

A Novel Automatic Relational Database Normalization Method

Emre Akadal ¹ , Mehmet Hakan Satman ² 

¹ Department of Management Information Systems, Faculty of Economics, Istanbul University, Beyazıt Kampüsü, 34116 Fatih/İstanbul, Turkey

² Department of Econometrics, Faculty of Economics, Istanbul University, Beyazıt Kampüsü, 34116 Fatih/İstanbul, Turkey

Corresponding author: Emre Akadal (emre.akadal@istanbul.edu.tr)

Abstract

The increase in data diversity and the fact that database design is a difficult process make it practically impossible to design a unique database schema for all datasets encountered. In this paper, we introduce a fully automatic genetic algorithm-based relational database normalization method for revealing the right database schema using a raw dataset and without the need for any prior knowledge. For measuring the performance of the algorithm, we perform a simulation study using 250 datasets produced using 50 well-known databases. A total of 2500 simulations are carried out, ten times for each of five denormalized variations of all database designs containing different synthetic contents. The results of the simulation study show that the proposed algorithm discovers exactly 72% of the unknown database schemas. The performance can be improved by fine-tuning the optimization parameters. The results of the simulation study also show that the devised algorithm can be used in many datasets to reveal structs of databases when only a raw dataset is available at hand.

Keywords

Relational databases; Automatic normalization; Genetic algorithms; Optimization; Decision support.

1 Introduction

It is generally impossible to represent real-world datasets in a single data table (Gärtner, 2003). However, many data analysis methods require a data table as input. These two opposite goals require processes of tabulation of raw data and summarization of structural tables. Once the data at hand are tabulated or stored in a regular form, it is possible and straightforward to reproduce the raw data in any format. Relational databases are good and effective solutions for storing the data in regular forms, usually called entities (O'Mara et al., 2016).

The relational model and database have been suggested by Codd after his relational data model suggestion (Codd, 1970). Relational databases, which are mostly used to store data in a structured form, have many advantages. Some of them are integrity, creating relationships, providing logical and physical independence, guaranteed consistency and accuracy, and retrieving data using queries easily and conveniently (Kraleva et al., 2018). However, designing the right database is a challenging task whether a raw dataset is available or not (Delplanque et al., 2018; Suranauwarat, 2017).

There are two ways commonly used for preprocessing and storing data: either storing data as the format collected or importing them into a pre-designed database (Kraleva et al., 2018). Storing a dataset as it is collected is an easy and fast task; however, the performance in the searching and querying stage can be drastically reduced.

Since individuals and organizations produce large volumes of data, and the numbers of data sources are steadily growing. On the other hand, the content of some of the available data sources is dynamic. Since the relational database design is not dynamic, it may be necessary to prepare new database designs for dynamic datasets. It has become a necessity to produce solutions that keep up with such features of datasets.

Designing a relational database is a process that requires expertise. In addition to this, data should be well explained, and tests and maintenance should be done in order to prevent unexpected errors. Moreover, relational database design is a subjective process. Designers may suggest different designs for the same dataset and none of them can be chosen as the single best design.

With this study, we propose a novel method that automatically provides a relational database design for an input dataset. We treat the relational database design as a multi-objective optimization problem and we use a genetic algorithm to solve this problem. Unlike the studies in the literature, the proposed method does not require any prior knowledge or custom settings except a dataset as input. Thus, the solution can be treated as a relational database design recommendation.

In Section 2, we give brief information on relational databases. In Section 3, a literature review of automatic normalization is given. In Section 4, a brief description of the genetic algorithm used in this paper is presented. In Sections 5 and 6, we present the proposed algorithm in a great detail and we show our research method. In Section 7, results of the simulation study are presented. Finally, Section 8 discusses and concludes.

2 Relational Databases

Relational databases are composed of tables (entities) and the relationships between these entities. An entity is an object which has attributes in its columns. Data are recorded in rows of an entity as tuples. A primary key, an attribute in an entity, ensures uniqueness of records. Designing a database schema based on the concepts of relational database can increase end users' and programmers' productivity (Codd, 1982; Kraleva et al., 2018; Read et al., 1992). With the development of relational databases, many advantages have been obtained over traditional file storage methods. Some of these advantages are: structured data, no data duplication, scalability, independence of content and method, data management, integration,

avoidance of inconsistency, multi-user support (Sumathi & Esakkirajan, 2007; Zhu, Zeng & Cao, 2010). However, the use of raw data sets brings with it many problems in terms of the advantages listed.

Designing a relational database requires applying the normalization process and realizing the rules determined with normal forms. Normalization is the process of dividing a dataset into small entities with many attributes by analysing the dependencies between them in a conflict-free and well-structured way (Hoffer et al., 2016; Lee, 1995). The majority of the designing process consists of normalizing the data model. Normal forms are the focal point of subjectivity in database design because their implementation can be interpreted by the designer. For example, the third normal form states that the attributes in an entity cannot be functionally dependent on each other if none of them are primary keys (Bernstein, 1976). In order to define functional dependencies, it is necessary to understand the data and foresee possible problems. The discovery of functional dependencies is vital to the process of defining and normalizing entities (Riordan, 2005).

Delplanque et al. (2018) mentioned six problems that arise in the architectural process of a relational database. These are dependencies between entities of the database, the impact of modifications on the database, stability on different instances of a database, testing process, synchronization between software and database itself, and the absence of automatic tools to define functional dependencies between entities. Relational databases are robust, effective and multiplatform solutions. Processes of designing and using a relational database are two different areas of expertise. When we focus on design, we find that functional dependence, automatic tools and the application of normal forms are important. These features are the main factors that make a database design difficult and require specialization, so much so that specialization can be considered a valuable talent and what most organizations need (Suranauwarat, 2017). While some work has been done to automate the database design process, a fully automated method has not yet been proposed. This study tries to make these processes that are subjective and require expertise fully automated.

