Optimal Answer Generation by Equivalent Transformation incorporating Multi-Objective Genetic Algorithm

Katsunori Miura (✉ k-miura@res.otaru-uc.ac.jp)

Otaru Shoka Daigaku

Courtney Powell

Hokkaido University: Hokkaido Daigaku

Masaharu Munetomo

Hokkaido University: Hokkaido Daigaku

Research Article

Keywords: Equivalent transformation, Multi-objective genetic algorithm, Clause replacement, Optimization, Pareto-optimal answer, Predicate logic

Posted Date: January 24th, 2022

DOI: <https://doi.org/10.21203/rs.3.rs-1244978/v1>

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
[Read Full License](#)

Version of Record: A version of this preprint was published at Soft Computing on March 17th, 2022. See the published version at <https://doi.org/10.1007/s00500-022-06923-1>.

Optimal Answer Generation by Equivalent Transformation incorporating Multi-Objective Genetic Algorithm

Katsunori Miura^{1*}, Courtney Powell^{2†} and Masaharu Munetomo^{2†}

¹Department of Information and Management Science, Otaru University of Commerce,
3-5-21, Midori, Otaru, 047-8501, Hokkaido, Japan.

²Information Initiative Center, Hokkaido University, Kita 11, Nishi 5, Kita-ku, Sapporo,
060-0811, Hokkaido, Japan.

*Corresponding author(s). E-mail(s): k-miura@res.otaru-uc.ac.jp;

Contributing authors: kotoni@ist.hokudai.ac.jp; munetomo@iic.hokudai.ac.jp;

[†]These authors contributed equally to this work.

Abstract

This paper proposes a framework for time-efficiently finding multiple answers that meet multiple logical constraints and are Pareto-optimal for multiple objective functions. The proposed framework determines a set of optimal answers by iteratively replacing a set of definite clauses. Clause replacement is performed using a SAT solver consisting of equivalent transformation rules (ETRs). An ETR replaces a definite clause with one or more clauses while preserving the declarative meaning of the union of the original clause and problem clauses. To efficiently find optimal answers, in this paper, we define a new class of ETRs that are generated based on the evaluation results of a multi-objective genetic algorithm (MOGA), and propose a method for generating ETRs that belong to the new class. ETRs belonging to the new class help to replace definite clauses according to user objectives such as cost-benefit performance, reliability, and financial constraints. Thus, a SAT solver that uses the new ETR class in addition to extant ETR classes can preferentially replace definite clauses that produce the optimal answer for user objectives. Experimental results indicate that the proposed framework can significantly reduce the computation time and memory usage necessary to determine a set of optimal answers for user objectives.

Keywords: Equivalent transformation, Multi-objective genetic algorithm, Clause replacement, Optimization, Pareto-optimal answer, Predicate logic

1 Introduction

Various practical problems in numerous fields, including business and research, use a *satisfiability problem* (SAT) solver [9, 13, 19, 20] to determine answers that completely satisfy multiple logical constraints. The *equivalent transformation* (ET) framework [1, 3, 5–7] is a program synthesis framework that can generate guaranteed

correct solver programs with respect to specifications. It has been applied to develop software systems such as cloud services brokerage (CSB) systems [10, 17, 25] and learning management systems. In the ET framework, a specification is a set comprising multiple problems, each of which is specified using definite clauses. A program to solve the problems consists of a set of rewriting

rules called *ET rules* (ETRs). An ETR replaces a definite clause with one or more clauses while preserving the declarative meaning of the union of the original clause and problem clauses. (Detailed explanation of the equivalence of the declarative meaning is given in Section 3.3.) Thus, we generate ETRs that can be applied to definite clauses. The ET solver helps develop larger and more complex software systems because programmers can dynamically improve their programs by adding more efficient ETRs. For example, the CSB system developed in [32, 33] comprises approximately four hundred and sixty ETRs and efficiently finds the optimal allocation configuration of cloud services according to the specified system requirements.

In the ET framework, a set of answers with respect to a problem is obtained by iteratively replacing a set of definite clauses using ETRs. The initial state is a singleton consisting of a definite clause representing a query, whereas the final state is a set of unit clauses, each representing an answer. Unit clauses obtained using ETRs represent the correct answer with respect to a problem. If there is no answer, the final state is an empty set. The state under computation to determine the answer consists of both definite clauses and unit clauses. The ET solver can output any unit clause from the current state, even when the computation to find the answer is still running. Therefore, if not all answers for a given problem are needed and the unit clause representing the optimal answer is found, the clause replacement can be terminated at a point determined by the user. Moreover, the ET solver can use the saved clause set to restart the computation and generate the remaining answers. However, the ET framework does not have a mechanism to evaluate the optimality of the answer, so clause replacement is performed until all answers have been generated.

If multiple answers are generated for a given problem, the optimal answer with respect to user objectives such as cost–benefit performance, reliability, and financial constraints should be selected from all the generated answers. In other words, the answer that the user needs should be selected from all answers that meet constraint satisfiability and optimality conditions for user objectives and requirements. However, solver programs built solely with ET cannot determine optimal

answers based on the maximization and/or minimization of values with respect to objective functions, and therefore cannot evaluate the optimality of answers. A *multi-objective genetic algorithm* (MOGA) [22] can identify answers with the best trade-off values (Pareto-optimal answers) for multiple objective functions from answers generated via non-dominated sorting. However, solver programs built solely with MOGA cannot guarantee the correctness of answers with respect to logical constraints. We assert that users can find satisfactory answers effectively by using a set of definite clauses that reflect the MOGA evaluation result when performing clause replacement using ETRs.

Consequently, this paper proposes a problem-solving framework that combines two different techniques, ET and MOGA, to time-efficiently find multiple answers that meet multiple logical constraints and are Pareto-optimal for multiple objective functions. In this paper, this framework is called the *ET with embedded multi-objective optimization* (ETMO) framework. ETMO uses ET to guarantee the satisfiability of each answer for logical constraints, and MOGA to evaluate the optimality of each answer for user objectives. In ETMO, a set of definite clauses that reflect the MOGA evaluation result is generated from the definite clause that appears in the state under computation. A set of reflected clauses is also generated from the definite clause while preserving the declarative meaning of the original clause. To generate the reflected clause, this paper defines a new class of ETRs that are generated based on the MOGA evaluation result and proposes a method for generating ETRs that belong to the new class. By preferentially replacing the reflected clause, a unit clause that represents the optimal answer is efficiently determined. ETMO works efficiently when solving problems in which there are more than hundreds of answers and each answer has a trade-off relationship with other answers with respect to user objectives. Compared to Basic ET, ETMO reduces the computation time and memory usage necessary to determine a set of optimal answers because clause replacement can end when the optimal answers are found during the calculation.

