eve),husband(igor,liz)<br>female(liz), female(eve),<br>female(amy), female(alice),<br>female(bella), male(tom),<br>male(sam),… |
| ***Positive Examples*** | aunt(liz,john)<br>aunt (eve, colin)<br>uncle-in-low(igor,john)<br>uncle-in-law(sam,colin) |
| ***Negative Examples*** | aunt(liz,sam)<br>aunt (bella, tony)<br>uncle-in-low(olin,colin)<br>uncle-in-law(tom,igor) |

In order to induce such rules, the ILP system generally conducts separated runs where each is fed with positive and negative examples as well as a set of background knowledge facts describing the given examples. In each run, the ILP engine constructs generic rules whose structure is predefined. The rule's head is specified using given examples predicate and arity, while its body depends on the given facts. Each set of similar predicates are generalized independently, so the currently used ILP systems rely on the user to identify dependency, feed the sets for generalization, and to revise the knowledge facts as needed. In the shown example "aunt" rule should be induced firstly to be able to utilize the results in inducting the predicate "uncle-in-law". However, in some cases, the user is not aware of the dependency between the sets. This motivated us to propose

a new ILP system that assists to automatically identify the rule structure, discover the potential dependency and revise the theory after each run.

## 2.2 GAILP System

GAILP is an ILP system that uses first-order logic as the concept definition language and generates a set of definite clauses. It employs a coverage algorithm in constructing clause definition. To investigate all possible combinations among the learning examples, GAILP employs two approaches brute-force and genetic algorithms. It is able to batch the background knowledge and all the positive examples one time to induce all the rules. In contrast, the existing systems at each run allow feeding only one set of examples having the same type declaration. The architecture of GAILP, depicted in Fig. 2, starts by taking the background facts and the examples for learning. It then constructs the ground bottom clauses (GBC) for all the given examples. A random population is generated for selecting the GBC groups to execute the substitution process that ends up by a substituted bottom clause (SBC) for the first GBC. The constructed SBC is considered the bottom base to substitute the remaining GBCs. The substitution process is viewed as a constraint satisfaction problem. Eventually, for all the GBC groups there are equivalent substituted clauses. Afterwards, a random population is generated to select a set of SBCs. Then one SBC is randomly selected to determine the search space for starting generalization process. A detailed description of the induction steps is provided in the following.

*Hypothesis Space Construction*

ILP system is a search problem in which a number of candidate solutions, called the search space, will be found and evaluated. The generalization process starts by constructing the bottom clauses for all given positive examples to determine the search space. The target bottom clause is constructed by using the relevant atoms detected in the background knowledge.

*Building the Bottom Clauses*

The proposed system differentiates between two versions of the bottom clauses: ground and substituted bottom clauses. In the sequel, we refer to them as gbc and sbc respectively. For each positive example, $e_i^+$, GAILP constructs a $gbc_i$ ($\forall\ e^+ \in E^+ \Rightarrow gbc$) where $gbc_i$ contains all facts known to be true about the current example $e_i^+$. The background knowledge facts are used to obtain the related atoms to a particular $e_i^+$. Hypothetically, the process results in a number of bottom clause equals the number of positives presented in the present group. It is supposed to consider only one $E^+$ group by an iteration.