3 Literature Review on Automatic Normalization

In the age of big data, data storage techniques also need to be compatible with dynamic data sources and types. It will be beneficial to use more techniques such as relational databases for datasets that increase in volume and number. However, the difficulties brought by this process require the database preparation process to become easier and further to become automated. There is a vast amount of works on the subject of automatic database normalization.

Du and Wery (1999) developed a utility called Micro. Micro can bring a relational database design to Boyce-Codd normal form using the information it receives from the user about attributes and functional dependencies between them. Dongare et al. (2011) developed a semi-automatic normalization tool called RDBNorma. The algorithm requires functional dependencies in order to design in third normal form. They compared RDBNorma with Micro and stated that RDBNorma had better performance than Micro.

Yazıcı and Karakaya (2007) developed a normalization tool in Mathematica. The tool can suggest a relational database design in third normal form, taking attributes and functional dependencies as input.

Bahmani et al. (2008) suggested an algorithm that requires a dependency graph diagram prepared by the user. The method generates a graph matrix using the dependency graph. After the steps of the method, a relational database design in Boyce-Codd normal form is obtained. The process depends on the dependency graph created by the user.

Ahmad et al. (2014) suggested a method to build a relational database design using Microsoft Excel. Their method requires a raw dataset in Microsoft Excel format and the dependencies between attributes. They stated that the main risk for the method they proposed may be misidentification of functional dependencies.

Some researchers have studied database normalization teaching and suggested various methods to improve the process. Verma (2012) stated that defining the primary key and remembering the rules of the normal forms can be a challenging task for students. Mitrovic (2002) developed an online smart education system called NORMIT to teach the concepts of normalization. Similarly, Fanguy et al. (2005) used gamification in their teaching methods. Soler et al. (2006) developed a normalization problem generator and asked for a design in Boyce-Codd normal form. Ahmedi et al. (2014) developed an e-learning tool called NormalDB. In addition to these methods, there are many studies that suggest relational database schemas for specific datasets (Dimitrieski et al., 2015; Kurnianda, 2018; Mallig, 2010; Tessler, 2002).

4 Genetic Algorithms

Genetic algorithms (GAs) are parallel search and optimization techniques that mimic the natural selection principle of genetics. The basics of GAs were suggested by Holland (1992) and developed by his doctoral students (Goldberg, 1989; Koza, 1995). A population consists of candidate solutions or individuals called chromosomes. Each single candidate solution has a fitness or cost value depending on the problem. A fitness or a cost value is the measurement of how a candidate solution satisfies the objective function. Since the GAs are blind and problem-independent, only the quality of a solution is considered. In GAs, each population forms a generation. The selection operator selects two individuals by considering their fitness or cost values. The crossover operator generates two offspring by combining the contents of the selected individuals. Random changes performed on the generated offspring are called mutations. The generated offspring form a new population called a generation. Many generations are created until a convergence criterion is met. The number of individuals in the population, the number of generations and the probability of applying mating operators determine the performance of a GA (Goldberg, 1989; Lim et al., 2017; Satman & Akadal, 2020).

The classical GAs are mainly work on binary chromosomes. A binary chromosome is a candidate solution or individual which consists of either 0s or 1s. The length of a chromosome is problem-specific. In this study, we use a binary-coded or classical GA in which an individual represents a database design. In our GA, the fitness function takes a database design as a chromosome for each individual in the population. Basically, equal-length segments of the chromosome specify in which entity the variables in the raw dataset should reside.

5 Proposed Algorithm

The algorithm proposed in the scope of this study enables creation of a relational database schema by obtaining only a dataset as input. Thanks to the algorithm, a dataset can transform a relational database design without any user intervention. The process of converting a dataset into a relational database consists of three stages:

- The first stage is to generate a database schema using genetic algorithms dependent on the given dataset.
- The second stage is to define the types of relationships between entities in the database schema.
- In the third stage, a fine-tuning process is applied to the result.

The flow of the algorithm is given in Figure 1. Each step is explained in detail in the next sections.

