Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2012, Article ID 893701, 14 pages
doi:10.1155/2012/893701

Research Article

Genetic Programming for Automating the Development of Data Management Algorithms in Information Technology Systems

Gabriel A. Archanjo and Fernando J. Von Zuben

Laboratory of Bioinformatics and Bioinspired Computing, School of Electrical and Computer Engineering, University of Campinas (Unicamp), 13083-970 Campinas, SP, Brazil

Correspondence should be addressed to Gabriel A. Archanjo, archanjo@dca.fee.unicamp.br

Received 11 December 2011; Revised 23 March 2012; Accepted 27 March 2012

Academic Editor: Phillip Laplante

Information technology (IT) systems are present in almost all fields of human activity, with emphasis on processing, storage, and handling of datasets. Automated methods to provide access to data stored in databases have been proposed mainly for tasks related to knowledge discovery and data mining (KDD). However, for this purpose, the database is used only to query data in order to find relevant patterns associated with the records. Processes modelled on IT systems should manipulate the records to modify the state of the system. Linear genetic programming for databases (LGPDB) is a tool proposed here for automatic generation of programs that can query, delete, insert, and update records on databases. The obtained results indicate that the LGPDB approach is able to generate programs for effectively modelling processes of IT systems, opening the possibility of automating relevant stages of data manipulation, and thus allowing human programmers to focus on more complex tasks.

1. Introduction

Information technology (IT) systems have become the basis of process management of today's successful enterprises. We can find this kind of system in virtually all fields of activities and inside corporations of any size. The intensive adoption of IT systems has promoted the emergence of an entire ensemble of technologies and services to supply a wide range of demands.

Similar to what happens in other areas of product development, methodologies, processes, and tools have been enhanced over the years in order to improve the development of software products, which are going to promote increasing productivity and reduced costs. The first methodologies were inspired by principles found in other areas of product development, like manufacturing. However, the dynamic environment involved in software development is fostering a continuous improvement and customization of methodologies to embrace inevitable uncertainties and necessary redefinition of the product specification, resulting in an iterative and evolutionary process [1].

The need for more agile methodologies is promoting the development of enhanced tools and techniques, more notably in the field of code and design reuse. Approaches to automate entire modules of the software development or to support decision on software engineering have been explored. However, the automated generation of computer algorithms still remains restricted to the scientific field. Knowledge discovery and data mining (KDD) applications are associated with many different approaches to extract relevant patterns from datasets, including solutions based on programs generated automatically. However, since the most common representation of data in the academic field is the linear dataset, due to its simplicity, the majority of works have been focusing on this representation. Using this type of data organization, genetic programming was employed to a wide range of applications, for instance, financial market analysis [2], cancer molecular classification [3], and bankruptcy prediction [4]. Nevertheless, large databases normally used on IT systems do not store records linearly. A more sophisticated framework is necessary to organize records in advanced structural configurations, as found on relational or object-oriented databases. Freitas [5] has proposed a framework for applying GP for classification and rule induction using relational databases. Ryu and Eick

[6] have used MASSON to induce programs that show commonalities between objects stored in an object-oriented database, and to derive queries with the purpose of extracting intentional information [7].

These works on KDD and query generation use the database only to organize and retrieve information, so that the programs do not modify the records. This limitation is not a problem since the objective is to find patterns in static data. On the other hand, aiming at modelling processes of IT systems, the capability to modify records is mandatory for the generated programs, since most processes are inherently conceived to change the state of a set of entities in the system. In the case of a library management system, for instance, the process of borrowing a book change the state of the system, associating a user and a book with a loan. By the same reasoning, in the case of a financial IT system, a process devoted to money transferring modifies the records related to the accounts involved. The modelling of these two examples are considered as case studies in this work.

To implement these kinds of processes, programs have to be able to find the proper associations among information stored in distinct data structures (tables or objects) and manipulate them in a correct manner. This work proposes linear genetic programming for databases (LGPDB), a tool for automatic generation of programs that can query, delete, insert, and update records in relational databases, and shows experiments that illustrate the feasibility of the proposed approach for automating the development of data management algorithms in IT systems. A preliminary version of LGPDB was presented in Archanjo and Von Zuben [8].

The paper is organized as follows. Section 2 presents an overview of the evolution of software development, covering three fields: methodologies, tools, and search-based software engineering. linear genetic programming for databases (LGPDB), a tool for automatic generation of data management algorithms for IT systems, is described in Section 3. An experiment for generating computer programs to provide features for a simple library system is described in Section 4. Section 5 introduces new instructions and a method for inducing more complex programs. The influence of records without consistent relationship on the evolutionary process is addressed in Section 6. Finally, in Section 7, concluding remarks and future prospects are outlined.

2. Evolution of Software Development

Just like the development of any other product, many methodologies, tools, and principles for software development have been created over the years, aiming at cost reduction, quality improvement, and increasing productivity.

2.1. Methodologies. When a product is under development, the sequence of steps from conception to the final product should be carefully conceived, focusing on high levels of productivity and quality, as long as low levels of cost and risk. In the 1970s, the first methodology for software development was denoted the Waterfall model [9], which specifies a sequence of phases, from requirements to operations, to deliver a software product.

Some mistakes in the first version of the project requirements is probably associated with misleading decision making. In the earlier stages of a project, it is common that even customers or the product managers do not know every specific detail of the project. Therefore, software prototyping was proposed to address this problem. Using this methodology, a software prototype is developed to answer open questions in the requirements that could only be managed by means of some experimentation. In this case, the client or any other person responsible for the software specification can interact with the prototype to check whether or not the most influential aspects of the project were properly addressed. This process can be repeated, producing multiple prototypes, until starting the product engineering.

Following the same idea of reducing risks and increasing interaction with experts who have the final assessment of the project, Boehm [10] has proposed the spiral model. In that model, phases like risk analysis, prototype development, and requirements are distributed over multiple cycles and, at the end of each one, the result is validated by people responsible for outlining the project. Extending these ideas, in the last decade, a new software development paradigm has gained attention, the agile software development (ASD) [11], in which iterative and evolutionary processes are the core idea. ASD breaks the project into multiple small cycles with constant analysis and customer feedback to reduce uncertainties surrounding the development of the product, decreasing costs, improving quality, and delivering products in the desired time range.

2.2. Tools and Techniques. Continuous software development and improvement, a core principle embraced by modern and more agile methodologies, require initiatives to increase productivity and to minimize the impact caused by frequent revisions in the software product, prevalent during the development phase. Although some tools presented in this section were proposed before the last revolution in the software development methodologies, they have gained more importance in software projects after the adoption of more agile methods.