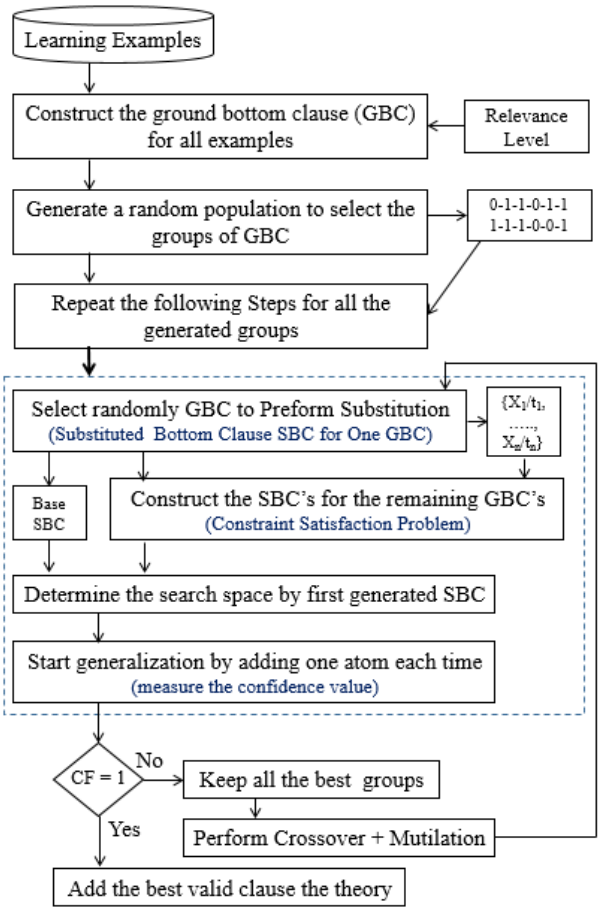


Fig. 2 Overview of GAILP's architecture

First, it takes the examples one by one. For a particular example $e^+ \in E^+$, the algorithm searches for all related atoms existing in the background knowledge. The atom is related directly to $e^+$ when one of its arguments appears in $e^+$. The construction of $gbc$ is determined by the language bias, if there is any. The given bias describes some restrictions to be considered in order to build the target bottom clause. Thus, our algorithm provides a parameter, called relevance level shown in Fig. 2. It allows consider/ignore the indirect related atoms. The relevance level value guides the search for the related atoms in the background knowledge to construct the hypothesis. Other indirect related atoms can be added based on the setting of the relevance level. The user can adjust this parameter to determine the need to involve the indirect related atoms in the construction process. The relevance level (L) can get one of three values 1, 2, or 3. With $L = 1$ only the literals having, as input variables, input variables of the head (layer 0) are added to the most-specific clause. At layer i only literals having input variables appearing in layer $i - 1$ (as output or input variables) can be constructed. It is important to note that with a low value for L not all facts from the

background knowledge will appear in a most-specific clause. When having a target instance, $pred(arg_1, arg_2, \dots, arg_n)$, the related atoms are searched in the background knowledge. An atom $a_i$ is a direct related atom to $pred(arg_1, arg_2, \dots, arg_n)$ if it shares a constant argument $arg_i$ appearing in the target instance. For example, given the target instance, $a(x, y)$, and the background $facts = \{ b(x, z), c(z, k, l), d(y, m)\}$. The facts $b(x, z)$ and $d(y, m)$ are considered direct related atoms. In contrast to direct related atoms, an atom $a_i$ is an indirect related atom to $pred(arg_1, arg_2, \dots, arg_n)$, if doesn't share any argument $arg_i$ with the given target instance. In the previous examples the fact $c(z, k, l)$ is seen as indirect related to the target instance, $a(x, y)$. However, the type of indirect related atoms might be involved in the bottom clause.

*Bottom Clauses Substitution*

A substitution is defined as the operation that replaces variables occurring in each $gbc$ by terms (values). A substitution θ can be defined as a finite set of the form: $\{x_1/t_1, \dots\dots, x_n/t_n\}, n \geq 0$, where the $x_i$ are distinct variables and the $t_i$ are terms, i.e. $t_i$ is substituted for $x_i$.

In this regard, we use the alphabet letters $\{A, B, C, \dots\}$ to achieve the substitution operation. Given a set of the ground most-specific clauses resulting from the previous operation. It is supposed to replace each variable of the $gbc$ by a unique value. The aim is to apply a particular substitution to a set of expressions (values or bottom clauses) to make them identical. We apply this substitution to make a particular clause ($gbc_i$) more specific to be matched with another clause ($gbc_j$). For the sake of suitable substitution for all the constructed gbc, the problem is solved as a constraint satisfaction problem (CSP [19]). So that, the problem can be expressed in the following form; given a set of variables $\{x_1, x_2, \dots\dots x_n\}$ and a finite set of possible terms $\{t_1, t_2, \dots\dots t_p\}$ that can be assigned to variable $x_i$, and set of constraints $\{y_1, y_2, \dots\dots y_m\}$. The constraint $y_i$ involves a subset of the variables and identifies the possible combinations of terms for that subset. That is, the problem can be defined by an assignment of terms to some or all of the variables such that $\{x_i = t_i,\ x_j = t_j,\ \dots\}$ with satisfying constraints.

For a particular set of *gbc*, the substitution simply starts by selecting one of the clauses $gbc_r$ (randomly) to perform a direct substitution for all its variables (i.e. with no constraints). Then the resulting substituted bottom clause is used as the set of constraints to be satisfied when applying substitution on the remaining $gbc_i$ belong to the present set. The algorithm follows the backtracking search strategy, where it selects term for one variable at a time and backtracks when a variable with the assigned term does not satisfy the constraints.

*Search for the Hypothesis*

Before starting the search of the candidate hypotheses, different combinations of the resulting bottom clauses are generated. Upon the number of the generated bottom clauses one of the two approaches is employed to find the possible combinations. A brute-force approach is used directly to find all possible combinations in case the number of bottom clauses is relatively small. The number of possible combinations increases exponentially with the number of given samples. Genetic algorithms approach is therefore used when there is a significantly increasing number of possible combinations.

GA approach generates a random population representing particular combinations, and then several generations can be found. Each combination is considered individually to perform clauses generalization and to learn the expected hypotheses. After that, from each presented set of bottom clauses, a random bottom clause is selected first to determine the hypothesis space. The combination operation results in several sets of bottom clauses. The concern here is to generalize one hypothesis in each set to include a number of bottom clauses in the specified set.

In turn, it covers the corresponding positive examples. In each set all the bottom clauses share the same head, thus one of them will be used as the head of the generalized rule. Then in the same set we search for the commonalities of the atoms among the different bottom clauses. It is expected to induce one or more rules from each set. Algorithm 6 demonstrates the steps followed to accomplish this operation.