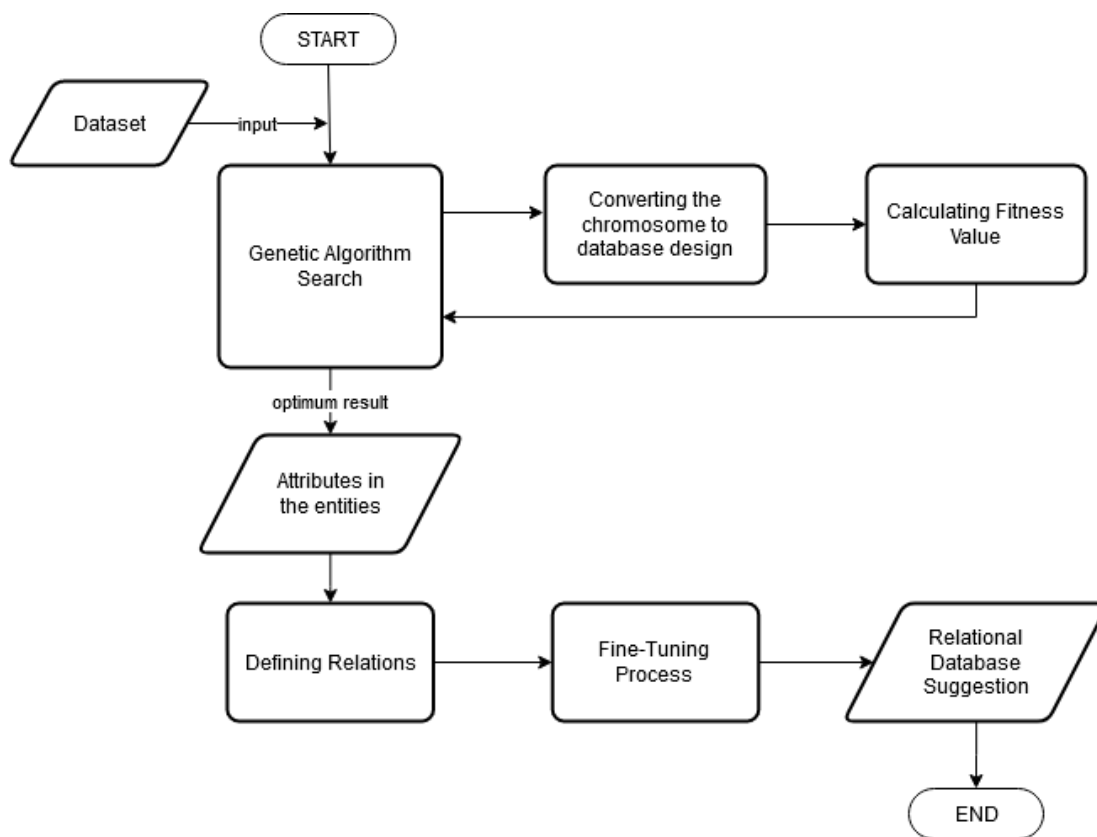


Figure 1. Algorithm flow chart.

5.1 Generating Database Schema

In this stage, the algorithm generates relational database entities and their attributes using the dataset. Database design has some essential features defined by normal forms. For example, good database design should have distinct records. In other words, the number of duplicate records should be minimized. Another example is that functionally dependent attributes must be clustered into entities.

In this study, database design process is considered a multi-objective optimization problem. It has been defined using a linear combination of four fitness functions to be minimized. We assume that the database schema with a minimum fitness function is considered an effective design. Fitness functions are explained in following subsections.

5.1.1 Fitness Function 1: Number of Cells

The first fitness function aims to minimize the number of cells in the generated database design. The definition of the function is

$$f_1 = \frac{\sum_{i=1}^n R_i A_i}{R_r A_r} \quad (1)$$

where n is the number of entities in a candidate database design, $i = 1, 2, \dots, n$, R_i is the number of records in the i th entity, R_r is the number of records in the dataset, A_i is the number of attributes in the i th entity, and A_r is the number of attributes in the dataset.

The numerator refers to the number of cells in the candidate database design, while the denominator indicates the dataset itself. f_1 shows the number of cells in a candidate database design based on the size of the dataset. Therefore, the value of f_1 is independent of the size of the dataset. f_1 takes values in the range $(0,1]$.

5.1.2 Fitness Function 2: Number of Repeated Records

It is an important rule that the records are distinct in a relational database. This property is chosen as the second fitness function. f_2 is defined as

$$f_2 = \sum_{i=1}^n \frac{R_i - (R_d)_i}{R_i} \quad (2)$$

where n is the number of entities in a candidate database design, $i = 1, 2, \dots, n$, R_i is the number of records in the i th entity, and $(R_d)_i$ is the number of distinct records in the i th entity. If a candidate entity includes distinct records, because of the equality of $R_i = (R_d)_i$, the contribution of the entity to f_2 would be 0. If the entity has the same value in all its records, the contribution would be almost 1 because of

$$\frac{R_i - 1}{R_i} \cong 1 \quad (3)$$

for a large dataset. f_2 takes values in the range $[0,1)$.

5.1.3 Fitness Function 3: Number of Entities

The previous fitness functions f_1 and f_2 can cause the algorithm to locate separate entities for each attribute. Thus, a database design is reached in which all entities contain an attribute, which is not a desirable solution. While we expect f_1 and f_2 to be minimized, we also look for the attributes in the entities to be grouped effectively. The third fitness function aims to reduce the number of entities. f_3 is defined as

$$f_3 = \frac{n}{2^m} \quad (4)$$

where n is the number of entities, the maximum number of entities based on the value m calculated using the chromosome size 2^m . A candidate solution has at least one entity and has a maximum of 2^m entities. f_3 takes values in the range $(0,1]$. It may not be the perfect solution that reduces both the cell count (f_1) and the number of entities. So, f_1 and f_3 , as in many other multi-objective optimization problems, may not have a common optimum solution. However, a good database design is chosen from Pareto-optimal solutions, which cover a set of solutions that are not comparable but are definitely better than others.

5.1.4 Fitness Function 4: Number of Numerical Entities

Some attributes only contain numerical values. If they take discrete values, the value of f_2 will increase due to the duplicate values. An attribute that holds age values is an example of this phenomenon. For a dataset with thousands of records, it is inevitable that the attribute containing the age records will contain duplicate records. Therefore, f_4 aims to avoid this situation. A numerical attribute expression is an attribute that takes numerical values; numerical entity refers to an entity whose records take numerical values. f_4 is defined as

$$f_4 = \frac{n_N}{n} \quad (5)$$

where n_N is the number of numerical entities and n is the number of entities in the candidate solution. Since it is possible that all entities are numerical or not, f_4 takes values in the range $[0,1]$.