The remainder of this paper is organized as follows. Section 2 gives an overview of relevant frameworks that generate a correct and efficient

SAT solver for a given problem and extant methods that generate ETRs from a definite clause. Section 3 explains the framework for solving problems using ETRs. Section 4 defines the new class of ETRs needed to efficiently find optimal answers, and outlines clause replacement according to ETMO. Section 5 demonstrates the generation of optimal answers using ETMO. Section 6 shows the effectiveness of problem solving using ETMO as compared with Basic ET, and explains future possibilities of ETMO. Section 7 presents concluding remarks.

2 Relevant Studies

In addition to ET, there are other frameworks for generating correct and efficient solver programs for given problems. *Constraint handling rules* (CHR) [14–16, 18] is an extended *logic programming* (LP) [23, 34] framework that generates solver programs by describing procedural knowledge using logical formulas that specify the equivalence between constraints. However, because the CHR solver uses logical formulas to describe procedures, the procedures are described using ground instances only. In contrast, the ET solver describes procedures using non-ground instances. Furthermore, previous studies [29, 31] have shown that the solver program generated by CHR is a subset of ET.

Conflict-driven clause learning (CDCL) [11, 24] is another framework for generating correct and efficient solver programs. The CDCL solver determines the satisfiability of propositional logic formulas described in conjunctive normal form and uses special clauses, called *learned clauses*, to efficiently generate answers that satisfy constraint conditions. Efficient calculation using the CDCL solver results in short computation times. However, the CDCL solver solely uses unfolding and folding rules, which belong to one of the ETR classes, whereas the ET solver can use a variety of other ETR classes. Furthermore, previous studies [1, 2] have shown that the solver program generated by CDCL is a subset of ET. Consequently, this paper proposes an extension framework for ET-based problem solving.

ETRs help develop correct and efficient solver programs for various practical problems. Consequently, several methods [8, 21, 31] for generating

ETRs have been proposed. The methods proposed in [8, 21] use *meta-computation* to generate ETRs. Meta-computation employs special definite clauses with constant domain/type information such as numbers, strings, and lists to generate procedural knowledge for generating ETRs. Another method for generating ETRs with a different approach has also been proposed, called the *LE-based method* [31]. The LE-based method generates ETRs via *logical equivalences* (LEs), each of which describes the equivalence of the declarative meaning between two sets of logical constraints. These methods help automate the generation of various ETRs that belong to different classes from a given definite clause. However, the ETRs generated by these methods cannot replace definite clauses according to user objectives to efficiently find the optimal answer for a given problem. Thus, this paper differs substantially from extant methods because the new class of ETRs provided in this paper focuses on clause replacement to efficiently find the optimal answer.

3 Problem Solving based on Equivalent Transformation

3.1 Problem Description

In the ET framework, a problem is a pair $\langle \mathbb{D}, \{q\} \rangle$, where $\mathbb{D}$ is a set of definite clauses representing the definition of predicates, called *background knowledge*, and $\{q\}$ is a singleton consisting of a definite clause representing a query. Any $d \in \mathbb{D}$ and q are defined according to first-order predicate logic [23, 34]. The aim is to generate a program that solves multiple problems, so a specification is a set of problems. The head part of a definite clause consists of one atomic formula (atom), and the body part consists of one or more atoms. An atom is composed of one predicate p and zero or more terms t . A term t is a given variable or constant. For example, an atom with two terms is written as $p(t_1, t_2)$. A variable is a symbol that begins with an asterisk. We assume that any predicate appearing in the head part of q exists neither in $\mathbb{D}$ nor in the body part of q . Various predicates can be defined in $\mathbb{D}$, and various queries can be made by combining multiple predicates on $\mathbb{D}$. For example, in the realm of predicate logic, the predicate for the concatenation of lists is generally described as

follows:

$$\begin{aligned} d_1 &: \text{Append}([], *x, *x) \leftarrow \\ d_2 &: \text{Append}[*a \mid *x], *y, [*a \mid *z]) \\ &\leftarrow \text{Append}(*x, *y, *z) \end{aligned}$$

These clauses for the *Append* predicate are stored in $\mathbb{D}$. The ground atoms that satisfy this definition are *Append*([A], [B], [A, B]) and *Append*([1], [], [1]), whereas *Append*([A], [B], [A]) and *Append*([1], [], []) do not satisfy this definition. In addition, the reverse order of list elements can be defined by adding the definite clauses d_3 and d_4 to $\mathbb{D}$.

$$\begin{aligned} d_3 &: \text{Reverse}([], []) \leftarrow \\ d_4 &: \text{Reverse}[*x \mid *a], *y) \\ &\leftarrow \text{Reverse}(*a, *b), \text{Append}(*b, [*x], *y) \end{aligned}$$

Following the definition of the *Append* predicate, we can define the following query q_1 representing the concatenation of two lists.

$$q_1 : \text{Answer}(*x, *y) \leftarrow \text{Append}(*x, *y, [A, B])$$

This query means that for [A, B] produced by concatenating the ground instances of $*x$ and $*y$, find the satisfiable ground instances of $*x$ and $*y$ with respect to $\mathbb{D}$.

3.2 Answers

In the ET framework, a set of multiple answers is generated with respect to a problem. Each answer is described by a unit clause, and the number of answers is the same as the number of unit clauses. The head part of a unit clause consists of one atom, and the body part has no atoms. For example, for the problem $\langle \mathbb{D}, \{q_1\} \rangle$, the following three unit clauses are obtained, where $\mathbb{D} = \{d_1, d_2, d_3, d_4\}$.

$$\begin{aligned} a_1 &: \text{Answer}([], [A, B]) \leftarrow \\ a_2 &: \text{Answer}([A], [B]) \leftarrow \\ a_3 &: \text{Answer}([A, B], []) \leftarrow \end{aligned}$$

There is no answer other than the three answers above because the use of the ET solver ensures that all the answers with respect to a problem are

obtained. If no unit clause is obtained, there is no answer with respect to a given problem.