The rationale behind considering several combinations for induction is to give another view about the given examples. Thus, from each combination set, regardless of the used approach, one bottom clause is randomly selected to establish the hypothesis space. The head of the selected bottom clause is managed as the head of the candidate hypothesis. Several iterations are carried out to add atoms to the hypothesis with measuring the confidence value (CF), shown in Fig. 2, simultaneously. Upon the confidence value, the new atom can be kept in the candidate hypothesis or retracted. This represents the search operation to find the best hypothesis as a solution for the presented ILP problem.

For each given example, GAILP traverses the search space using the refinement operator, adding/changing a new literal one at a time within the language constraints, keeping clauses with the best compression at each refinement level along the way. All generated refinements must subsume the most specific clause under 0-subsumption (i.e. at least as general as the most specific clause).

## 3. Experiment Setup

### 3.1 Induction Methodology

To evaluate the induction capability of GAILP, two transformation experiments have been conducted. In each experiment, various models of a number of software systems were utilized. Each case study focuses on different learning approach. The first one considers the single-predicate learning while the second one takes into account the learning of multi-predicates. In both cases, GAILP system was used to induce the rules, however we considered further steps to assess and validate the obtained transformation rules. Fig. 3 demonstrates the steps followed in our methodology to conduct the experiments.

The declaration of modes that specify the structure of the intended hypotheses is optional. The learner can utilize the given predicates (as modes) or it can extract the head and body of the hypothesis by using the positive examples and the related atoms provided in the background knowledge. GAILP allows starting the induction by using positive examples along with the background knowledge. In some cases, the negative examples are not available as input. Thus using negative examples is an option to help pruning large parts of the search space.

To measure the induced rules' performance, each rule was evaluated individually against all the learning systems, batched together. The induced rules were ranked according to the achieved accuracy to compare their generalization capability. Next, the rules with outstanding accuracy can be assessed in order to find the most generic rules. This subset of rules can be utilized by the practitioners for transforming new given source instances (using appropriate tools for application). A particular threshold can be specified to specify the subset offered the optimal performance. Afterward, the candidate rules, with different combinations, are applied on the learning systems one by one. GA-based procedure can be used to determine the best possible combination.

Another perspective is considered to determine the most generic rules. The number of times each rule was applied to form the best combinations for the learning systems. The rules were ranked based on the frequency of their applications. Such a frequency measure was calculated in a form of percentage by dividing the application times by the total number of systems used for training. This percentage can be used as a guide when applying the rules on new unseen samples. Some rules have been selected several times, for instance Rule <17> was selected with 60% of the systems when looking for the best results. In contrast, some rules were not selected any time with learning examples, thus they were ignored.