Considering the high demand for IT systems, it is inevitable that multiple systems share similar features. Software engineers and programmers involved in system development across different project domains solve the same or similar problems during the development. These scenarios naturally promote the emergence of techniques to reuse software solutions. The most well-known strategies include the identification of high demanded features and the creation of a set of customizable software components, known as frameworks [12]. Currently, there are frameworks for almost everything, from user authentication to optical character recognition. Since frameworks add significant value to software development products, their creation is commercially exploited. Consequently, there are software companies that focus solely on developing frameworks. Software algorithms are not the only element that can be reused. Taking into account the importance of software architectural design, it was proposed the concept of design pattern [13] which is a catalogue of designs matured and

validated in real-world applications, organized to be reused. Other notable strategy for software reuse has emerged with the success of the Internet. Instead of providing features via frameworks that have to be tightly coupled to applications, the features are provided as web services [14]. The rise of hardware and software services for broad integration of tools, features, and purposes, available on the web, has created a new computational concept, known as cloud computing [15].

Nowadays, in order to store and manage information about its processes, IT systems use a relational database which represents the information in a scalar and structured fashion. On the other hand, this kind of system is usually developed using object-oriented (OO) programming languages or similar programming paradigms that manage information in a nonscalar fashion, using composite variables like arrays and lists, hash maps, records, and objects. Therefore, both the database format and the algorithm that load and store information from the database should obey some consistency rules, so that changes promoted in one of them will be followed by corresponding changes in the other. Since this requisite must be handled by most IT systems, a technique called object-relational mapping (ORM) [16, 17] was proposed to minimize the amount of effort allocated to this task. In one instance of these approaches, instead of promoting pairwise changes in format and algorithm for each situation, an XML file is used to store the mapping between objects and database tables. A generic purpose framework does the conversion using that XML file. Thus, to change the mapping, it is only necessary to change the XML file.

Other issue that impacts productivity and costs in software development involves quality assurance processes. Methods to automate parts of the software test were proposed to improve testing procedures. One of the most-well known techniques is the automated unit testing that uses assertions to test if a source unit (i.e., the smallest testable part of the software) is working properly. For each unit, it is created a test case. A tool for this purpose, like JUnit [18], is used to run all tests and generate a report automatically. This testing strategy is receiving more and more attention, being employed as the core concept of a software development methodology called test-driven development [19].

2.3. Search-Based Software Engineering. In virtually all human activities, there are problems in which the objective is to find out the best decision, given different choices and problem restrictions. This kind of situation can be modelled mathematically and solved applying mathematical optimization techniques, like classical optimization methods or metaheuristics. Search-based software engineering (SBSE) is a research field that embodies optimization methods in software engineering tasks [20]. For example, in the case of software effort estimation, a dataset containing attributes of finished software projects (e.g., number of transactions and entities, complexity, team experience) and the necessary effort was used in [21] to provide estimations for new projects. Neural networks, k-nearest neighbor, and genetic programming were employed to map the relationship between those attributes and the necessary effort. Moreover, SBSE has been applied to many other problems in software engineering, for instance, software testing [22], requirements [23], automated maintenance [24], and quality assurance [25]. The SBSE is a growing research field with a clear tendency of being incorporated into the next generation of real-world commercial software.

3. Linear Genetic Programming for Databases (LGPDB)

Section 2 has shown the evolution of software development, mainly by means of reusing solutions and automating processes. However, the task of transforming ideas, architecture designs, and processes into algorithms are predominantly being made by human programmers. Thus, an automatic method to generate this kind of algorithms is desired to improve the software development process and also to alleviate the burden usually assigned to human programmers.

The field of automated generation of computer programs is not new. In 1975, Holland [26] suggested the possibility of evolving genetic algorithm representations more similar to computer programs. In 1985, Cramer [27] evolved programs by means of genetic operators and natural selection. Finally, at the beginning of the 1990s, Koza [28] formalized the genetic programming (GP) as an extension of genetic algorithms designed specifically for program evolution. GP has been used for generating computer programs devoted to a wide range of applications, from robotics [29] to electrical circuit synthesis [30]. GP is an effective method to generate computer programs automatically. However, even with the extensive list of applications of GP, the usage of this approach on the IT environment is narrow and practically devoted exclusively to SBSE and KDD. It is important to mention the existence of other approaches for the automatic generation of computer programs with some restrictions. In the case of automatic programming [31], the objective is the generation of computer programs from a high level representation, specified by a human engineer, which models the solution. There are also many approaches for evolutionary programming where only the program parameters are optimized and not its structure. Finally, there are approaches that model solutions by combining automatically selected and parameterized algorithms [32]. However, in the case of this work, the capability of optimizing the parameters and the program structure in the level of instructions, using genetic programming, seems more appropriate for providing a general purpose tool for data manipulation.

Another relevant issue is the data representation since most works do not use a structured method. Generally, the datasets are composed of a sequence of attribute vectors. Nevertheless, IT systems usually have to organize information in more complex structures, involving multiple entities or objects. IT systems generally use a relational database to organize information. In [5], it was proposed a method for query generation, using a fixed query structure combined with a genetic program, to model relationships present in a relational database. However, in that work, the correct association among tables is fixed and it is informed *a priori*. Thus, the genetic program does not need to model

the correct path associating records stored in distinct tables, as it will be implemented in this work. Instead of using a relational database, Ryu and Eick [6, 7] have used an object-oriented database and a domain knowledge base to generate queries using MASSON. The relationship and interactions among objects in the database are also defined *a priori*. The use of an object-oriented database is also a barrier since it is not a common approach for IT system development.

This work proposes a method to automate the development of algorithms capable of manipulating records for information technology systems. A commonly used architectural approach for this type of systems is the MVC (model-view-controller) [33] in which the application is divided into three layers. The model layer is responsible for data representation and management, for instance, creating methods to store and request information from the database and to convert database representations into programming language representations. The controller is responsible for modelling processes that manipulate data, sometimes called "business logic." Finally, the view creates an interface from user interaction to the controller layer. Regarding this architectural approach, the LGPDB programs implement the layers, controller, and model.

In order to induce computer programs to manipulate entities stored in a database, LGPDB is composed of a simple relational database management system (DBMS) and a programming induction module based on linear genetic programming (LGP). The LGPDB architecture is illustrated in Figure 1. The program induction module (PIM) is the core of the system. It is responsible for evolving the candidate solutions. For each candidate program in the population at the current generation, it is executed fitness cases, scripts with the desired outcome for the target algorithm. The program execution environment is used by PIM to execute candidate solutions, operating on the database. PIM compares the outcome provided by the candidate program with the outcome provided by the scripts in the fitness cases. Thus, it is possible to measure the quality of a candidate solution, a necessary step in the evolutionary process. In the case of inducing programs that modify records, the solutions have to address three issues: (i) relate multiple tables on the database in order to associate the input attributes with the target records, (ii) filter the target records using the input attributes, (iii) modify the target records using the input attributes. In the case of programs for querying, only the first two issues have to be addressed. The entire process, from modelling a system to generating computer programs automatically, is addressed in detail in the next sections.

3.1. Database Management System. This module provides, for users and programs, interface to set up the database and manage the records stored on it. In comparison with traditional DBMS like PostgresSQL or MySQL, this module provides a smaller set of features. On the other hand, it was developed exclusively for program induction. Therefore, it combines some features for this purpose.