3.3 Computation

In the ET framework, computation iteratively replaces a clause set based on the equivalence of the declarative meaning, so that computation is a sequence $seq = [G_1, G_2, \dots, G_n]$, where $G_1 = \{q\}$. Given a clause set G , the declarative meaning of G , denoted by $\mathcal{M}(G)$, represents all ground atoms determined from G . (The definition of the declarative meaning is explained in [3, 31].) Given two clause sets G_i and G_j and background knowledge $\mathbb{D}$, if $\mathcal{M}(\mathbb{D} \cup G_i) = \mathcal{M}(\mathbb{D} \cup G_j)$ holds, the declarative meaning of G_i is equivalent to that of G_j with respect to $\mathbb{D}$. The first element of seq , that is G_1 , is a singleton consisting of a definite clause, whereas the last element is a unit clause set or an empty set. The other elements of seq consist of both definite clauses and unit clauses. For example, in the computation for solving the problem $\langle \mathbb{D}, \{q_1\} \rangle$ shown in Section 3.1, the first element of seq of the problem is $\{q_1\}$, whereas the last element is $\{a_1, a_2, a_3\}$.

3.4 Programs

In the ET framework, a program to solve a given problem is a set of ETRs. Given a clause set G_i and background knowledge $\mathbb{D}$, an ETR replaces a definite clause $cl(\in G_i)$ with a set of one or more clauses cls , while preserving the declarative meaning of $\{cl\}$ and $\mathcal{G}$ with respect to $\mathbb{D}$; i.e., $\mathcal{M}(\mathbb{D} \cup \{cl\} \cup \mathcal{G}) = \mathcal{M}(\mathbb{D} \cup cls \cup \mathcal{G})$ holds, where $\mathcal{G} = G_i - \{cl\}$. The partial correctness of a program, composed of ETRs, with respect to $\mathbb{D}$ is always guaranteed because each ETR is completely independent and individually correct. When an ETR $(B, C \Rightarrow D.)$ is applied to a definite clause $(\text{Answer} \leftarrow A, B, C)$, a clause set $\{(\text{Answer} \leftarrow A, D)\}$ is obtained. In addition, when two or more arrows appear in an ETR, such as $(B, C \Rightarrow D; \Rightarrow E, F.)$, the same number of clauses as the number of arrows is obtained, such as $\{(\text{Answer} \leftarrow A, D), (\text{Answer} \leftarrow A, E, F)\}$. An ETR applied to a definite clause is generated based on $\mathbb{D}$. (Detailed explanation on the definition of ETRs is given in [3, 31].) The following rule r_1 is one of the ETRs that can be applied to definite clause q_1 shown in Section 3.1. Two clauses

are obtained by applying r_1 because two arrows appear in r_1 .

$$\begin{aligned} r_1 : & \text{Append}(*a, *b, [*c \mid *d]) \\ & \Rightarrow \text{Equal}(*a, []), \text{Equal}(*b, [*c \mid *d]); \\ & \Rightarrow \text{Equal}(*a, [*c \mid *e]), \text{Append}(*e, *b, *d). \end{aligned}$$

By applying r_1 to q_1 , q_1 is replaced with the two clauses q_2 and q_3 .

$$\begin{aligned} q_2 : & \text{Answer}(*x, *y) \\ & \leftarrow \text{Equal}(*x, []), \text{Equal}(*y, [A, B]) \\ q_3 : & \text{Answer}(*x, *y) \\ & \leftarrow \text{Equal}(*x, [A \mid *z]), \text{Append}(*z, *y, [B]) \end{aligned}$$

The definite clause q_3 can be replaced using r_1 , whereas a new rule is required to replace the definite clause q_2 .

4 Clause Replacement Framework using ETMO

4.1 Definition of OpETRs

A rule for clause optimization (OpETR) belongs to a new class of special ETRs provided in this paper and is generated based on the MOGA evaluation result and background knowledge $\mathbb{D}$, which replaces a definite clause with one or more controlled definite clauses. An OpETR describes a procedure that controls the specialization of variables appearing in the head atom of a definite clause such as an *Answer* atom. The body part of the controlled definite clause has control atoms to efficiently find the optimal answer for user objectives. Specialization control procedures are determined based on the MOGA evaluation results. Each OpETR is composed of a head part $\mathcal{X}$ and a body part $\mathcal{Y}_n$, where $n \geq 1$. $\mathcal{X}$ consists of one atom, and $\mathcal{Y}_n$ consists of one or more atoms. If the number of body parts is one, the OpETR is specified by $(\mathcal{X} \Rightarrow \mathcal{Y}_1)$ using $\mathcal{X}$ and $\mathcal{Y}_n$. If there are two or more body parts, the body parts are specified separated by semicolons. The head part $\mathcal{X}$ of the OpETR is an *Optimize* atom composed of terms that appear in an *Answer* atom. If two terms $[A \mid *x]$ and $*y$ appear in an *Answer* atom, the atom is written as *Optimize* $([A \mid *x], *y)$. The elements of $\mathcal{Y}_n$ are an *OpPattern* atom and a *NotOpPattern* atom. These atoms have two

Fig. 1 Differences in clause replacement between ETMO and Basic ET

terms; the first term is selected from terms that appear in an *Optimize* atom, and the second term is an instance obtained by generalizing optimal ground instances determined based on the MOGA evaluation result.

As shown in Fig. 1, OpETRs are used in conjunction with normal ETRs when using ETMO to replace clause sets. When using Basic ET to replace clause sets, normal ETRs only are used. A normal ETR is simply called an ETR in the rest of this paper. An OpETR can be applied to a definite clause at the timing specified by the user. In this paper, we use an OpETR to replace definite clause q as an element of the initial state of a clause set. All unit clauses obtained from an initial state $\{q\}$ using ETMO completely meet constraint satisfiability with respect to background knowledge $\mathbb{D}$. Furthermore, the final state of a clause set determined using ETMO is the same as using Basic ET.