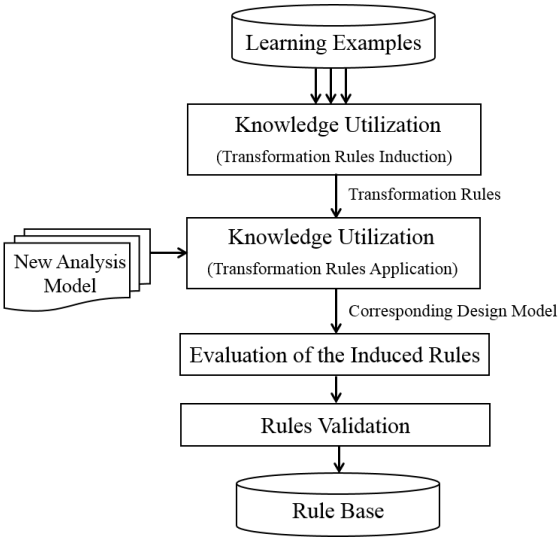


Fig. 3 Procedure to assess the induced transformation rules

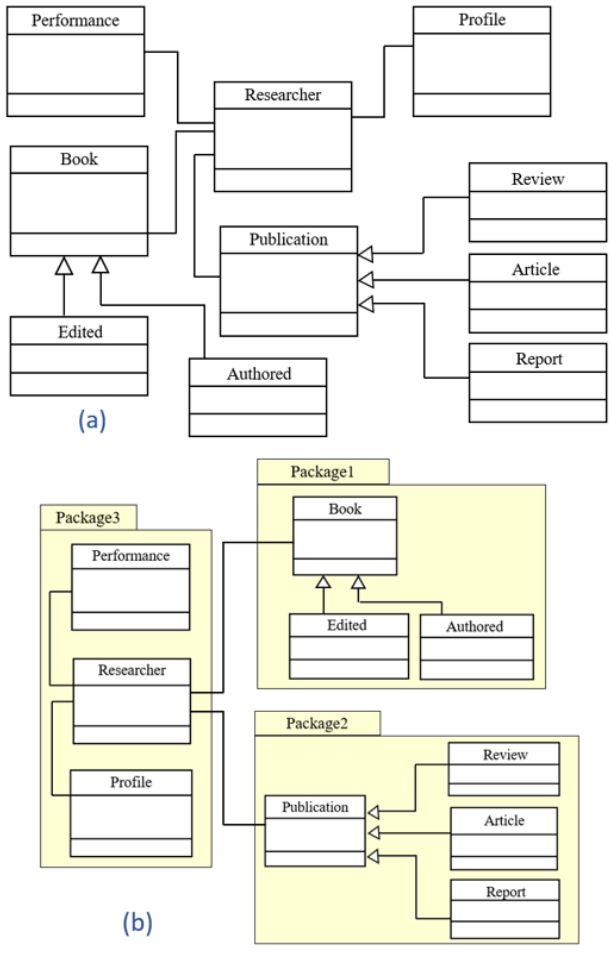


Fig. 4 The UML class diagram for analysis-design pair (a) the source model (requirement analysis) and (b) the target model (software design)

## 3.2 Case Studies

The presented case studies consider two substantial tasks to transform the software analysis model to the corresponding design models.

*Packaging Class Diagram*

Locating the classes into packages is one of the necessary tasks when moving from analysis to design. The analysis models present the class diagram depicting all system classes and the relations linking them. Such models are used to develop highly cohesive and loosely coupled packages. The aim of this case study is to learn packaging rules from analysis-design pair examples. Fig. 4 depicts a simple example the analysis model along with the corresponding initial design model after introducing the packages.

*Introducing Façades*

Another high-level software design activity is to introduce façades. It is common for a class in one package to have external relations with classes in other packages. A façade provides a one "point of contact" to a package of classes (i.e., component) which simplifies the interaction process and improves the overall design coupling and cohesion. It hides the implementation of the component from its clients, making the component easier to use.

Fig. 5 (a) depicts a class diagram that has many inter-packages relationships making the design highly coupled and less maintainable. Fig. 5 (b) shows the simplified design and loosely coupled model when using Façade interface.

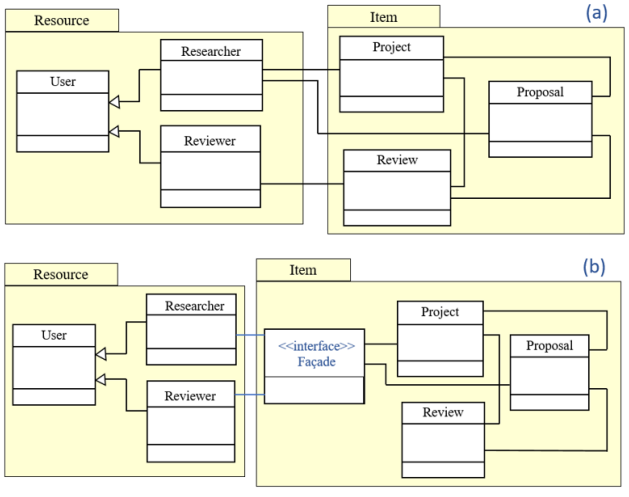


Fig. 5 Part of class diagram of application (a) the source model and (b) the target model

## 3.3 Experimental Data

The datasets used in the experiments comprises around 34 systems. Each system consists of the analysis and design models. These cases were collected mostly from academic projects, examples from textbooks, and by reserve engineering [18]. Each system consists of analysis/design pair. In turn, each design system comprises at least three packages. The total number of packages in the base is 217 while the total number of classes and interfaces is 1540. Table 2 shows a brief statistics of the systems' artifacts i.e., packages, classes, interfaces, and relationships (including association, aggregation, generalization, and realization).

Table 2 Statistical Analysis of the used datasets

| ***Margin*** | ***Min*** | ***Max*** |
|---|---|---|
| Packages | 3 | 27 |
| Classes and Interfaces | 10 | 151 |
| Relationships | 11 | 188 |

## 3.4 Evaluation Measurements

Different measures can be used to evaluate the produced rules and their applicability. The following performance measures were used to evaluate the generalization capacity of the induced transformation rules. It is worth mentioning that, the same measurements have been used in the literature to evaluate similar studies.

$$\text{Accuracy}_{p_j} = 100\ \frac{(TP_{pj} + TN_{pj})}{(TP_{pj} + TN_{pj} + FP_{pj} + FN_{pj})} \quad (1)$$

$$\text{Specificity}_{p_j} = \frac{TN_{pj}}{TN_{pj} + FP_{pj}} \quad (2)$$

$$\text{Precision}_{p_j} = \frac{TP_{pj}}{TP_{pj} + FP_{pj}} \quad (3)$$

$$\text{Recall or Sensitivity}_{p_j} = \frac{TP_{pj}}{TP_{pj} + FN_{pj}} \quad (4)$$

$$F - \text{measure}_{p_j} = \frac{2 * \text{Precision}_{pj} * \text{Recall}_{pj}}{\text{Precision}_{pj} + \text{Recall}_{pj}} \quad (5)$$

For the problem solved by ILP-based systems, usually the performance can be measured by grouping the results as true positive (TP), true negative (TN), false positive (FP) and false negative (FN). In the context of this work, TP refers to the correct target model artifacts generated by the transformation rules. FP indicates the incorrect target model artifacts generated by the transformation rules. FN denotes

the correct target model artifacts in the actual target model, but the transformation rules do not generate. Lastly, TN refers to the other model target artifact that were correctly ignored by the transformation rules (in grouping problems). Eq. (1)–(5) demonstrate the measures considered, through this work, to measure the induced rules performance.