(i) *In-memory database:* since the induction process may involve thousands of candidate programs running at each generation, each one performing multiple operations on the database, it is important to make the operations as simple as possible. In this case, features for recovering from adverse situations are unnecessary since the induction process can be repeated after such situation. Most importantly, the database operates entirely on the RAM memory, reducing the access time for the records on the database [34].

(ii) *Database comparison:* when evaluating candidate solutions, it is important to know if the correct records were manipulated and the desired state of the database was achieved. Therefore, this module provides a feature for database comparison, returning the set of different records between the outcome provided by the candidate programs and the outcome provided by the validation scripts (fitness cases).

(iii) *Weakly typed:* to reduce the amount of concepts that candidate programs have to capture from data to model an information manipulation process, the database fields and program variables are weakly typed. Thus, numerical, categorical, and textual data are stored in a generic representation that can be accessed and modified by programs through a unique interface.

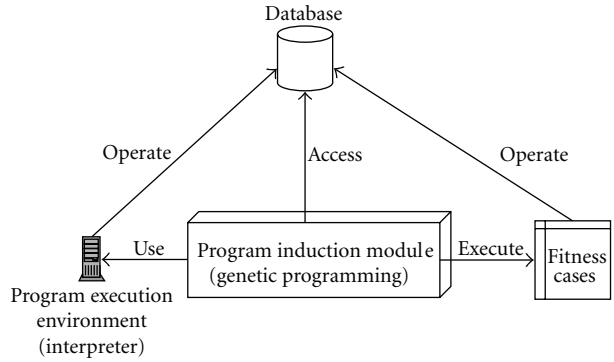


Figure 1: LGPDB architecture.

Another important feature provided by the DBMS is the concept of transaction, in which multiple operations can be executed atomically and reversed if desired, or in the case of errors [35], using the command *rollback*(). This command is extensively used in the induction process to restore the state of the database. After the evaluation of each candidate solution, the command *rollback*() is executed, then all candidate programs start operating the database at the same state.

3.2. Software Induction Environment. This module is devoted to generating computer algorithms automatically by employing an inductive learning approach with genetic programming. Having the database configured and the desired features modelled, it is specified a set of input variables and the desired outcome for each feature. In the induction process, programs compete with each other to solve the

problem. Mapping the relationship between the input and the outcome, genetic programs can model processes for information management.

3.3. Program Representation. Computer program can be interpreted as a set of instructions that tell a computer how to perform a task. Instructions can access and operate upon information stored in memories. Loops and conditional branches can be created using conditional and jump functions. These elements, present on low-level machine program representations like assembly, are also present in some sense in the majority of other representations, from C language to the ones used in GP. In the case of LGPDB, the program representation was adapted for information management tasks. Therefore, the first notable difference is on the instruction set, defined exclusively to manipulate information in a relational database, as listed below.

(i) Select(Table tb, ResultSet rs). Select all records in tb and put them in rs. Before this process, the result set is cleared, therefore, any existent information stored before is deleted.

(ii) Filter(ResultSet rs, Attribute attr, Rule r, InputValue v). Filter the result set rs using the rule r and the input value v for the attribute attr.

(iii) Related(ResultSet rs1, ResultSet rs2). For each record in rs1, if there is no foreign key associating it with a record in rs2, remove it.

(iv) UnRelated(ResultSet rs1, ResultSet rs2). For each record in rs1, if there is a foreign key associating it with a record in rs2, remove it.

(v) Delete(Table tb, ResultSet rs). Delete all records in tb with the same id of the records in rs. By definition for our problem, every entity has an attribute id.

(vi) CreateRelation(Table tb, ResultSet rs1, ResultSet rs2). If there is a foreign key associating records in rs2 with table tb, insert records in table tb associating them with the records in rs2 and store these records in rs1.

(vii) SetRelation(ResultSet rs1, ResultSet rs2). If there is a foreign key associating records in rs2 with records in rs1, set the association in the records in rs1 using the id's of the records in rs2, and update the records in rs1 on the database.

As can be seen, the LGPDB instructions do not operate using memory addresses or a generic type of variable. The LGPDB uses a strongly typed genetic programming representation [36] in which instruction parameters have a specific type and all programs in the population have instructions well parameterized, consequently reducing the search space. Each type of parameter has a specific purpose. Table represents the database tables. ResultSet is a data structure for database record management. Attribute is the possible attributes of a given ResultSet considering its table. Rule is used to compare two variables, given a comparison criteria such as "equal," "not equal," "greater," and "less." InputValue is any information specified by the user.

There are different program structures that might be used for GP programs such as tree, graph, and linear array. The program structure affects execution order, use and locality of memory, and the application of genetic operators [37]. In the case of LGPDB, instructions have many parameters and the data manipulated by programs are in the database or in global variables. The linear structure representation [38] has been chosen because of the use of memory and the similarity with imperative programming languages. Therefore, LGPDB programs can be interpreted by human programmers easily. Conditions and loops are implicitly available by means of the instruction Filter. Depending on the rule and the input value, it is determined whether a set of records should be kept or not inside the data structure ResultSet. Every instruction that manipulates a ResultSet also implements a loop to access each record inside that data structure. Compared with the program representation used by programming languages for general purpose, LGPDB representation is more restrictive. However, since LGPDB programs are generated automatically, the expressiveness of the language affects the size of the search space of solutions. In order to reach feasibility for the target problems, it was necessary to create a representation with restrictions and highly specific functions.

3.4. Evolutionary Process. As in any type of genetic algorithm, candidate solutions are evolved by means of natural selection. Candidate solutions compete, in an iterative process, generation after generation, to solve a problem. At each generation, solutions more adapted to solve the problem, given an evaluation criterion, are selected with more probability, to produce offspring to the next generation. Using this strategy, programs evolve along the generations until possibly reaching the desired solution. The first step in this process is the generation of the initial random population, given the maximum initial program size and the initial population size. The size of the program determines its complexity and the computational cost of its execution. The maximum size of a program can be increased along the generations, thus allowing more complex candidate solutions to be proposed in advanced stages of the evolutionary process. After the initialization, the iterative process that selects and creates a new population of candidate solution begins.

A selection strategy is necessary to implement the concept of survival of the fittest, so that individuals with the better fitness have a higher probability of being selected. LGPDB only selects the best individuals to generate offspring for the next generation, known as elitist strategy. The degree of adaptation of an individual for a given problem (fitness) is used to measure how far individuals are from the best solution. Since LGPDB employs a supervised and inductive learning strategy, the fitness of an individual is estimated based upon a set of examples containing input variables and the expected outcome, or simply *fitness cases*. In the induction database, the fitness cases are executed and the desired outcome is recorded. Candidate programs operate the induction database aiming at finding the correct sequence of instructions that changes the state of

the database in order to arrive at the same state of the database obtained with the fitness cases. Comparing the desired outcome, provided by the fitness cases, and the outcome provided by each candidate program, it is possible to determine how distant the candidate program is from achieving the correct solution.