4.2 Clause Replacement Workflow using ETMO

ETMO is primarily utilized to generate a clause set comprising unit clauses that represent the answer evaluated as optimal using MOGA in the middle of the calculation. With ETMO, optimal answers are obtained from the current clause set even if the computation to generate the answer has not been completed. Clause replacement using ETMO proceeds as follows:

Phase 1: Generate Pareto-optimal answers using MOGA

In MOGA [22], a search is performed to find individuals with the best trade-off values for multiple objective functions. The output of the objective function represents the fitness value of an individual with respect to evaluation metrics such as cost–benefit performance, reliability, and financial constraints. In this paper, we use *non-dominated sorting genetic algorithm II* (NSGA-II) [12], in which individuals with the best trade-off values are determined via non-dominated sorting and crowding distance. The operations performed in this phase are as follows:

- (1) Generate individuals based on the problem $\langle \mathbb{D}, \{q\} \rangle$;
- (2) Determine an initial population composed of the individuals;
- (3) Generate Pareto-optimal answers from the population.

In ETMO, each individual can be represented by a unit clause, because individuals represent ground instances of an *Answer* atom. However, the constraint satisfiability of the unit clauses obtained by MOGA computation is not guaranteed. This is because MOGA finds all optimal answers, but cannot determine their satisfiability with respect to $\mathbb{D}$. Step (1) generates various ground substitutions for the variables appearing in an *Answer* atom based on the problem $\langle \mathbb{D}, \{q\} \rangle$. This step uses the problem $\langle \mathbb{D}, \{q\} \rangle$ to refer to constant domains and constant types, such as numbers, strings, and lists, which apply to variables. Therefore, an individual that represents an unsatisfiable atom with respect to $\mathbb{D}$ could be generated. For example, when generating an individual based on the problem $\langle \mathbb{D} = \{d_1, d_2, d_3, d_4\}, \{q_1\} \rangle$ shown in Section 3.1, an individual representing atoms such as the

following is obtained.

$$\begin{aligned} i_1 &: \text{Answer}([A], [B]) \\ i_2 &: \text{Answer}([A], [C]) \\ i_3 &: \text{Answer}([], [A, B]) \\ i_4 &: \text{Answer}([A, B], [D]) \\ i_5 &: \text{Answer}([C], [B, D]) \\ i_6 &: \text{Answer}([A, B], []) \end{aligned}$$

The individuals i_1 , i_3 , and i_6 are the satisfiable atoms, whereas the individuals i_2 , i_4 , and i_5 are the unsatisfiable atoms. In ETMO, Step (2) determines an initial population for MOGA computation with a size of one hundred individuals. If more than one hundred individuals are generated in Step (1), one hundred individuals are randomly selected in Step (2). In Step (3), the optimal population is generated from the initial population over multiple generations. The objective functions used in MOGA computation are defined according to user objectives such as cost–benefit performance, reliability, and financial constraints. Therefore, the objective function changes depending on the given problem and the user objective.

Phase 2: Replace clause sets using ETRs and OpETRs

In ETMO, a definite clause is replaced with a set of one or more clauses by a one-time application of an ETR or an OpETR. If a unit clause set or an empty set is obtained, the clause replacement is terminated, as it would mean there is no clause applicable to ETRs or OpETR. In addition, if a unit clause representing the optimal answer is obtained, the clause replacement can be terminated in the timing specified by the user. Various methods for generating ETRs have been proposed [4, 21, 30], and it is assumed that the ETRs used in ETMO are generated according to these methods. Thus, only the OpETR generation method is explained in this paper, as it has not been discussed previously. Given a problem $\langle \mathbb{D}, \{q\} \rangle$, the initial state of a clause set in this phase is $\{q\}$, which is named $\mathbb{Q}$. The operations performed in this phase are as follows:

- (1) Select one definite clause $cl(\in \mathbb{Q})$ to apply the rule;
- (2) Replace definite clause cl using ETR or OpETR;
- (3) Generate the next state of clause set $\mathbb{Q}$;

- (4) Evaluate the optimality of the unit clauses obtained in (3).

ETMO ranks each definite clause to efficiently generate unit clauses representing the optimal answers. In Step (1), the high-priority definite clauses are preferentially selected from clause set $\mathbb{Q}$. If all the definite clauses have the same rank, definite clause cl is randomly selected. In Step (2), definite clause cl is replaced using an ETR, but an OpETR is used if control is needed for the specialization of variables that appear in the head atom of cl . In addition, in this paper, definite clause cl selected from clause set $\mathbb{Q}$ of the initial state is replaced using an OpETR. An ETR is applicable to a definite clause cl , whereas an OpETR is applicable to a definite clause cl' automatically generated by a simple transformation of the formula from cl . Given a definite clause $cl = (H \leftarrow B_1, \dots, B_k)$, a definite clause cl' generated from cl is as follows:

$$H \leftarrow Optimize(v_1, \dots, v_i), B_1, \dots, B_k,$$

where H and B_k are atoms, v_i is a variable that appears in H , and the *Optimize* atom consists of all variables that appear in H . The definite clause cl' is generated from definite clause cl while preserving the declarative meaning of cl with respect to background knowledge $\mathbb{D}$; i.e., $\mathcal{M}(\mathbb{D} \cup \{cl\}) = \mathcal{M}(\mathbb{D} \cup \{cl'\})$ holds. Step (3) combines clause set Q and the clause set obtained in (2) to generate the next state of clause set $\mathbb{Q}$, where $Q = \mathbb{Q} - \{cl\}$. If the clause replacement continues, Step (1) selects a definite clause cl from the new clause set $\mathbb{Q}$ generated in Step (2). Step (4) evaluates the optimality of the unit clause(s) obtained by the processing of Step (2). In addition, Step (4) uses the Pareto-optimal answer generated in Phase 1 to specify the optimality of the unit clause. If there is no optimal answer for the unit clauses obtained in Step (2), the clause replacement is restarted from Step (1). Conversely, if there is an optimal answer, it is possible to abort the clause replacement. The clause replacement can be resumed using the current state of clause set $\mathbb{Q}$ even if the clause replacement is aborted during the computation.

4.3 OpETRs Generation Workflow

As shown in Section 4.1, an OpETR replaces an *Optimize* atom in the body part of definite clause

cl' with multiple *OpPattern* or *NotOpPattern* atoms. An OpETR is generated according to the following steps:

- (1) Classify individuals in the Pareto-optimal answer according to the features of the answer;
- (2) Select the class that has the most elements;
- (3) Make a general instance that applies to variables of the *Optimize* atom according to the features of the class;
- (4) Generate *OpPattern* and *NotOpPattern* atoms using the general instance and variables of the *Optimize* atom;
- (5) Generate an OpETR using the *Optimize*, *OpPattern*, and *NotOpPattern* atoms.