## 4. Induction Results

This section presents conducted experiments that are in line with the above case studies. The first experiment considers packaging class diagrams to evaluate the single predicate learning. Likewise, the second experiment examines the multi-predicates learning capacity of the ILP system by studying the use of Façade interface.

### 4.1 Single Predicate Learning

A dataset consists of twenty-two systems was used to feed the GAILP to start the induction procuress. Each system consists of pairs of source- target models used as learning samples. Table 3 shows samples of the rules resulted from the induction. By using all learning samples, GAILP was able to induce 63 rules for packaging class diagrams. These rules were individually evaluated against the learning systems, batched together. Almost have of the rules showed a satisfactory accuracy.

Next, a threshold of 0.4 was chosen to specify the subset offered the optimal performance. As a result, the top twenty-four rules were selected for further assessment to determine the most generic rules. With considering different combinations, the rules were applied to the learning systems one by one. Table 3 demonstrates the accuracy measures in a training system-wise manner. The average of the accuracy measure equals to %93.

To assess the generalization of the transformation rules, we counted the number of times each rule was applied to form the best combinations for the learning systems. The rules have been applied with different frequencies. It is supposed to consider the high frequently applied rules to be validated further using unseen samples. A threshold of (.30) was considered to take the most frequent applied rules. Several validation runs have been conducted using a validation set that consists of twelve systems. In each run, a subset of the candidate rules were applied to the validation systems one by one. In the first run, only the top two rules were used, while the second run added one more rule to the subset, and so on. The averages of performance measures for the different runs using the validation systems are depicted in Table 3.

Table 3 Evaluating the induced rules using the learning and validation systems

| *System* | *Specificity* | *Recall* | *Precision* | *F-Measure* | *Accuracy* |
|---|---|---|---|---|---|
| ***Learning Systems*** | | | | | |
| **Average** | **0.96** | **0.77** | **0.94** | **0.83** | **0.93** |
| sys1 | 0.942 | 0.976 | 0.825 | 0.888 | 0.947 |
| sys2 | 0.892 | 1 | 0.833 | 0.903 | 0.933 |
| Sys05 | 1 | 0.5 | 1 | 0.667 | 0.833 |
| Sys07 | 1 | 0.731 | 1 | 0.842 | 0.944 |
| Sys08 | 1 | 0.694 | 1 | 0.819 | 0.9 |
| Sys09 | 0.8 | 0.56 | 0.8 | 0.654 | 0.753 |
| Sys10 | 0.944 | 0.889 | 0.944 | 0.903 | 0.939 |
| Sys12 | 0.969 | 0.818 | 0.923 | 0.858 | 0.935 |
| Sys14 | 1 | 0.746 | 1 | 0.851 | 0.938 |
| Sys15 | 1 | 0.74 | 1 | 0.839 | 0.95 |
| Sys17 | 0.987 | 0.836 | 0.944 | 0.877 | 0.962 |
| Sys18 | 1 | 0.725 | 1 | 0.817 | 0.975 |
| Sys19 | 0.638 | 0.642 | 0.638 | 0.627 | 0.667 |
| Sys20 | 1 | 0.772 | 1 | 0.87 | 0.97 |
| Sys22 | 1 | 0.642 | 1 | 0.764 | 0.978 |
| Sys23 | 1 | 0.738 | 1 | 0.843 | 0.967 |
| Sys26 | 0.97 | 1 | 0.933 | 0.963 | 0.978 |
| Sys28 | 0.955 | 0.599 | 0.955 | 0.719 | 0.936 |
| Sys29 | 1 | 0.748 | 1 | 0.85 | 0.964 |
| Sys30 | 1 | 0.859 | 1 | 0.923 | 0.976 |
| Sys31 | 0.981 | 0.831 | 0.921 | 0.868 | 0.954 |
| Sys32 | 1 | 0.889 | 1 | 0.933 | 0.967 |
| ***Validation Systems*** | | | | | |
| **Average** | **0.93** | **0.70** | **0.91** | **0.77** | **0.88** |
| Sys3 | 1 | 0.619 | 1 | 0.729 | 0.948 |
| Sys4 | 0.958 | 0.822 | 0.933 | 0.859 | 0.917 |
| Sys6 | 0.815 | 0.535 | 0.815 | 0.638 | 0.804 |
| Sys11 | 1 | 0.76 | 1 | 0.861 | 0.958 |
| Sys13 | 1 | 0.711 | 1 | 0.83 | 0.909 |
| Sys16 | 0.976 | 0.84 | 0.893 | 0.839 | 0.955 |
| Sys21 | 0.625 | 0.642 | 0.673 | 0.646 | 0.667 |
| Sys24 | 0.952 | 0.806 | 0.917 | 0.838 | 0.9 |
| Sys25 | 1 | 0.711 | 1 | 0.822 | 0.923 |
| Sys27 | 1 | 0.584 | 1 | 0.698 | 0.925 |
| Sys33 | 0.983 | 0.744 | 0.917 | 0.803 | 0.942 |
| Sys34 | 0.8 | 0.621 | 0.8 | 0.698 | 0.762 |