In LGPDB, the fitness function $F(p)$, for a given distance $D(p)$ and a program p, returns 1 if the individual has solved the problem completely or a positive value smaller than 1 for partial solutions, as shown in:

$$F(p) = \frac{1}{1 + D(p)}. \quad (1)$$

LGPDB programs can manipulate records by means of four basic database operations: query, deletion, insertion, and updating. The desired outcome presented by the fitness cases determines which type of operations candidate solutions can perform and the distance measure used by the fitness function, since distinct distance measures are used depending on the type of manipulation. For querying, it is provided a set of examples E containing input information and the expected result. In the evaluation, it is considered the distance between the result provided by this set and the result provided by the program, stored in the ResultSet rs0 by default. As shown in (2), for querying, the number of false negatives FN between the expected result set and the result set provided by a candidate solution is more penalized, using a constant $\alpha > 1$, than the number of false positives FP. At the beginning, it is better to query all the data than to query a subset possibly missing the desired information. Thus, the solution is improved by filtering the entire dataset along the generations. This strategy is also used in [6, 7] and has the objective of stimulating the evolution of filtering techniques:

$$D_{\text{query}}(p) = \sum_{i=0}^{E} (\text{FN}_i * \alpha + \text{FP}_i). \quad (2)$$

Instruction Delete() deletes records passed in a ResultSet as a parameter, therefore, the problem of inducting a program to delete records can be seen as a querying problem. If the desired records to be deleted are in the parameter ResultSet, like the correct records in rs0 for querying problems, the deletion operation is performed correctly. However, by this reasoning, the distance estimation must prioritize the false negatives. At the beginning, solutions tend to delete all records. The correct set of records to be deleted emerges adding filtering instructions throughout the evolution of the programs. The distance estimation for deletion is presented, as follows:

$$D_{\text{delete}}(p) = \sum_{i=0}^{T} (\text{FN}_i + \text{FP}_i * \alpha). \quad (3)$$

Querying and deletion operations manipulate entire records. All attributes of the target records are necessarily involved in the operation, hence, it is only necessary to check whether or not a record is presented in the result for comparison. In the case of insertion and updating, records are not totally inserted or updated in a single operation. In the case of insertion, for instance, one instruction creates an initial record (CreateRelation) and another set the value of individual attributes (SetRelation), one at a time. Therefore, in order to determine a distance for two database states with inserted or updated records, it is necessary to compare them in the level of attributes. Before the induction process, the database states are compared, then inserted or modified records are listed. Therefore, when comparing the program outcome and the desired one, two elements are considered: if the program has manipulated the correct records, in the case of updating, and the Hamming distance (HD) between the two results. The HD is calculated using attributes of records to be compared. If all attributes are equal in the two records, the distance is zero, otherwise, it is given by the number of different attributes. The total distance between the two results is the sum of the HD between each compared pair of records. In the case of updating, records listed to be altered have their HD multiplied by α, penalizing programs that have not manipulated them.

After selecting interesting candidate solutions, based on the evaluation method, the next step is the creation of a new population of solutions. For this purpose, conventional genetic operators for mutation and crossover are applied. Given an individual, the reproduction operator generates x new individuals with the same genome. In the case of this work, it generates new programs with the same instructions and parameters. Then, given a probability for each operator, crossover and mutation take place to promote variation in the individuals. The crossover operation is used to combine blocks or sections of good solutions to generate new ones. LGPDB uses linear crossover in which, given two individuals, a segment is randomly selected in each individual and then exchanged. Mutation simply promotes a small variation in an individual. In the case of LGPDB, one of the following mutation operations is selected given a probability: (1) change instruction type, (2) change instruction parameter, (3) add an instruction in a random position, (4) change the instruction position, and (5) remove an instruction from a random position.

The steps above are combined to form the evolutionary process, shown in the flowchart presented in Figure 2. In the first step, a copy of the original database is created to be used in the induction process. Then, the initial random population is generated. In the next step, the iterative process is started and each individual is evaluated. Since all individuals start operating the database in the same state, after the evaluation of an individual, a *rollback*() operation is performed to recover the state of the database for the next individual. After the evaluation of all individuals, the selection process takes place. If the problem has not yet been solved by any individual in the population at the current generation or the maximum number of generations has not been achieved, the genetic operators are employed to generate a new population of individuals from the selected individuals and the process proceeds to the evaluation step. This iterative process is repeated until the satisfaction of the stopping criteria. When a candidate program succeeds in solving the problem, a final step removes unnecessary

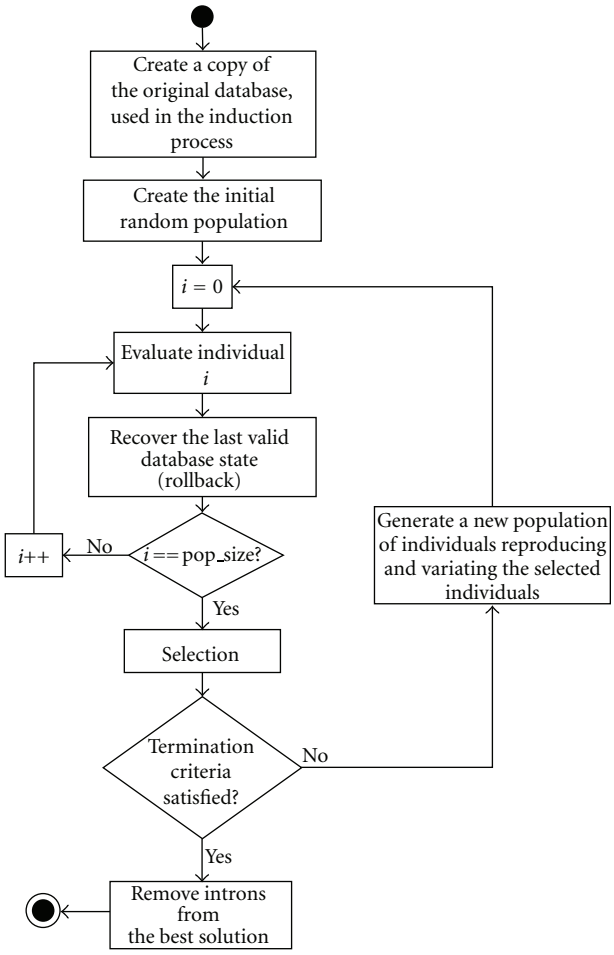


Figure 2: Flowchart of the LGPDB evolutionary process.

instructions, called introns. The presence of introns does not affect the outcome of the program. However, it might consume significant computational resources. Aiming at removing introns, it is verified whether or not the execution of each instruction of a program affects the program outcome. Instructions that overwrites unused variables or manipulates variables not associated with the program outcome should be discarded. Moreover, programs without introns are more parsimonious and easier to be interpreted by humans. The interpretability is relevant in scenarios in which human programmers interact with the tool to validate the automatically generated programs or even to adapt them so that they become capable of performing similar tasks.

4. Experiment with a Library System

The induction of features for a library system was the first application of LGPDB [8]. It was used to validate the main concepts and to identify weak points for improvements.

In this experiment, it was modelled a simple library management system containing the entities User, Author, Book, Periodical, Paper, Publisher, Tag, and Message. The relationship between these entities is intuitive, like user borrows books and periodicals that have authors and publishers. The entity-relationship diagram (ERD) [39] showing entities, attributes, and the relationships is presented in Figure 3.