The usefulness of an OpETR depends on a general instance applied to the second term of *OpPattern* and *NotOpPattern* atoms. To determine an effective general instance, it is important to perform appropriate classification of optimal ground instances obtained by MOGA computation. In this paper, the classification of the optimal ground instances is performed manually, because the classification policy changes depending on the given problem. For example, it is assumed that a general instance $answer([*a \mid *b], [*c \mid *d])$ is given for the problem $\langle \mathbb{D} = \{d_1, d_2, d_3, d_4\}, \{q_1\} \rangle$, shown in Section 3.1, and an OpETR is applied to the following definite clause cl' :

$$cl' : Answer(*x, *y) \\ \leftarrow Optimize(*x, *y), Append(*x, *y, [A, B])$$

From $answer([*a \mid *b], [*c \mid *d])$, two *OpPattern* atoms and two *NotOpPattern* atoms are generated, because $*x$ is substituted with $[*a \mid *b]$ and $*y$ is substituted with $[*c \mid *d]$.

$$OpPattern(*x, [*a \mid *b]) \\ OpPattern(*y, [*c \mid *d]) \\ NotOpPattern(*x, [*a \mid *b]) \\ NotOpPattern(*y, [*c \mid *d])$$

As a result, an OpETR composed of these atoms is generated as follows:

$$Optimize(*x, *y) \\ \Rightarrow OpPattern(*x, [*a \mid *b]), \\ OpPattern(*y, [*c \mid *d]);$$

$$\Rightarrow \text{NotOpPattern}(*x, [*a \mid *b]),$$

$$\text{NotOpPattern}(*y, [*c \mid *d]).$$

As this OpETR consists of two body parts, the definite clause cl' is replaced with two definite clauses. A definite clause with an *OpPattern* atom has a higher rank than a definite clause with a *NotOpPattern* atom.

5 Experimental Evaluation

5.1 Problem Settings

This section uses the optimal allocation problem of cloud services treated in [33]. The objective is to determine the optimal number of virtual machines (VMs) and the computational performance required to deploy a genome analysis workflow [35]. In many cases, a genome analysis workflow is a sequential processing of multiple analysis tools. Details on how to determine the appropriate cloud services for deploying a genome analysis workflow is given in [33]. The method gives five predicates for specifying user requirements and two objective functions to determine the optimal answer. The *Equal* atom is a built-in atom of the ET solver. A query in the problem is generated from a logical formula composed of atoms generated by the five predicates. As there are numerous cloud providers, instance types, and legal/policy requirements, we can determine many different allocation patterns of cloud services for a given query. Fig. 2 depicts one hundred and four allocation patterns of cloud services to deploy a genome analysis workflow composed of three analysis tools. The optimal answer is determined with the best trade-off between analysis time and infrastructure usage fees where these values are estimates. The method proposed in [33] aims to determine optimal computation infrastructures displayed in the dashed ellipse section shown in Fig. 2. This experiment demonstrates that ETMO can preferentially find the optimal computation infrastructure shown in Fig. 2. For the problem that produces the answers shown in Fig. 2, the query is as follows:

dcl_1 :

$$\text{Answer}(*structure) \leftarrow$$

$$\text{Equal}(*toolA, [L1, T1, [0.9]]),$$

$$\text{Equal}(*toolB, [L2, T2, [0.9]]),$$

$$\text{Equal}(*toolC, [L3, T3, [0.9]]),$$

$$\text{Workflow}(*structure, [*toolA, *toolB, *toolC]),$$

$$\text{Budget}(*structure, [L1, L2, L3], 7.0),$$

$$\text{Time}(*structure, [L1, L2, L3], 2, h),$$

$$\text{Location}(*structure, [L1], [\text{Tokyo}]),$$

$$\text{Location}(*structure, [L2, L3], [\text{Seoul}, \text{Singapore}]),$$

$$\text{Policy}(*structure, [L1, L2, L3], \text{public}, \text{single})$$

In the genome analysis workflow of this problem, tool T1 is executed first, then tool T2, and finally tool T3. The variable **structure* is given a computation infrastructure that satisfies nine constraints that appear in the body part of dcl_1 . (Detailed explanation of the atoms appearing in this definite clause is omitted in this paper.) In this experiment, MOGA evaluation results are used to control the specialization of **structure* according to the appropriate number of VMs. The initial state of clause set $\mathbb{Q}$ in Phase 2 shown in Section 4.2 is a singleton consisting of the above definite clause. Moreover, in this experiment, we terminate the clause replacement when a unit clause representing the optimal answer is generated.

5.2 Experimental Results

In Phase 1, the Pareto-optimal answer set is generated from the initial population composed of one hundred individuals over ten generations, and consists of seventy eight individuals. The answers that satisfy the constraints described by the definite clause dcl_1 do not appear in the Pareto-optimal answer set. However, the satisfiable answers could be generated depending on the number and combination of constraint conditions. Now, we focus on the number of VMs used to deploy the genome analysis workflow shown in each answer. Given the definite clause dcl_1 , the maximum number of VMs is three. The VM deployment pattern shown in the Pareto-optimal answers is as follows:

- (P1) All tools are executed on one VM; i.e., one VM is used. (There are three answers.)
- (P2) The workflow is divided into first half and second half; i.e., two VMs are used. (There are thirty nine answers.)
- (P3) VMs are deployed per analysis tool; i.e., three VMs are used. (There are thirty six answers.)

Fig. 2 Universal set of answers obtained for a given problem

In the above (P2), there are cases where “analysis tools T1 and T2 are executed on the same VM” and “analysis tools T2 and T3 are executed on the same VM.” In this experiment, we will treat these cases as one deployment pattern.

In Phase 2, as the initial clause set $\mathbb{Q}$ is $\{dcl_1\}$, the definite clause cl selected in Step (1) is the definite clause dcl_1 . In addition, the definite clause dcl_1 is replaced using an OpETR generated based on the VM deployment pattern determined by MOGA computation. The *Optimize* atom that appears in the definite clause cl' generated from the definite clause dcl_1 is *Optimize(*structure)*. Therefore, we generate an OpETR that controls the specialization of **structure*.