## 4.2 Multi-predicate Learning

One of the limitations reported about current ILP systems is their inability to deal with multi-predicates learning automatically [16]. "Introducing Façade interface" is an example of model transformation learning tasks that requires a multi-predicates learning approach. The experiments conducted, in this section, is to evaluate GAILP's capacity in terms of the multi-predicates learning. In the software modeling process, introducing a façade to a model means not only adding the façade interface, but also adding more than one artifact.

The new artifacts may include a new interface, association relationships linking the external clients to the interface, and others linking the interface to the classes placed in its package. Thus, the used ILP system should be able to learn multi-predicates hypotheses and discover the accurate order to start inducing the hypotheses.

The learning process started with the given background facts and the first group positives based on initial order. The system tried different orders until it found the most appropriate one. Fig. 6 presented samples of rules induced for introducing a façade, and its related relationships. The first group shows sample of the rules induced for introducing façade, while the other two groups shows the rules for the relationship (associations) artifacts.

```
packageHasFacade(A, B) ←
        package(A), packgeHasClass(A, C), packgeHasClass(A, D),
        interfaceFacade(B), association(E, F, A, C), association(E,
        G, A, D),  packgeHasClass(E, F), packgeHasClass(E, G).

packageHasFacade(A, B) ←
        package(A), packgeHasClass(A, C), packgeHasClass(A, D),
        interfaceFacade(B), association(E, F, A, C), association(E,
        G, A, C), packgeHasClass(E, F), packgeHasClass(E, G),
        association(H, I, A, D), packgeHasClass(H, I).

packageHasFacade(A, B) ←
        package(A), packgeHasClass(A, C), packgeHasClass(A, D),
        packgeHasClass(A, E), interfaceFacade(B), association(F,
        G, A, D), association(F, H, A, E), packgeHasClass(F, G),
        packgeHasClass(F, H), association(I,J, A, E),
(a)     association(I,K, A, F),  packgeHasClass( I, J),
        packgeHasClass( I, K).

accociationFromClasstoFacade(A, B, C, D) ←
        packgeHasClass(A, B), packageHasFacade(C, D),
(b)     association(A, B, C, E), packgeHasClass(C, E).

accociationFromfacadetoClass(A,B,A,C) ←
        packgeHasClass(A,C), packageHasFacade(A,B),
        packgeHasClass(D, E), association(D, E, A, C),
(c)     accociationFromClasstoFacade(D,E, A, B).
```

Fig. 6 Samples of the induced transformation rules - Introducing Façade (a) Façade artifact, (b) Association between a client and façade, (c) Association between a façade and a class

It is worth mentioning that, the last two rules are supposed to be applied whenever there is application of any rule in the first group. The induced transformation rules were applied (one by one) to the learning systems (source models) and the results were compared against the actual target models. The rules evaluation focused only on the first group of the transformation rules presented in Fig. 6. When applied the rules on the learning examples, some rules are applied perfectly many times, while others might be used one time. As specified in the methodology, the rules were ranked according to their application frequency as several combinations have been assessed to find the best results. A subset of the rules have been selected based on a threshold, (30% was determined to take into account most frequent applied rules). Only five rules were applied on more than 30% of the presented systems. Further validation, using the validation dataset, was conducted for the candidate rules as top generic induced rules. The results obtained when applying the transformation rules on unseen systems are shown in Table 4. Obviously, GAILP was able to learn efficiently multi-predicates hypothesis and to discover accurately the order of the induced rules (f-measure = 0.90).

Table 4 Evaluating the induced rules using the learning and validation systems

| ***System*** | ***Recall*** | ***Precision*** | ***F-Measure*** |
|---|---|---|---|
| ***Learning Systems*** | | | |
| **Average** | **1** | **0.955** | **0.965** |
| Sys4 | 1 | 0.96 | 0.95 |
| Sys17 | 1 | 1 | 1 |
| Sys25 | 1 | 0.98 | 0.94 |
| Sys21 | 1 | 0.95 | 0.98 |
| Sys11 | 1 | 1 | 1 |
| Sys12 | 1 | 0.75 | 0.85 |
| Sys14 | 1 | 1 | 1 |
| Sys27 | 1 | 1 | 1 |
| ***Validation Systems*** | | | |
| **Average** | **1** | **0.88** | **0.90** |
| Sys5 | 1 | 0.97 | 0.95 |
| Sys7 | 1 | 1 | 1 |
| Sys30 | 1 | 0.67 | 0.72 |
| Sys2 | 1 | 0.93 | 0.94 |
| Sys23 | 1 | 0.86 | 0.92 |

## 4.3 Comparisons with ALEPH

We evaluated the capability of GAILP in learning model transformation rules. Two main transformation tasks have been considered, namely: packaging class diagrams and introducing Façade interface. In this paper we showed that

GAILP overcomes limitations encountered when using traditional ILP systems. Through this work, we conducted a comparison between GAILP and ALEPH within the context of the model transformation. Several experiments were conducted on a considerable large dataset consists of 34 systems that vary in size and were collected from different sources. The dataset was divided into independent sets for learning and validation purposes.