Table 1: A subset of the database tables for a library system. The attributes in italic are primary or foreign keys.

Table	Fields
user	*id*, name, email, user, password
author	*id*, name
book	*id*, *publisher_id*, title, pages, isbn
paper	*id*, title
periodical	*id*, *publisher_id*, year, volume, issue, isbn
publisher	*id*, name
tag	*id*, value
message	*id*, message
bookLoan	*id*, *book_id*, *user_id*
periodicalLoan	*id*, *periodical_id*, *user_id*
authorBookRel	*author_id*, *book_id*
authorPaperRel	*id*, *author_id*, *paper_id*
tagRel	*id*, *book_id*, *paper_id*, *tag_id*
messageRel	*id*, message

The next step was to set up the relational database using the previously presented ERD to define the tables, attributes, and keys, as shown in Table 1. The database is composed of 14 tables, one for each entity and others to map many-to-many relationships among entities, like book and author, so that a book can be associated with multiple authors and vice versa.

After the initial setup, the database was populated with records, part of them is real data obtained on the Internet and others are hypothetical, used to simulate interesting relationships. Moreover, a few records without consistent relationships such as a book without author or a periodical without paper were inserted to promote the exploration of entity relationships by candidate solutions.

Finally, having defined and configured the database, the next step was the definition of which features have to be provided by the system, listed in Table 2.

As mentioned in Section 3.4, the LGPDB programs are evaluated under a set of fitness cases containing the input attributes and desired outcome for a given problem. Therefore, for each feature in Table 2, it is created a set of fitness cases. Table 3 shows an example of fitness case for the querying task Q4. The input is a tag previously inserted on the database, and the outcome is a set of books associated with that tag. Table 4 shows a fitness case for the deletion task D3. The input is the name of an author and the outcome is the deletion of books and papers. Table 5 shows an example for the insertion task I1. The input is the name of a book and a previously inserted message, and the outcome is the insertion of the records. Finally, for the updating task U1, Table 6 shows an example of fitness case. The input is the name of two publishers, the current and the new one. The outcome is the updating of the correct records.

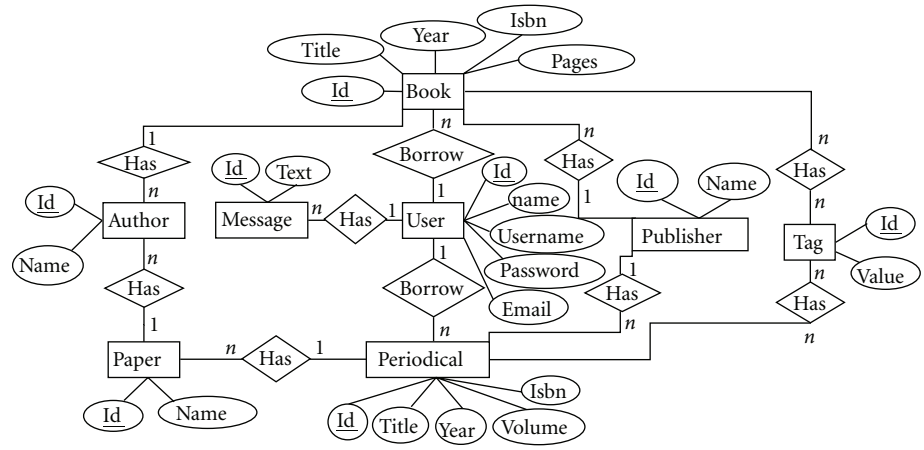


Figure 3: Entity-relationship diagram (ERD) for a simple library management system. Entities are drawn as rectangles, relationships as diamonds, and attributes as ovals.

Table 2: Features for the target system.

ID	Description
Q1	List books written by an author named X.
Q2	List users who borrowed books written by an author named X.
Q3	List users who borrowed a periodical containing a paper written by an author named X.
Q4	List books available for loan, having the tag X.
D1	Remove books published before the year X.
D2	Remove loan of a book titled X by the user Y.
D3	Remove books and papers authored by an author named X.
I1	Send the message Y to every user who borrowed a book titled X.
I2	Add a new tag Y for a book titled X.
U1	Update book publisher from X to Y.

Table 3: Example of a fitness case for the querying task Q4.

Feature	Q4
Description	List books available for loan, having the tag X.
Input	"Artificial Intelligence"
Type	Querying
Outcome	{"0," "Artificial intelligence: a modern approach," "0," "1132," "2009," "0137903952"}, {"1," "AI Programming Paradigms," "1," "946," "1991," "1558601910"}, {"2," "Machine Learning," "2," "432," "1997," "0070428077"}, {"3," "Genetic Programming," "3," "840," "1992," "0262111705"}

Table 4: Example of a fitness case for the deletion task D3.

Feature	D3
description	Remove books and papers authored by an author named X.
Input	"Peter Norvig"
Type	Deletion
Outcome	delete(book, 0) delete(book, 1) delete(paper, 142)

Table 5: Example of a fitness case for the insertion task I1.

Feature	I1
Description	Send the message Y to every user who borrowed a book titled X.
Input	"Machine Learning," "A new book is available with similar subject"
Type	Insertion
Outcome	Insert(messageRel, "2," "7") Insert(messageRel, "3," "8") Insert(messageRel, "4," "9")

Table 6: Example of a fitness case for the updating task U1.

Feature	U1
Description	Update book publisher from X to Y.
Input	"Springer," "Springer-Verlag"
Type	Updating
Outcome	Update(book, 13, "publihser_id," "10") Update(book, 14, "publisher_id," "10") Update(book, 15, "publisher_id," "10")

For each task, multiple fitness cases are used. The database instructions used as outcome manipulate the correct records, but without any association with the input information. The desired solution has to find the association between the input data and the records manipulated by the fitness cases, and the correct manipulations.

Table 7: Programs induced to provide the selected features for a library system.