From the MOGA results above, it is recommended to substitute **structure* with the deployment pattern (P2) or (P3) because the number of individuals belonging to (P2) is almost equal to the number of individuals belonging to (P3). When the deployment pattern (P2) is selected, the *OpPattern* and *NotOpPattern* atoms will be generated as follows:

OpPattern(*structure, [*vm1, *vm2])
NotOpPattern(*structure, [*vm1, *vm2])

Therefore, the OpETR generated using these atoms is written as follows:

*opr*₁ : *Optimize*(*structure)
 $\Rightarrow$ *OpPattern*(*structure, [*vm1, *vm2]);
 $\Rightarrow$ *NotOpPattern*(*structure, [*vm1, *vm2]).

The definite clause cl' is replaced with two definite clauses by applying the rule *opr*₁. One of the clauses is a definite clause with an *OpPattern* atom. Let dcl_2 be this clause. The other clause is a definite clause with a *NotOpPattern* atom. Let dcl_3 be this clause. The definite clause dcl_2 has a higher rank than the definite clause dcl_3 because dcl_2 includes an *OpPattern* atom. As a result of applying the rule, the next state of clause set $\mathbb{Q}$ is $\{dcl_2, dcl_3\}$. The clause replacement will be restarted from Step (1) because there is no optimal answer in clause set $\mathbb{Q}$. In this experiment, ETRs are used to preferentially replace dcl_2 and definite clauses obtained from dcl_2 . After replacing clause set $\mathbb{Q}$ a total of 8,568 times using ETRs, the following unit clause is generated:

*ucl*₁ : *answer*([
 [aws, c5.9xlarge, ap-northeast-1, [T1]],
 [aws, c5.4xlarge, ap-northeast-2, [T2, T3]]
]) $\leftarrow$

Unit clause *ucl*₁ represents the optimal answer and is plotted in the dashed ellipse section shown in Fig. 2. Unit clauses other than the definite clause *ucl*₁ are not obtained before generating *ucl*₁. The clause replacement ends because a unit clause representing the optimal answer is generated. However, if the user needs a different answer, clause set $\mathbb{Q}$ can be replaced to find other unit clauses.

If the deployment pattern (P3) is selected to generate an OpETR, the *OpPattern* and

NotOpPattern atoms will be generated as follows:

```
OpPattern(*structure, [*vm1,*vm2,*vm3])
NotOpPattern(*structure, [*vm1,*vm2,*vm3])
```

Therefore, the OpETR generated using these atoms is written as follows:

```
opr2 : Optimize(*structure)
      ⇒ OpPattern(*structure,
                  [*vm1,*vm2,*vm3]);
      ⇒ NotOpPattern(*structure,
                  [*vm1,*vm2,*vm3]).
```

The definite clause cl' is replaced with two definite clauses, similar to the replacement of the definite clause cl' using an OpETR opr_1 . Let dcl_4 be a definite clause with an *OpPattern* atom and dcl_5 be a definite clause with a *NotOpPattern* atom. Therefore, the state of clause set $\mathbb{Q}$ changes from $\{dcl_1\}$ to $\{dcl_4, dcl_5\}$. After replacing clause set $\mathbb{Q}$ a total of 19,104 times using ETRs, the following unit clause is generated:

```
ucl2 : answer([
    [aws, c5.9xlarge, ap-northeast-1, [T1]],
    [aws, c5.9xlarge, ap-northeast-2, [T2]],
    [aws, m5.xlarge, ap-northeast-2, [T3]]
]) ←
```

The definite clause dcl_4 and definite clauses obtained from dcl_4 are preferentially selected here, because dcl_4 has a higher rank than dcl_5 . The unit clause ucl_2 represents the optimal answer and is plotted in the dashed ellipse section in Fig. 2. The clause replacement ends because a unit clause representing the optimal answer is generated.

Three unit clauses other than the definite clause ucl_2 are obtained before generating ucl_2 . However, the three unit clauses are not optimal answers. For example, one of the three definite clauses is as follows:

```
ucl3 : answer([
    [aws, c5.9xlarge, ap-northeast-1, [T1]],
    [aws, m5.12xlarge, ap-northeast-2, [T2]],
    [aws, m5.large, ap-northeast-2, [T3]]
]) ←
```

The definite clause ucl_3 is not an optimal answer, and is plotted outside the dashed ellipse section in Fig. 2. In Phase 2, the clause replacement continues automatically if the optimal answer is not available. Therefore, the clause replacement continues after generating ucl_3 . As the optimality of the answers is determined by the objective functions, the definite clause ucl_3 could be an optimal answer when using different objective functions in MOGA computations.

This experiment showed that ETMO can efficiently find an optimal answer using an OpETR and ETRs. However, we need to generate an OpETR that is not misleading to find a satisfiable answer because MOGA does not guarantee the satisfiability of answers with respect to background knowledge $\mathbb{D}$. As a future challenge, we will propose a method to effectively utilize the results of MOGA computation.

6 Effectiveness and Applicability of ETMO

6.1 Comparison of ETMO with Basic ET

The advantage of ETMO over Basic ET is that the results of MOGA computations can be used to generate optimal answers. This advantage allows ETMO to use the OpETR to preferentially generate optimal answers and terminate the clause replacement when an optimal answer is generated. Many methods [4, 21, 26–28, 30] for generating ETRs have been proposed for use in Basic ET. These are methods for generating normal ETRs, not OpETRs. A normal ETR specializes in variables in order to efficiently generate all answers, and replaces each atom with another atom to simplify the constraint conditions. For example, the method proposed in [30] produces the following ETR, called a *functionality-based rule*:

```
Append(*x,*y,*a), Append(*x,*y,*b)
⇒ Equal(*a,*b), Append(*x,*y,*a),
   Append(*x,*y,*b).
```

Functionality-based rules unify two variables included in two atoms owing to the constraint of a predicate between the two atoms that satisfy the specified conditions. Functionality-based rules

Table 1 Difference in computation cost between Basic ET and ETMO

	Number of replacements (Time $\pm$ Standard variation (msec))	Maximum size of $\mathbb{Q}$ (Data size (byte))
Basic ET	863,116 (4,475,673 $\pm$ 36,095)	136 (55,354)
ETMO (Select P2)	8,569 (18,590 $\pm$ 273)	33 (31,606)
ETMO (Select P3)	19,105 (91,067 $\pm$ 442)	51 (52,641)

are used to facilitate the specialization of variables appearing in a definite clause and prevent infinite clause replacement loops. Meanwhile, the method proposed in [28] produces the following ETR, called a *false rule*:

$$\text{Append}(*x, [*y \mid *z], *z) \Rightarrow \text{false}.$$

False rules remove any definite clause that contains atoms that do not satisfy the constraints of predicates. False rules are used to reduce the fruitless replacement of definite clauses that do not generate unit clauses. That is, a normal ETR cannot control the clause replacement in order to efficiently generate the unit clause representing the optimal answer, as in OpETRs. Moreover, Basic ET cannot evaluate the optimality of the unit clause for user objectives because Basic ET cannot search for answers by maximizing or minimizing the values of user objectives. Because Basic ET always needs to generate all the answers, the clause replacement cannot end in the middle of the calculation even if the optimal answers are generated. ETMO evaluates the optimality of each answer in the middle of clause replacement.