Hamdi and Ahmed have recently reported results of using ALEPH against a software modeling problem, namely packaging class diagram. They observed some limitations such as the prior declarations of modes and types to start the learning process. The same learning and validation datasets were used when testing both systems. Therefore, we considered it acceptable to compare performance through quality and accuracy of the induced transformation rules. For the single predicate learning, on most cases ALEPH was able to produce rules for packaging the class diagrams based on the given examples. However, it required a rigorous preparation of the files and models declarations in each run. For packages with many classes and complicated relations, ALEPH could not find a solution using different settings of search. For the multi-predicates learning, ALEPH was not an efficient in inducing the required rules as it manipulates the problem as a regular single predicate learning [19].

It is obvious, GAILP showed a better performance in both tasks. The validation results for the packaging task showed that the accuracy of GAILP was 0.88 comparing with 0.83 achieved by ALEPH, Fig. 7 (a). Similarly, for the second task, the f-measure obtained using GAILP was 0.96, while the f-measure of ALPEH was 0.66, Fig. 7 (b).

Definitely, the number of induced rules varies from one system to another. For packaging problem, GAILP was able to induce 63 rules comparing with 25 rules induced via ALEPH. The increase of the induced rules came from the coverage of all the positive examples using GA combinations. We explained that, not all the induced rules can be stored in the rule base for future development. The rules were evaluated in different ways and validated to select the best subset to be stored in the rule base. That mean only 12 out of 63 rules are recommended to be stored in the rule base. It is important to mention that, for introducing Façade interface the results obtained from GAILP came from a combination of applying façade and associations rules. However, the comparison with rules induced by ALEPH for façade interface is reasonable. The reason is that the application of the association rules is a consequence of the façade rules application. In other words, the associations' artifacts will be added correctly to the model if the façade interface was added to the right package. Similarly, if there is a missed façade interface in the model, consequently all the related associations will not be added.

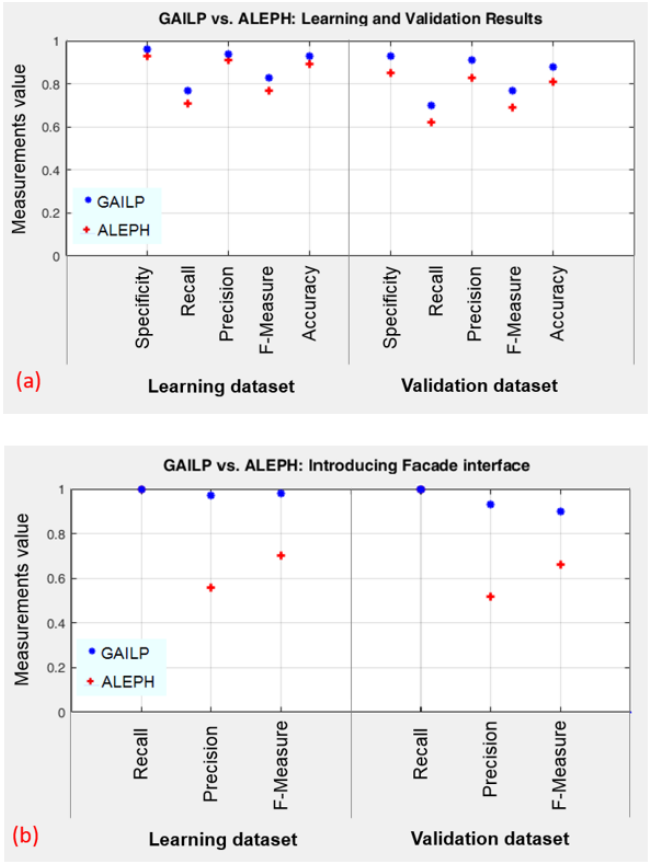


Fig. 7 (a) Introducing packages to the class diagram, (b) Introducing Façade design pattern

From the previous experiments, we noticed that GAILP and ALEPH have common characteristics where there are many problems that can be solved correctly using both systems such as classification, learning transitive rules, arch problem without modifications, and packaging of class diagrams. However, ALPEH could not find solutions for multi-predicates learning problems. As pointed out by our experimental results, GAILP is not meant to be a general-purpose ILP system; rather it is meant to be domain specific in the sense that when having multi-predicates that can be batched together to learn different types of hypotheses with different number of arity.