ID	Program	Gen_{avg}	Gen_{std}	Gen_{median}	NC
Q1	select(author,rs3) select(authorBookRel,rs2) filter(name,equals,x,rs3) related(rs2,rs3) select(book,rs1) related(rs1,rs2)	75	34	79	0
Q2	select(author,rs2) filter(name,equals,x,rs2) select(authorBookRel,rs1) related(rs1,rs2) select(book,rs3) select(bookLoan,rs2) related(rs3,rs1) related(rs2,rs3) select(user,rs1) related(rs1,rs2)	283	161	265	0
Q3	select(author,rs3) select(paper,rs1) filter(name,equals,x,rs3) select(authorPaperRel,rs2) related(rs2,rs3) related(rs1,rs2) select(periodicalLoan,rs2) select(periodical,rs3) related(rs3,rs1) related(rs2,rs3) select(user,rs1) related(rs1,rs2)	430	205	365	0
Q4	select(tag,rs4) filter(value,equals,x,rs4) select(book,rs1) select(tagRel,rs2) related(rs2,rs4) related(rs1,rs2) select(bookLoan,rs3) unrelated(rs1,rs3)	216	134	198	0
D1	select(book,rs2) filter(date,less,x,rs2) delete(book,rs2)	28	31	14	0
D2	select(bookLoan,rs2) select(book,rs3) filter(title,equals,x,rs3) related(rs2,rs3) select(user,rs3) filter(name,equals,y,rs3) related(rs2,rs3) delete(bookLoan,rs2)	715	495	746	6
D3	select(paper,rs3) select(author,rs4) filter(name,equals,x,rs4) select(authorBookRel,rs1) select(book,rs2) relate(rs1,rs4) relate(rs2,rs1) select(authorPapelRel,rs4) select(author,rs1) delete(book,rs2) filter(name,equals,x,rs1) relate(rs4,rs1) relate(rs3,rs4) delete(paper,rs3)	402	374	279	3
I1	select(book,rs2) filter(title,equals,x,rs2) select(bookLoan,rs3) related(rs3,rs2) select(user,rs2) related(rs2,rs3) select(message,rs1) createRelation(messageRel,rs3,rs2) filter(text,equals,y,rs1) setRelation(rs3,rs1)	709	391	662	1
I2	select(tag,rs2) filter(value,equals,y,rs2) createRelation(tagRel,rs3,rs2) select(book,rs4) filter(name,equals,x,rs4) setRelation(rs3,rs4)	80	49	69	0
U1	select(publisher,rs2) filter(name,equals,x,rs2) select(book,rs4) select(publisher,rs3) filter(name,equals,y,rs3) related(rs4,rs3) setRelation(rs4,rs2)	209	201	151	0

Before the induction process, the following parameters have to be defined: number of available ResultSets nrs for manipulation, population size pop_{size}, maximum number of instructions in a program max_{size}, probability of crossover pcros, and probability of mutation pmut. Using the features Q1, I1, D1, and U1, empirical trials were performed looking for the configuration with the faster convergence to the correct solution, given by $nrs = 4, pop_{size} = 1000, max_{size} = 20$, pcros $= 0.3$, and pmut $= 0.9$. In fact, the convergence was not affected significantly by the crossover operator. A lower probability has been chosen in order to improve performance.

Finally, individual programs were induced to provide the features listed on Table 2. In Table 7, for each feature, it is shown one of the obtained induced programs and the average (Gen_{avg}), standard deviation (Gen_{std}), median (Gen_{median}) number of generations, and the number of attempts without convergence (NC) to the correct solution in 20 executions within the maximum of 2000 generations. The results indicate that LGPDB can induce very interpretable programs to query, delete, insert, and update records in relational databases. The induction of a program to perform the task D3 indicates that LGPDB can induce programs that alter multiple tables. Regarding the column "no convergence" (NC), only three tasks (D2, D3, I1), the evolutionary process has not obtained the correct solution in all 20 executions. In 6 cases for the D2 task, 3 cases for the D3 task, and 1 case for the I1 task, the population converged to a local optimal candidate solution, and the genetic operators were not capable of conducting the population to more promising regions in the search space, even increasing the maximum number of generations of the evolutionary process. Nevertheless, for practical applications, multiple executions, characterized by a distinct set of randomly initialized individuals at the first generation, can be made until the convergence to the correct solution is reached. In the case of the applications tackled in this work, it is important to note that partial solutions are not acceptable. Therefore, programs have to reach the correct solution to be useful. Usually, processes devoted to manage information on IT systems have to achieve all goals, otherwise, they will accumulate errors not tolerated in this kind of application.

On the other hand, the results highlight some limitations of LGPDB. Although it is not an objective of LGPDB to be a high-performance classifier, given its purpose, more powerful filtering rules are desired. The restriction of not generating single programs that can alter information in multiple ways, for instance, inserting and deleting, has also to be addressed. The next section describes initial efforts to overcome these limitations.

5. Adding New Instructions and Inducing More Complex Programs

The previous experiment has shown that LGPDB can induce programs to query, delete, insert and update records in a relational database. However, this experiment has raised some shortcomings of the first version of LGPDB, such as its limitation as a classifier and the impossibility to generate programs that operate multiple tables using different operations. In order to overcome or at least alleviate these limitations, some improvements were made.

The list below shows three new instructions added to the LGPDB instruction set. The first two are used to combine filtering rules, using the operators *AND* and *OR*, in order to model associations involving more input values or attributes. The third instruction is used to set attributes in ResultSets using input values, passed as parameters.

(i) addRule(Operator op, Attribute attr, Rule r, InputValue v, RuleObject ro). Add rule r, associated with the attribute attr and the input value v, to RuleObject ro with the operator op.

(ii) Filter_2(ResultSet rs, RuleObject ro). Filter the ResultSet rs using the combination of rules in the RuleObject ro.

(iii) setValue(Attribute attr, InputValue v, Operation o, ResultSet rs). Set the value of the attribute attr, for the records in rs, using the operation o and the input value v.

In fact, even without these new instructions, LGPDB can combine filtering rules using multiple Filter instructions, for instance, as shown in the induction of task D2. However, using multiple dissociated Filter instructions, LGPDB cannot model the *OR* operator. In order to demonstrate the combination of filtering rules using the *OR* operator, a new feature for the library system is proposed "List users who borrowed a book written by the authors X or Y." Fitness cases were created and a program was induced to model this feature, as shown in what follows:

```
select(author,rs3)
addRule(_,name,equals,X,rule1)
select(authorBookRel,rs2)
addRule(or,name,equals,Y,rule1)
filter_2(rs3,rule1)
select(user,rs1)
relate(rs2,rs3)
select(book,rs3)
relate(rs3,rs2)
select(bookLoan,rs4)
relate(rs4,rs3)
relate(rs1,rs4)
```

Beside the addition of new instructions, another improvement allows the induction of more complex processes, involving more tables and different operations. If a process has multiple steps that are only dependent on the input values, these distinct steps can be tackled by distinct programs, induced separately. The final solution is the concatenation of the programs. Following the same procedures as in the experiment presented in Section 4, an experiment is proposed here to induce a program to update and insert records in a hypothetical financial system. The first step was the creation of the database with tables and fields presented in Table 8.

Table 8: A subset of the database tables for a financial system. The attributes in italic are primary or foreign keys.

Table	Fields
Client	*id*, name, ssn, address, phone
Account	*id*, number, branch, *client_id*, balance
Saving	*id*, number, branch, *client_id*, balance
Transaction	*id*, operation, *account_id*, value, code

Table 9: Example of a fitness case for the feature F1.

Feature	F1
Input	999-1, 111-1, 999-2, 111-2, 500

Outcome	database.update(account, 3002, "balance", "1800");
	database.update(account, 3001, "balance," "1500");
	database.insert(transaction, 4001, "send," "3002," "500," "t01");
	database.insert(transaction, 4001, "recv," "3001," "500," "t01");

After setting up the database, the next step was the insertion of hypothetical records, part of them without consistent associations. In order to demonstrate how LGPDB can model a more complex process combining programs, the following task was chosen "Given the accounts A1 in the branch B1 and A2 in the branch B2, transfer the value V, from A1 to A2." For this task, multiple fitness cases similar to the one presented in Table 9 were created.