5.1.5 Genotype and Phenotype

In this study, binary-coded GA is used. With the binary-coded GA, each chromosome is represented as a string of binary numbers which takes either 0 or 1 in each gene. Each binary-coded chromosome represents a candidate database design. Since the chromosome size differs by the number of attributes in the dataset, we have formulized the chromosome size as below.

$$2^m = A \quad (6)$$

$$m = \log_2 A \quad (7)$$

In Equations 6 and 7, A represents the number of attributes in the dataset and m is a function of A . The value m refers to the size of the genes and is used to indicate locations of attributes. Each attribute in the dataset is represented by m -sized genes in a chromosome. The *chsize* is defined as

$$chsize = [m] \times A \quad (8)$$

where $[m]$ is the smallest integer that is greater than or equal to m . In the algorithm, before starting of the GA search, chromosome size is calculated using Equation 8. In the fitness function, chromosomes are divided into m -sized parts and each single part represents the attributes' location in the entities.

5.1.6 Weights of Fitness Functions

The algorithm is based on a multi-objective genetic algorithm with a weighted sum of fitness functions. We tried various methods to define the best weights, but a global set of weights was not available for all cases. Therefore, we used equal weights in the study. Also, depending on the type of the dataset, we can use different weights to increase the success rate.

5.2 Defining Relation Types

The first stage enables creation of the entities that include the attributes of the dataset given as the input. A relational database schema includes some connections between the entities to reveal the integrity of the data. At this stage, the algorithm handles all binary combinations to expose potential relationships.

Step 1: Defining the abstract table: The algorithm produces a table called the abstract table. The abstract table allows representation of entities in terms of data repetition. The table includes two attributes for each entity. Attributes symbolize the values of the dataset in the same row. The records in the abstract table are representative. Regardless of the record itself, each record stands for a different record in the dataset.

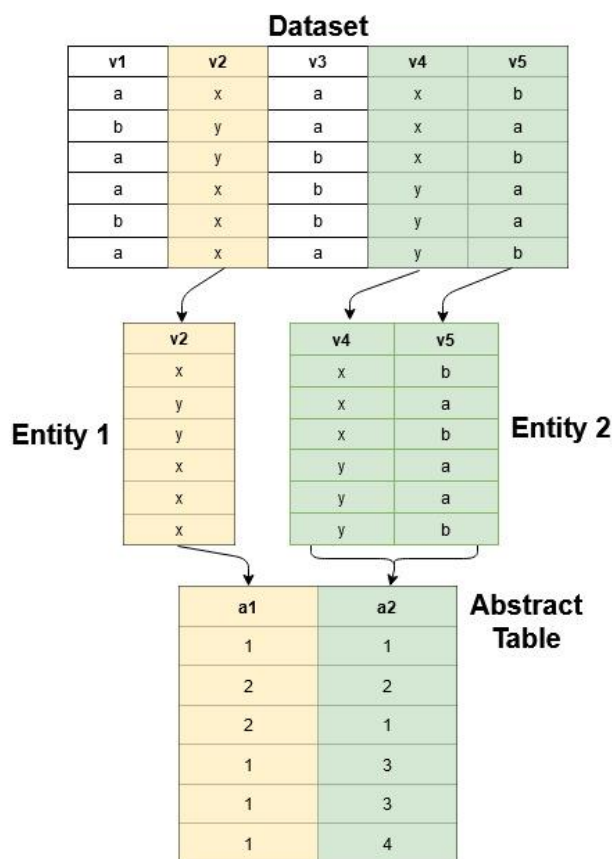


Figure 2. Example of abstract table.

Figure 2 shows an example of the abstract table. Assume that there is a dataset that includes five attributes. The abstract table was created for two entities which are composed by using the dataset. We examined the relation between two entities. The first entity has $v1$ and the second entity has $v4$ and $v5$. $a1$ represents *Entity 1* while $a2$ represents *Entity 2*, which includes $v4$ and $v5$ together. Since $a1$ depends on $v2$, the similarity of the values can be considered easily, but $a2$ is related to $v4$ and $v5$ together. In case $v4$ and $v5$ take the same value in a record and it is represented by a single value in $a2$. The example shows an abstract table for two entities which includes $v2$ for the first entity and $v4$ and $v5$ for the second entity. The content of records in the abstract table is not important for the process of defining the relations. The table just shows the repeats in the given entities.

Step 2: Number of transitions: In this step, the abstract table is analysed to reveal the transition between entities. We calculate two transition values called *transition1*, which shows the transition from *Entity 1* to *Entity 2*, and *transition2*, which shows the opposite of *transition1*. The values can be easily calculated by using an SQL query. For *transition1*:

```
SELECT COUNT (distinct(a1))
FROM 'Abstract Table'
GROUP BY a2
HAVING COUNT (distinct(a1)) > 1
```

can be used. Similarly, for *transition2*:

```
SELECT COUNT (distinct(a2))
FROM 'Abstract Table'
GROUP BY a1
HAVING COUNT (distinct(a2)) > 1
```

can be used. The numbers of transitions show the type of relation. Table 1 shows the relation types dependent on the number of transitions.

Table 1. Relation types by number of transitions.

<i>transition1</i>	<i>transition2</i>	Relation type
0	0	One to one
> 0	0	One to many
0	> 0	Many to one
> 0	> 0	Many to many

Step 3: The success rate of relation: Relation types are searched for each combination of the entities. The algorithm can report wrong relationships. In step 3, we calculate a success rate to show the quality of relationships. Although this may not produce a definitive solution, it is a guiding method for choosing the right relationships. The success rate is calculated as

$$SuccessRate = \frac{n}{n + transition1^2} \times \frac{n}{n + transition2^2} \quad (9)$$

where n indicates the number of rows in the given dataset as the input. The success rate takes values in the range (0,1]. Lower values point out much more transition, so the relationship may be quite likely valid with such a rate.