We will now use the example shown in Section 5 to demonstrate that the calculation cost of ETMO is lower than that of Basic ET. The efficiency of calculations in the ET solver depends on the number of clause replacements and the maximum size of clause set $\mathbb{Q}$ until the user finds the answer that satisfies the constraint conditions. The values in the Time column shown in Table 1 are the average times when calculating the optimal answer generation ten times; plus/minus values are standard variations. As shown in Table 1, when the deployment pattern P2 is selected, the number of ETMO-based clause replacements is 8,569. In addition, when the deployment pattern P3 is selected, the number of ETMO-based clause replacements is 19,105. In contrast, the number of Basic ET-based clause replacements is 863,116. That is, the number of ETMO-based

clause replacements is significantly less than that of Basic ET-based clause replacements by a factor of forty five. Moreover, when the deployment pattern P2 is selected, the maximum size of clause set $\mathbb{Q}$ in ETMO is 33. In addition, when the deployment pattern P3 is selected, the maximum size of clause set $\mathbb{Q}$ in ETMO is 51. In contrast, the maximum size of clause set $\mathbb{Q}$ in Basic ET is 136. That is, the maximum size of clause set $\mathbb{Q}$ of ETMO is smaller than that of Basic ET by a factor of three. The maximum size of clause set $\mathbb{Q}$ increases with the number of satisfiable answers and ground substitutions with respect to background knowledge $\mathbb{D}$. As shown in Fig. 3, Basic ET increases memory usage because the clause replacement cannot end in the middle of the calculation, even if the optimal answer for user objectives is found. In contrast, ETMO will end the clause replacement at the timing specified by the user when a satisfactory answer is found.

The time complexity of ETMO is determined by the number of clause replacements using ETRs, which depends on the constraints specified in the query. For example, if *Location*(*structure, [L1], [Tokyo]) appearing in *dcl*₁ shown in Section 5.1 is changed to *Location*(*structure, [L1], [Tokyo, Seoul]), the number of atoms appearing in *dcl*₁ does not change, but it is expected that the number of clause replacements will increase. This is because we need to determine whether there is an answer that meets the added location, that is, Seoul. Therefore, the time complexity is highly dependent on the number of atoms and the described content of each atom. Furthermore, it is not possible to predict whether the number of clause replacements will increase or decrease.

6.2 Application to ETR generation method

In developing ETMO, we collected the data used for the example application within the

Fig. 3 Difference in the size transition of clause set $\mathbb{Q}$ between Basic ET and ETMO

CREST project. To apply ETMO to another actual problem, we would need to collect problem and domain-specific data to tune it. However, as another application example, we believe that ETMO can be used to propose new ETR generation methods. For example, given the following query cl_a , the answer cl_b to the query is obtained using ETRs. The *Palindrome* predicate specifies that a given list is a palindrome. Therefore, the query cl_a means that for $[*x, 2, 3, *y, 1]$ that is a palindrome, find satisfiable ground instances of the variables $*x$ and $*y$ with respect to background knowledge $\mathbb{D}$.

$$\begin{aligned}
 cl_a &: \text{Answer}(*x, *y) \\
 &\leftarrow \text{Palindrome}([*x, 2, 3, *y, 1]) \\
 &\downarrow \text{Replace using ETRs} \\
 cl_b &: \text{Answer}(1, 2) \leftarrow
 \end{aligned}$$

The program that solves the problem $\langle \mathbb{D}, \{cl_a\} \rangle$ consists of five ETRs; the two ETRs, r_a and r_b , presented below are components of the program. In this section, the three other ETRs are omitted.

$$\begin{aligned}
 r_a &: \text{Palindrome}([*a \mid *b]) \\
 &\Rightarrow \text{Reverse}(*b, [*a \mid *c]), \text{Palindrome}(*c). \\
 r_b &: \text{Reverse}([*a \mid *b], *c) \\
 &\Rightarrow \text{Reverse}(*b, *d), \text{Append}(*d, [*a], *c).
 \end{aligned}$$

The ET solver will generate cl_b from cl_a by applying ETR 168 times. On the other hand, the following ETR, r_c , can be generated by combining

r_a and r_b :

$$\begin{aligned}
 r_c &: \text{Palindrome}([*a \mid *b]) \\
 &\Rightarrow \text{Append}(*c, [*a], *b), \text{Palindrome}(*c).
 \end{aligned}$$

If the program uses r_c instead of r_a and r_b , the ET solver will generate cl_b from cl_a by applying ETR fourteen times. Combining current ETRs in this way can facilitate the generation of new ETRs that help to reduce the number of clause replacements. However, finding the appropriate combination requires high-level experience in ET programming. We believe that ETMO can be used to find the appropriate pattern from various combinations of atoms that appear in current ETRs.

7 Conclusions

In this paper, we proposed a problem-solving framework that combines two different techniques, ET and MOGA, to time-efficiently find multiple answers that meet multiple logical constraints and are Pareto-optimal for multiple objective functions. Because optimal answers are generated by replacing the set of definite clauses that reflect the results of MOGA computation, based on ET, any optimal answer satisfies the constraint satisfiability. The advantage of ETMO is that it uses a special ETR called an OpETR that controls the variable specialization for efficiently generating satisfactory answers by the user. As the OpETR concept does not appear in Basic ET, this paper extends the ET-based problem solving framework.

Furthermore, this paper explained the effectiveness of ETMO based on a comparison with Basic ET.

However, we consider that there is a need for an advanced mechanism for utilizing the results of MOGA computation when generating an OpETR. Therefore, as future work, we are developing a method to effectively utilize the results of MOGA computation.

Compliance with Ethical Standards

Funding: This study was funded by CREST, Japan Science and Technology Agency (grant number JPMJCR1501).