As can be seen, the outcome of the feature involves four database operations associated with the input information, two account updates and two transaction insertions. The first update adds the value 500 to the balance of the account 999-2, 111-2. On the other hand, the second update removes 500 from the balance of the account 999-1, 111-1. Finally, the other two subprograms insert logs to the transaction. This process is automatically divided into four induction process, one for each step. The final program that models the entire process is the concatenation of four LGPDB subprograms, induced separately, as shown in Algorithm 1.

6. Influence of Records without Consistent Relationships

Programs have to explore the relationships among different entities in the database to filter records and access new discriminant attributes. In order to list users who borrowed a book, for instance, it is necessary to relate tables user and bookLoan, thus filtering any user not associated with a loan.

```
//SubProg1
01: addRule(_,number,equals,B2,rule2)
02: select(branch,rs1)
03: filter_2(rs1,rule2)
04: Select(account,rs2)
05: addRule(_,number,equals,A2,rule1)
06: filter_2(rs2,rule1)
07: relate(rs2,rs1)
08: setValue(balance,V,+,rs2)
//SubProg3
09: clearEnvironment()
10: addRule(_,number,equals,B1,rule2)
11: addRule(_,number,equals,A1,rule1)
12: select(branch,rs4)
13: filter_2(rs4,rule2)
14: select(account,rs3)
15: relate(rs3,rs4)
16: filter_2(rs3,rule1)
17: setValue(balance,V,-,rs3)
//SubProg2
18: clearEnvironment()
19: addRule(_,number,equals,B1,rule1)
20: addRule(_,number,equals,A1,rule2)
21: select(account,rs4)
22: select(branch,rs2)
23: filter_2(rs4,rule2)
24: filter_2(rs2,rule1)
25: relate(rs4,rs2)
26: createRelation(transaction,rs2,rs4)
27: setValue(operation,send,=,rs2)
29: setValue(value,V,=,rs2)
30: setValue(id,transID,=,rs2)
31: //SubProg4
32: clearEnvironment()
33: addRule(_,number,equals,A2,rule1)
34: addRule(_,number,equals,B2,rule2)
35: select(branch,rs3)
36: select(account,rs4)
37: filter_2(rs4,rule1)
38: filter_2(rs3,rule2)
39: relate(rs4,rs3)
40: createRelation(transaction,rs1,rs4)
41: setValue(operation,recv,=,rs1)
42: setValue(id,transID,=,rs1)
43: setValue(value,V,=,rs1)
```

Algorithm 1: Final program that model the task F1. The four subprograms were induced separately and them concatenated.

This type of dissociation among entities is fully acceptable considering the system modelling.

However, other types of dissociation between entities are not expected like a book without author or a periodical without papers. If the database does not have dissociation of entities like that, programs are not rewarded to exploring these relationships, since relating authors with books does not filter any record, supposing that all authors are related with a book. Inserting hypothetical registers without consistent relationships, like a periodical without paper, programs are rewarded to explore this relationship, hence having access to higher levels of relationship and more discriminant attributes.

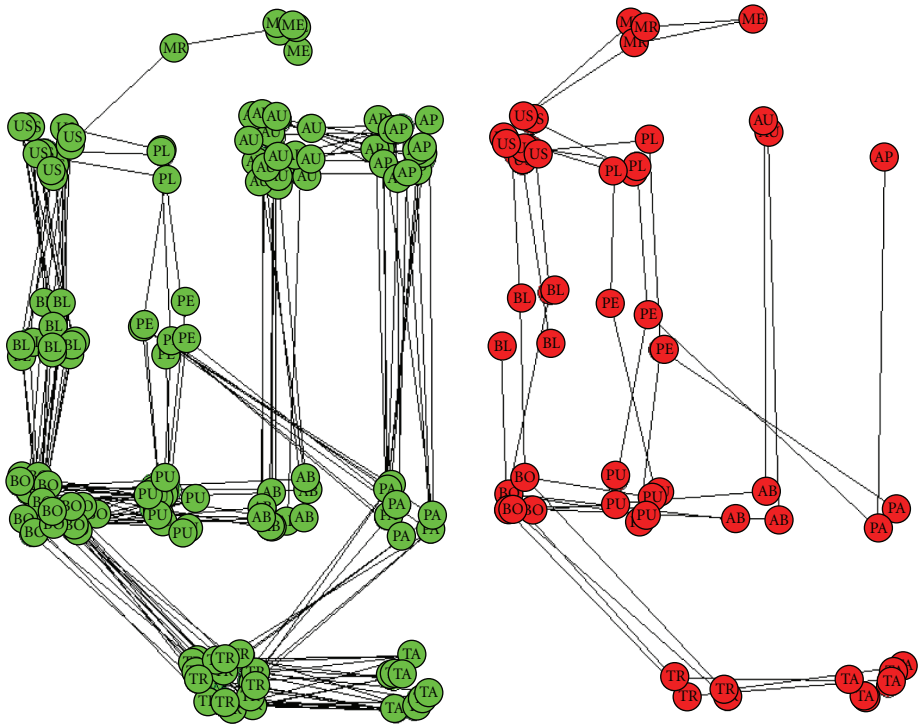
If a program needs to find users who borrowed a periodical containing a paper written by a specific author, the program has to relate each entity, from user to author, before filtering the author with the name passed as the input, as shown in Figure 5. It is important to note that these hypothetical registers are used only in the induction database. The solutions obtained using this technique works straightforwardly on databases without this kind of record since real records are not associated with hypothetical ones, as show in Figure 4. The nodes are records in the database. The acronym of a node is the table of the record, as follows: *user* (US), *author* (AU), *book* (BO), *paper* (PA), *periodical* (PE), *tag* (TA), *message* (ME), *bookLoan* (BL), *periodicalLoan* (PL), *authorBookRel* (AB), *authorPaperRel* (AP), *tagRel* (TR), and *messageRel* (MR). The edges represent associations among records by means of *foreign keys*. At the left are the real records and at the right are the hypothetical ones. As can be seen, there is no link between real records (green) and hypothetical records (red).

7. Conclusion and Future Work

This paper presented linear genetic programming for databases (LGPDB), a tool devoted to induce programs capable of manipulating records stored in a relational database. LGPDB combines a linear genetic programming (LGP) induction environment and a simple database management system (DBMS). Generally, previous works on induction of programs that manipulate databases have focused exclusively on the use of databases as a method to organize and query information, mainly for knowledge discovery and data mining (KDD). This limitation is not a problem when the objective is to find patterns in static scenarios. However, to model processes of information technology (IT) systems, the capability of modifying records is mandatory for the generated programs, since most processes change the state of the entities in the system. The experiments performed and reported in this paper indicate that genetic programming can be used to generate programs for record querying, deletion, insertion, and updating.