5.3 Fine-Tuning

The one-to-one relationship is rare, and it is seen as a special type of the one-to-many relationship (Sumathi & Esakkirajan, 2007). This type of relationship is mostly used for logical purposes depending on the database designer's decision. Therefore, it does not make sense for the output reported by the algorithm to have a one-to-one relationship. In the last step, the algorithm aggregates the entities which

have a one-to-one relationship. In the experimental stage of the study, this approach provides an improvement in the success of the algorithm.

6 Method

In order to measure the success of the suggested algorithm, we compare the output of the algorithm with a real database design for an input dataset. As a result of an extensive literature and web research, a sufficient number of datasets with the best database design could not be reached. Thus, we generate datasets using well-known and well-designed database designs in conjunction with the normalization process. For the generated datasets, the output of the algorithm can be compared to the best case, as the best solution would be pre-denormalization of database designs.

6.1 Data Collection

It is difficult to label a database design as a well-designed database because the normalization process is subjective and can be changed by the designer. In the database design selection process, Abantecart¹, Drupal², Joomla³, Moodle⁴, Opencart⁵ and WordPress⁶ open-source tools are used, which are developed by a wide community and are widely used to benefit from web-based systems. They have a high usage rate and are improved by the communities in terms of being bug-free and showing better performance. Thus, it is assumed that these database designs can be considered well-designed. Since database schemas have many entities, it is not easy to denormalize an entire database. For example, Abantecart and Opencart are e-commerce systems and have a large number of entities for various purposes such as management of customers, products, orders, content, etc. As a whole, not all entities produce a rational dataset. In the scope of this study, we split the comprehensive database designs into relatively small ones. Table 2 shows the database designs used.

Table 2. Sample databases.

#	Source database	Number of entities	Number of attributes	#	Source database	Number of entities	Number of attributes
1	Abantecart	8	25	26	Joomla	3	36
2	Abantecart	5	20	27	Moodle	9	69
3	Abantecart	2	15	28	Moodle	6	33
4	Abantecart	3	17	29	Moodle	6	109
5	Abantecart	5	32	30	Moodle	8	54
6	Abantecart	2	45	31	Moodle	4	22
7	Abantecart	6	60	32	Moodle	2	15
8	Abantecart	2	10	33	Opencart	5	23
9	Abantecart	5	43	34	Opencart	5	22
10	Abantecart	13	57	35	Opencart	3	48
11	Abantecart	17	66	36	Opencart	4	14
12	Abantecart	8	23	37	Opencart	4	32
13	Drupal	4	22	38	Opencart	2	23
14	Drupal	2	23	39	Opencart	3	12

¹ See, <https://www.abantecart.com/>

² See, <https://www.drupal.org/>

³ See, <https://www.joomla.org/>

⁴ See, <https://moodle.org/>

⁵ See, <https://www.opencart.com/>

⁶ See, <https://wordpress.org/>

#	Source database	Number of entities	Number of attributes	#	Source database	Number of entities	Number of attributes
15	Drupal	2	18	40	Opencart	6	19
16	Drupal	2	9	41	Opencart	3	11
17	Drupal	4	23	42	Opencart	2	11
18	Joomla	3	31	43	Opencart	3	5
19	Joomla	3	23	44	Opencart	2	4
20	Joomla	2	34	45	Opencart	4	9
21	Joomla	3	57	46	Opencart	2	22
22	Joomla	3	61	47	Wordpress	6	49
23	Joomla	3	14	48	Wordpress	2	30
24	Joomla	2	16	49	Wordpress	3	32
25	Joomla	3	22	50	Wordpress	4	34

During the dataset generation process, the datasets are produced for a total of 250 cases with five different versions of 50 database designs.

6.2 Measuring the Success of Output

In order to measure the performance of the algorithm, we perform a simulation study based on an environment where the best solutions of 250 datasets are searched ten times. Thus, 50 different scenarios are tested for each database design. The output of the algorithm and the real database designs are compared. Each binary attribute combination in the output design is compared with a real dataset. Figure 3 demonstrates an example.

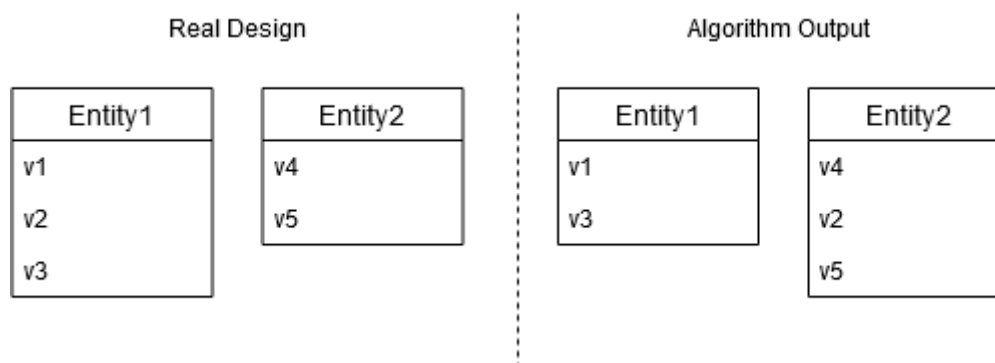


Figure 3. Example of assessing algorithm output.

The example given in Figure 3 includes two entities and five attributes. The left side shows an example of a real design, while the right side shows a representative solution. Five attributes can be grouped with ten different forms. *SuccessRate* measures the quality of the output and is defined as

$$SuccessRate = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n control(v_i, v_j)}{\binom{n}{2}} \quad (10)$$

where n is the number of attributes in the design, v_i and v_j are the attributes, $control()$ is a function that returns either 0 or 1, $i = 1, 2, \dots, n-1$, and $j = i+1, i+2, \dots, n$. The *control* function returns the binary state whether two attributes are in the same entity or not. For example, the success rate for the design showed in Figure 3 can be calculated as in Equation 11.

$$\frac{0 + 1 + 1 + 1 + 0 + 0 + 0 + 1 + 1 + 1}{10} = 0.6 \quad (11)$$

If the output and the actual design are the same, the success rate will be 1. Thus, the value is in the range [0,1].