Conflict of interest: The authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

- [1] K. Akama, N. Ekawit: Correctness of solving query-answering problems using satisfiability solvers, *Lecture Notes in Computer Science*, 7802 (2013) 404-413.
- [2] K. Akama, N. Ekawit: An ET-based low-level solution for query-answering problems, *Lecture Notes in Computer Science*, 8271 (2013) 1-12.
- [3] K. Akama, N. Ekawit: Equivalent transformation in an extended space for solving query-answering problems, *Lecture Notes in Computer Science*, 8397 (2014) 232-241.
- [4] K. Akama, H. Koike, H. Mabuchi: A theoretical foundation of program synthesis by equivalent transformation, *Perspectives of System Informatics*, 2244 (2001) 131-139.
- [5] K. Akama, H. Koike, E. Miyamoto: A theoretical foundation for generation of equivalent transformation rules (program transformation, symbolic computation and algebraic manipulation), *Research Institute for Mathematical Sciences Kyoto University Koukyuroku*, (1125) (2000) 44-58.
- [6] K. Akama, E. Nantajeewarawat: Unfolding existentially quantified sets of extended clauses, *The 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2 (2016) 96-103.
- [7] K. Akama, E. Nantajeewarawat: Skolemization that preserves logical meanings, *International Journal of Innovative Computing, Information & Control*, 17(1) (2021) 1-13.
- [8] K. Akama, E. Nantajeewarawat: Computing logically with generation of equivalent transformation rules, *International Journal of Innovative Computing, Information & Control*, 18(1) (2022) 315-330.
- [9] C. Barrett, C. Tinelli: *Satisfiability Modulo Theories*, Springer International Publishing, (2018) 305-343.
- [10] S. Bayless, N. Kodirov, S. Iqbal, I. Beschastnikh, H. Hoos, A. Hu: Scalable constraint-based virtual data center allocation, *Artificial Intelligence*, 278 (2020).
- [11] M. Davis, G. Logemann, D. Loveland: A machine program for theorem-proving, *Communications of the ACM*, 5(7) (1962).
- [12] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan: A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, 6(2) (2002) 182-197.
- [13] N. Eén, N. Sörensson: An extensible SAT-solver, *Theory and Applications of Satisfiability Testing (SAT2003)*, *Lecture Notes in Computer Science*, 2919 (2004) 502-518.
- [14] T. Frühwirth: Theory and practice of constraint handling rules, *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3) (1998) 95-138.

- [15] T. Frühwirth: A devil's advocate against termination of direct recursion, The 17th International Symposium on Principles and Practice of Declarative Programming, (2015).
- [16] T. Frühwirth: Constraint handling rules - what else?, CoRR, abs/1701.02668 (2017).
- [17] Gartner, Inc.: Definition of Cloud Services Brokerage (CSB) - IT Glossary, <https://www.gartner.com/en/information-technology/glossary/cloud-services-brokerage-csb>, (2021).
- [18] C. Holzbaur, T. Frühwirth: Compiling constraint handling rules into prolog with attributed variables, Lecture Notes in Computer Science, 1702 (1999) 117-133.
- [19] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, K. Leyton-Brown: The configurable SAT solver challenge (CSSC), Artificial Intelligence, 243 (2017) 1-25.
- [20] M. Järvisalo, D. Berre, O. Roussel, L. Simon: The international SAT solver competitions, AI Magazine, 33(1) (2012) 89-92.
- [21] H. Koike, K. Akama, H. Mabuchi, Y. Shigeta: Rule generation by meta-computation, The 6th International Conference on Information Systems Analysis and Synthesis, (2000) 429-434.
- [22] A. Konak, D. W. Coit, A. E. Smith: Multi-objective optimization using genetic algorithms: A tutorial, Reliability Engineering & System Safety, 91(9) (2006) 992-1007.
- [23] J. W. Lloyd: Foundations of Logic Programming 2nd Edition, Springer-Verlag, (1993).
- [24] J. Marques-Silva, K. Sakallah: GRASP: A search algorithm for propositional satisfiability, IEEE Transactions on computers, 48(5) (1999) 506-521.
- [25] F. Micota, M. Erascu, D. Zaharie: Constraint satisfaction approaches in cloud resource selection for component based applications, The 2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing, (2018).
- [26] K. Miura, K. Akama: ET-based bidirectional search for proving formulas in the class ES, International Journal of Innovative Computing, Information & Control, 10(6) (2014) 1999-2009.
- [27] K. Miura, K. Akama: Generation of logical equivalences belonging to the C2LE class applied to program synthesis based on equivalent transformation, International Journal of Innovative Computing, Information & Control, 17(4) (2021) 1119-1135.
- [28] K. Miura, K. Akama, H. Koike, H. Mabuchi: Proof of unsatisfiability of atom sets based on computation by equivalent transformation rules, International Journal of Innovative Computing, Information & Control, 9(11) (2013) 4419-4430.
- [29] K. Miura, K. Akama, H. Mabuchi: Creation of ET rules from logical formulas representing equivalent relations, International Journal of Innovative Computing, Information & Control, 5(2) (2009) 263-277.
- [30] K. Miura, K. Akama, H. Mabuchi: Generating functionality-based rules for program construction, International Journal of Innovative Computing, Information & Control, 5(9) (2009) 2463-2479.
- [31] K. Miura, K. Akama, H. Mabuchi, H. Koike: Theoretical basis for making equivalent transformation rules from logical equivalences for program synthesis, International Journal of Innovative Computing, Information & Control, 9(6) (2013) 2635-2650.
- [32] K. Miura, M. Munetomo: A predicate logic-defined specification method for systems deployed by intercloud brokerages, The 2016 IEEE International Conference on Cloud Engineering Workshops (IC2EW), (2016) 172-177.
- [33] K. Miura, C. Powell, M. Munetomo: Optimal and feasible cloud resource configurations generation method for genomic analytics applications, The 2018 IEEE International Conference on Cloud Computing Technology and

Science (CloudCom), (2018) 137-144.

[34] L. Sterling, E. Shapiro: The Art of Prolog 2nd Edition, The MIT Press (1994).

[35] C. Trapnell, A. Robert, L. Goff, G. Pertea, D. Kim, D. Kelley, H. Pimentel, S. Salzberg, J. Rinn, L. Pachter: Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks, Nat. Protocols, 7(3) (2012) 562-578.