The software engineer can select features for which he or she knows the exact outcome a program has to produce, given input information. For each feature of this type, evaluation cases are created. Using these cases and a program induction environment, LGPDB can generate programs to provide the target features. The proposed approach is a first step toward automating relevant and basic stages of IT system development, which may give the opportunity for human programmers to concentrate their efforts on more complex tasks. To pursue this goal, a method to integrate or convert programs in the LGPDB format into standard technologies (programming or database languages) is desired and will be addressed in a future work.

This work provides an initial effort on this issue and additional efforts are required. Section 5 has shown some improvements that have to be explored in more detail. The



(a) (b)

Figure 4: Association among real and hypothetical records. The nodes represent records and the edges the relationships between them by means of *foreign keys*. At the left are the real records and at the right are the hypothetical ones.

generation: 1, standardized fitness: 270.0
//select all users
select(user,rs1)
generation: 15, standardized fitness: 90.0
//select all users associated with a periodical loan
select(periodicalLoan,rs2) select(user,rs1) related(rs1,rs2)
generation: 62, standardized fitness: 75.0
//select all users associated with a periodical loan—disconsider loan without an existent Periodical
select(periodicalLoan,rs2) select(periodical,rs3) related(rs2,rs3) select(user,rs1) related(rs1,rs2)
generation: 148, standardized fitness: 60.0
//select all users associated with a periodical loan—disconsider periodical without paper
select(paper,rs1) select(periodicalLoan,rs2) select(periodical,rs3) related(rs3,rs1) related(rs2,rs3)
select(user,rs1) related(rs1,rs2)
generation: 364, standardized fitness: 45.0
//select all users associated with a periodical loan—disconsider paper without author
select(paper,rs1) select(authorPaperRel,rs2) related(rs1,rs2) select(periodicalLoan,rs2)
select(periodical,rs3) related(rs3,rs1) related(rs2,rs3) select(user,rs1) related(rs1,rs2)
generation: 401, standardized fitness: 0.0
//select all users associated with a periodical loan that has a paper from author X
select(author,rs3) select(paper,rs1) filter(name,equals,x,rs3) select(authorPaperRel,rs2)
related(rs2,rs3) related(rs1,rs2) select(periodicalLoan,rs2) select(periodical,rs3)
related(rs3,rs1) related(rs2,rs3) select(user,rs1) related(rs1,rs2)

Figure 5: The impact in the evolutionary process caused by registers without consistent relationships inserted in the database.

capability to break down a process into multiple simpler processes seems to be promising in the sense of expanding the methodology to deal with interrelated processes. In a future work, we will also consider experiments to evaluate the scalability of LGPDB, for instance, attacking problems with more tables and deeper associations among records in the database. In this context, local search operators may be required to improve the search capability of the evolutionary process.

Acknowledgment

The authors would like to thank CAPES and CNPq for the financial support.

References

[1] C. Larman, *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley Professional, 2004.

[2] G. Wilson and W. Banzhaf, "Fast and effective predictability filters for stock price series using linear genetic programming," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, pp. 1–8, Barcelona, Spain, July 2010.

[3] J. Yu, J. Yu, A. A. Almal et al., "Feature selection and molecular classification of cancer using genetic programming," *Neoplasia*, vol. 9, no. 4, pp. 292–303, 2007.

[4] T. Lensberg, A. Eilifsen, and T. E. McKee, "Bankruptcy theory development and classification via genetic programming," *European Journal of Operational Research*, vol. 169, no. 2, pp. 677–697, 2006.

[5] A. A. Freitas, "A genetic programming framework for two data mining tasks: classification and generalized rule induction," *Genetic Programming*, pp. 96–101, 1997.

[6] T. W. Ryu and C. F. Eick, "MASSON: discovering commonalities in collection of objects using genetic programming," in *Proceedings of the 1st Annual Conference on Genetic Programming*, pp. 200–208, MIT Press, 1996.

[7] T. W. Ryu and C. F. Eick, "Deriving queries from examples using genetic programming," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pp. 303–306, August 1996.

[8] G. A. Archanjo and F. J. von Zuben, "Induction of linear genetic programs for relational database manipulation," in *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI '11)*, pp. 347–352, August 2011.

[9] W. W. Royce, "Managing the development of large software systems," in *Proceedings of the IEEE WESCON*, vol. 26, Los Angeles, Calif, USA, 1970.

[10] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[11] A. Manifesto, "Manifesto for agile software development," *Retrieved November*, 29: 2006, 2001.

[12] M. E. Fayad, R. E. Johnson, and D. C. Schmidt, *Building Application Frameworks: Object-oriented Foundations of Framework Design*, 1999.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley Longman, 1995.

[14] M. Bichier and K. J. Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99–101, 2006.

[15] M. Armbrust, A. Fox, R. Griffith et al., "Above the clouds: a berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Calif, USA, 2009.

[16] S. W. Ambler, *Mapping Objects to Relational Databases: What You Need to Know and Why*, IBM DeveloperWorks, 2000.

[17] S. W. Ambler, *Mapping Objects to Relational Databases: O/R Mapping in Detail*, Ambysoft, 2006.

[18] E. Gamma and K. Beck, Junit, 2005, http://www.junit.org/.

[19] K. Beck, *Test Driven Development: By Example*, Addison-Wesley Professional, 2003.

[20] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[21] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? a comparative evaluation," *Information and Software Technology*, vol. 43, no. 14, pp. 863–873, 2001.

[22] E. B. Boden and G. F. Martino, "Testing software using order-based genetic algorithms," in *Proceedings of the 1st Annual Conference on Genetic Programming*, pp. 461–466, MIT Press, 1996.

[23] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[24] M. Harman, R. Hierons, and M. Proctor, "A new representation and crossover operator for search-based optimization of software modularization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*, pp. 1351–1358, 2002.

[25] T. M. Khoshgoftaar, Y. Liu, and N. Seliya, "A multiobjective module-order model for software quality enhancement," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 6, pp. 593–608, 2004.

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, 1975.

[27] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the 1st International Conference on Genetic Algorithms*, vol. 183, p. 187, 1985.

[28] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[29] S. Luke, "Genetic programming produced competitive soccer softbot teams for robocup97," *Genetic Programming*, pp. 214–222, 1998.

[30] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane, "Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming," in *Proceedings of the 1st Annual Conference on Genetic Programming*, pp. 123–131, MIT Press, 1996.

[31] R. Balzer, "A 15 year perspective on automatic programming," *IEEE Transactions on Software Engineering*, vol. 11, no. 11, pp. 1257–1268, 1985.

[32] L. Xu, H. H. Hoos, and K. Leyton-Brown, "Hydra: automatically configuring algorithms for portfolio-based selection," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010.

[33] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

[34] H. Garcia-Molina and K. Salem, "Main memory database systems: an overview," *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 6, pp. 509–516, 1992.

[35] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, vol. 72, McGraw-Hill, New York, NY, USA, 2002.

[36] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[37] W. Banzhaf, *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications*, Morgan Kaufmann, 1998.

[38] M. Brameier and W. Banzhaf, *Linear Genetic Programming*, Springer, New York, NY, USA, 2007.

[39] P. P. S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.