## 5. Conclusion and Future Work

In this paper, we presented a GA-based ILP system, GALIP. We evaluated the capability of GAILP in learning model transformation rules. Two main software engineering transformation tasks have been considered, namely: packaging class diagrams and introducing Façade interface. Through this work, we conducted a comparison between GAILP and ALEPH within the context of the model transformation. Several experiments were conducted on a

considerable large dataset consists of 34 systems that vary in size and were collected from different sources. The dataset was divided into two independent sets; one for learning and one for validation. Experiments showed that GAILP outperforms ALEPH. The validation results for the packaging task showed the accuracy of GAILP to be 0.88 comparing with 0.83 achieved by ALEPH. Similarly, for introducing Façade interface task, the f-measures obtained using GAILP and ALEPH were 0.90 and 0.66 respectively. As pointed out by our experimental results, GAILP is not meant to be a general-purpose ILP system; rather it is meant to be more appropriate for domains having multi-predicates that can be batched together to learn different types of hypotheses with different number of arity. As a future work, we will use GAILP to induce transformation rules with considering other transformation problems

**Acknowledgment**

This research was funded by the National Plan for Science, Technology and Innovation (MAARIFAH) – King Abdulaziz City for Science and Technology – through the Science & Technology Unit at King Fahd University of Petroleum & Minerals (KFUPM) – the Kingdom of Saudi Arabia, award number 11-INF1633-04.

## References

[1] J. C. A. Santos, H. Nassif, D. Page, S. H. Muggleton, and M. J. E. Sternberg, "Automated identification of protein-ligand interaction features using inductive logic programming: A hexose binding case study," *BMC Bioinformatics*, vol. 13, no. 1, p. 162, 2012.

[2] U. Schmid, S. H. Muggleton, and R. Singh, "Approaches and Applications of Inductive Programming (Dagstuhl Seminar 17382)," in *Dagstuhl Reports*, 2018, vol. 7, no. 9.

[3] I. Bratko and S. Muggleton, "Applications of inductive logic programming," *Commun. ACM*, vol. 38, no. 11, pp. 65–70, 1995.

[4] N. Amir, D. Cohen, and H. J. Wolfson, "DockStar: A novel ILP-based integrative method for structural modeling of multimolecular protein complexes," *Bioinformatics*, vol. 31, no. 17, pp. 2801–2807, 2015.

[5] S. Ushikubo, K. Kanamori, and H. Ohwada, "Extracting time-oriented relationships of nutrients to losing body fat mass using inductive logic programming," in *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2016 IEEE 15th International Conference on*, 2016, pp. 226–230.

[6] Y. Qiu *et al.*, "Knowledge discovery for pancreatic cancer using inductive logic programming," *IET Syst. Biol.*, vol. 8, no. 4, pp. 162–168, 2014.

[7] T. Op De Beéck, A. Hommersom, J. Van Haaren, M. van der Heijden, L. O. Jesse Davis, and I. Nagtegaal, "Mining hierarchical pathology data using inductive logic programming," in *Proceedings of the 15th Conference of Artificial Intelligence in Medicine*, 2015.

[8] H. Karimi and A. Kamandi, "A learning-based ontology alignment approach using inductive logic programming," *Expert Syst. Appl.*, vol. 125, pp. 412–424, 2019.

[9] R. Lima, B. Espinasse, and F. Freitas, "OntoILPER: an ontology-and inductive logic programming-based system to extract entities and relations from text," *Knowl. Inf. Syst.*, vol. 56, no. 1, pp. 223–255, 2018.

[10] D. Varró and Z. Balogh, "Automating Model Transformation by Example Using Inductive Logic Programming," in *SAC'07*, 2007.

[11] H. A. Al-Jamimi and M. A. Ahmed, "Knowledge acquisition in model driven development transformations: An inductive logic programming approach," in *TENCON 2014- 2014 IEEE Region 10 Conference*, 2014, pp. 1–6.

[12] S. Muggleton, "Inductive logic programming," *New Gener. Comput.*, vol. 8, no. 4, pp. 295–317, 1991.

[13] S. Muggleton and L. De Raedt., "Inductive logic programming: Theory and methods," *J. Log. Program.*, vol. 19, pp. 629–679, 1994.

[14] A. Srinivasan, "The Aleph Manual," *Univ. Oxford*, 2007.

[15] Y. Li, M. Niu, and J. Guo, "An Inductive Logic Programming Algorithm Based on Artificial Bee Colony," *J. Inf. Technol. Res.*, vol. 12, no. 1, pp. 89–104, 2019.

[16] H. A. Al-Jamimi and M. A. Ahmed, "Learning Requirements Analysis to Software Design Transformation Rules By Examples: Limitations of the Current ILP systems.," in *5th IEEE International Conference on Software Engineering and Service Science (ICSESS 2014)*, 2014.

[17] M. Al-Jamimi, Hamdi A. ; Ahmed, "A genetic algorithm-based ILP incremental system," in *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, 2017.

[18] H. A. Al-Jamimi, "A New ILP System for Model Transformation by Examples," King Fahd University of Petroleum and Minerals, 2015.

[19] H. A. Al-Jamimi and M. A. Ahmed, "Model Driven Development Transformations using Inductive Logic Programming," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 11, pp. 531–541, 2017.