6.3 Mutation Operator

A mutation operator is developed and integrated into the algorithm to enrich the search capabilities. Five different mutation operators are performed with equal probability at each turn.

1. The standard mutation operator of the GA is applied.
2. A randomly selected gene is replaced with a randomly generated gene.
3. Two arbitrarily selected genes are replaced.
4. A randomly selected gene is replaced with another randomly selected gene.
5. The chromosome is reconstructed using random sampling with replacement from a set of all genes.

6.4 Software

An R library is developed to perform the simulation study. The library depends on the GA package to run the genetic algorithm search (Scrucca, 2013, 2017). The fitness function is as mentioned in the section Generating Database Schema. The number of iterations is 500, the core count for the parallelization is 39, the weights of the sub fitness functions are equally 1, and the mutation probability is 0.1. The other parameters are used with the default settings for the GA package.

7 Findings

We performed a test suite to measure the performance of the suggested algorithm. We investigated 250 different datasets consisting of 50 different databases schemas with their five random variations. These calculations are repeated ten times.

Table 3. Simulation results of algorithm by databases.

Database	Number of attributes	Success rate				
		Min.	Max.	Mean	Median	Std. dev.
1	25	0.15	1	0.45	0.41	0.21
2	20	0.33	1	0.62	0.62	0.14
3	15	0.86	1	0.99	1	0.03
4	17	0.37	1	0.78	0.8	0.2
5	32	0.58	1	0.7	0.69	0.07
6	45	0.93	1	0.99	0.99	0.01
7	60	0.27	0.68	0.45	0.45	0.11
8	10	0.76	1	0.91	1	0.12
9	43	0.24	0.62	0.42	0.41	0.09
10	57	0.05	0.32	0.17	0.17	0.06
11	66	0.18	0.55	0.36	0.36	0.09
12	23	0.13	1	0.31	0.28	0.15
13	22	0.33	1	0.88	1	0.18
14	23	0.54	1	0.91	1	0.12
15	18	1	1	1	1	0
16	9	1	1	1	1	0
17	23	0.18	0.76	0.38	0.33	0.13
18	31	0.36	1	0.73	0.76	0.21

Database	Number of attributes	Success rate				
		Min.	Max.	Mean	Median	Std. dev.
19	23	0.87	1	0.99	1	0.02
20	34	0.88	1	0.98	1	0.04
21	57	0.58	1	0.74	0.72	0.1
22	61	0.38	0.82	0.62	0.64	0.09
23	14	0.79	1	0.99	1	0.04
24	16	0.85	1	0.99	1	0.03
25	22	0.28	0.88	0.55	0.52	0.14
26	36	0.68	0.98	0.89	0.9	0.07
27	69	0.19	0.68	0.41	0.39	0.12
28	33	0.12	0.41	0.25	0.24	0.07
29	109	0.1	0.31	0.18	0.18	0.05
30	54	0.12	0.73	0.27	0.21	0.14
31	22	0.34	0.87	0.49	0.47	0.12
32	15	0.72	1	0.99	1	0.04
33	23	0.31	1	0.61	0.64	0.19
34	22	0.35	1	0.7	0.72	0.15
35	48	0.78	1	0.99	1	0.03
36	14	0.46	1	0.79	0.8	0.2
37	32	0.31	0.85	0.45	0.45	0.1
38	23	0.79	1	0.91	0.89	0.08
39	12	0.55	1	0.92	0.94	0.09
40	19	0.24	1	0.6	0.55	0.19
41	11	0.47	1	0.94	1	0.12
42	11	0.78	1	0.87	0.78	0.11
43	5	1	1	1	1	0
44	4	1	1	1	1	0
45	9	0.44	1	0.96	1	0.1
46	22	1	1	1	1	0
47	49	0.21	0.59	0.37	0.35	0.08
48	30	0.93	1	0.98	1	0.03
49	32	0.89	1	0.93	0.92	0.03
50	34	0.43	1	0.76	0.75	0.15
Mean	30.08	0.52	0.9	0.72	0.73	0.09

We calculated the success rates as in Equation 10. The success rates take values in the range [0,1]. We listed the minimum, maximum, mean, median, and standard deviation of the success rates. Each cell shows the success rates of ten retries of the five different variations of the database given in the first column in the table. After 2500 runs of the algorithm, the minimum, the maximum, the mean, the median, and the standard deviation are 0.52, 0.9, 0.72, 0.73, and 0.09, respectively. The findings show that the success of the algorithm is 72% with a standard deviation of 0.09. In addition, the perfect solution is reached for 35 out of 50 databases with a success rate of 100%.

The findings show that the number of attributes can affect the success rate. To investigate the relation between the number of attributes and the success rate, we tested the null hypothesis

H_0 : There is no correlation between the number of attributes and the success rate.

in contrast to the alternative hypothesis

H_a : There is a correlation between the number of attributes and the success rate.

using Spearman Nonparametric Correlations Test. We obtained a p-value of 0.01. Therefore, there is sufficient evidence for rejecting the null hypothesis. The correlation coefficient is estimated as $r = -0.581$, so the number of attributes and the success rate are negatively correlated. The variables are shown in the scatter plot of Figure 4.

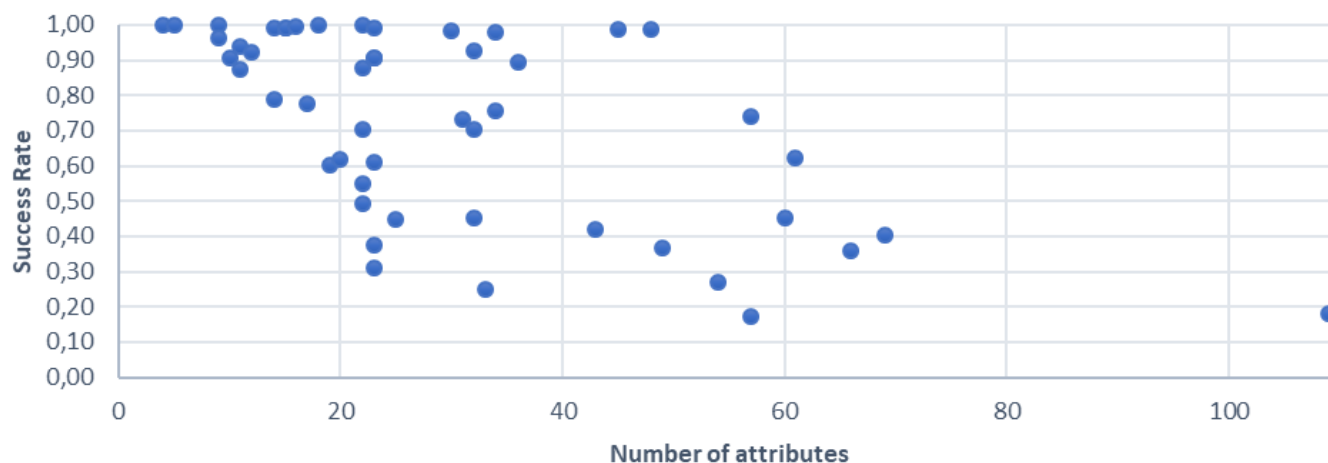


Figure 4. Graph of success rates by number of attributes.

In Figure 5, it is shown that the standard deviations of the success rates take values approximately between 0 and 0.2, but means are scattered over a larger range.

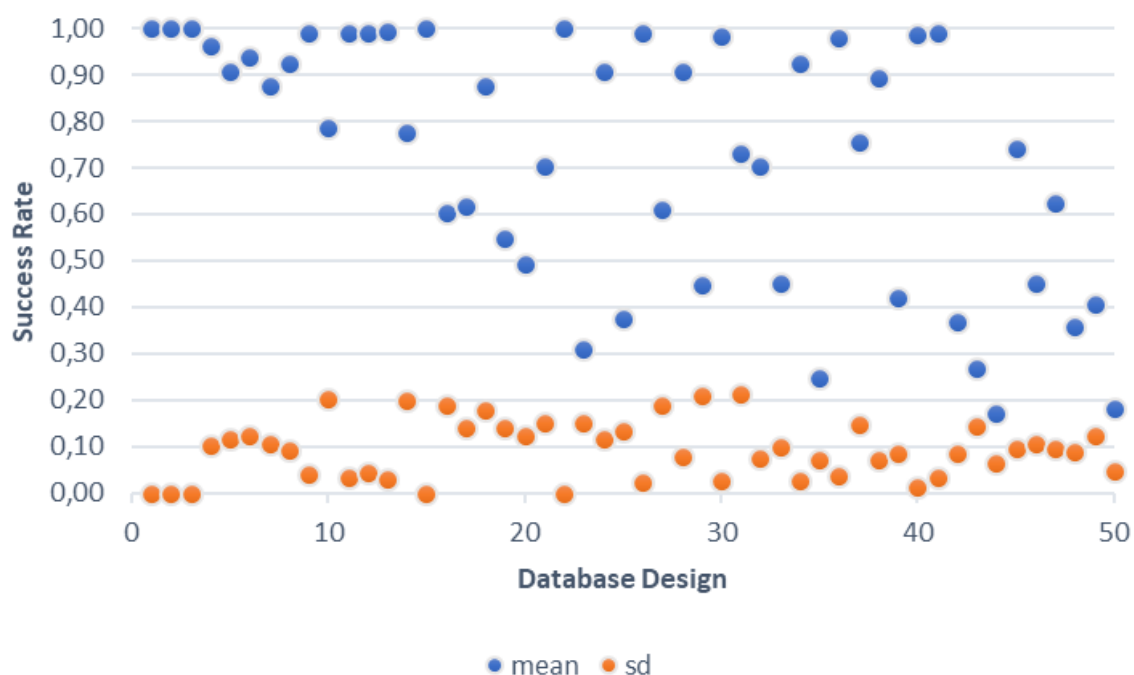


Figure 5. Success rates and their standard deviations.

8 Discussion and Conclusion

Designing a database, with or without prior knowledge, is a process that does not have an ideal solution and requires great skill and experience. Due to the constantly increasing data in terms of both variables and observations, creating a new design or updating the changing designs is a costly process in terms of

both budget and time. Automating this process can help reduce costs. In this study, we propose a new algorithm for creating a relational database design for any given dataset.

Our findings show that the suggested fully automatic database normalization algorithm performs with a success rate of 72% for various sizes of different datasets. We examined 250 datasets generated from 50 databases which are used in real projects and continuously being developed by a large community. We assumed that these databases had been well optimized by professionals for years.

Our suggested algorithm can generate a relational database design for a given dataset without any prior information and user input. However, since the success rate of the algorithm is not 100%, we cannot claim that it offers a definitive and ready-to-use solution. Nevertheless, the proposed algorithm can be a good support at the beginning of the design process, offering useful advice to designers and a shortened path.

The number of entities in database designs used in the simulations varied between 2 and 17, and the number of attributes of the datasets obtained by denormalization varied between 2 and 109. We performed a correlation test to measure the effect of the change in the number of attributes on the success rate and concluded that these two values are significantly correlated. A different set of optimization parameters can influence search results in terms of achieving a high success rate for a high number of attributes, and this can be regarded as a topic for future study.

In the proposed algorithm, we considered the normalization process as an optimization problem. Since our optimization problem consisted of four fitness functions, we used the weighted sum method. The method requires defining four weights for each fitness function. However, we could not come up with a valid weight set for each problem, and we used equal weights for each. Searching and finding topic-specific or global weights can result in a higher success rate. It can also be viewed as potential future work.

We designed the algorithm to create a database design with an input dataset. Using a handled database as an input to normalize again is impossible. Alternatively, however, users can denormalize a database of any size and normalize it with the proposed algorithm again. With this approach, a validation process can be executed. This may be the subject of future work.

The genetic algorithm used in the proposed algorithm is population-based and uses iterations. The algorithm creates a candidate database design for each candidate solution, and places input data into the database. Then the fitness functions are calculated based on the handled database. The size of input data affects the duration of the preparation process for each candidate solution, but it is not crucial in terms of calculation of the fitness function. The algorithm may run more slowly with big datasets, but using a sampling method may reduce the search duration. So, the algorithm can be used for input raw datasets of any size. Devising a novel sampling method can be the subject of future work too.

As another limitation of the study, the dataset entered must be in the first normal form. The first normal form requires each cell in the dataset to have atomic data. This arrangement should be performed by the user as a preprocessing step. As a result, it is not easy to specify what normal form the output is in. Since the steps of the algorithm do not contain normal form rules, the output can be in any normal form.

The algorithm can be performed for denormalized datasets that do not contain complex relationships such as star schema database designs. We also recommend increasing the number of iterations and population size for large datasets. As a future work, we plan to publish an implementation of the algorithm as an R package on the CRAN repository of R.

Additional Information and Declarations

Acknowledgments: We would like to thank Istanbul University Real Data Application and Research Laboratory for the opportunity it offers to perform analyses.

Conflict of Interests: The algorithm proposed in the article is under patent protection by the Turkish Patent and Trademark Office with the registration number 2017 23004. All of the authors of this study also own this patent.


Author Contributions: E.A.: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Software, Visualization, Writing – Original Draft. M.H.S: Conceptualization, Investigation, Methodology, Project Administration, Supervision, Validation, Writing – Review & Editing.

Data Availability: The data that support the findings of this study are available from the corresponding author.

References

- Ahmad, R., Saknakosnak, P., & Hooi, Y. K. (2014). Excel-database converting system using data normalization technique. In *Proceedings of the First International Conference on Advanced Data and Information Engineering*, (pp. 23–30). Springer. https://doi.org/10.1007/978-981-4585-18-7_3
- Ahmedi, L., Jakupi, N., & Jajaga, E. (2014). NORMALDB-A Logic-Based Interactive e-Learning Tool for Database Normalization and Denormalization. In *eLmL 2012: The Fourth International Conference on Mobile, Hybrid, and On-line Learning*. http://personales.upv.es/thinkmind/dl/conferences/elml/elml_2012/elml_2012_2_40_50084.pdf
- Bahmani, A. H., Naghibzadeh, M., & Bahmani, B. (2008). Automatic database normalization and primary key generation. In *Canadian Conference on Electrical and Computer Engineering*, (pp. 11–16). IEEE. <https://doi.org/10.1109/CCECE.2008.4564486>
- Bernstein, P. A. (1976). Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transactions on Database Systems*, 1(4), 277–298. <https://doi.org/10.1145/320493.320489>
- Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377–387. <https://doi.org/10.1145/362384.362685>
- Codd, E. F. (1982). Relational database: A practical foundation for productivity. *Communications of the ACM*, 25(2), 109–117. <https://doi.org/10.1145/358396.358400>
- Delplanque, J., Etien, A., Anquetil, N., & Auverlot, O. (2018). Relational database schema evolution: An industrial case study. In *Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018*, (pp. 635–644). IEEE. <https://doi.org/10.1109/ICSME.2018.00073>
- Dimitrieski, V., Čeliković, M., Aleksić, S., Ristić, S., Alargt, A., & Luković, I. (2015). Concepts and evaluation of the extended entity-relationship approach to database design in a multi-paradigm information system modeling tool. *Computer Languages, Systems and Structures*, 44, 299–318. <https://doi.org/10.1016/j.cl.2015.08.011>
- Dongare, Y., Dhabe, P., & Deshmukh, S. (2011). RDBNorma: A semi-automated tool for relational database schema normalization up to third normal form. *International Journal of Database Management Systems*, 3(1), 133–154. <https://doi.org/10.5121/ijdms.2011.3109>
- Du, H., & Wery, L. (1999). Micro: A normalization tool for relational database designers. *Journal of Network and Computer Applications*, 22(4), 215–232. <https://doi.org/10.1006/jnca.1999.0096>
- Fanguy, R. A., & Betty Kleen, N. A. (2005). Normalization Shootout: A Competitive Game That Impacts Student Learning. *Issues in Information Systems*, 6(1), 21–27. https://doi.org/10.48009/1_iis_2005_21-27
- Gärtner, T. (2003). A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1), 49–58. <https://doi.org/10.1145/959242.959248>
- Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman Publishing.
- Hoffer, J., Ramesh, V., & Topi, H. (2016). *Modern Database Management*. Pearson.
- Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press.
- Koza, J. R. (1995). Survey of genetic algorithms and genetic programming. In *Proceedings of WESCON'95*, (pp. 589–594). IEEE. <https://doi.org/10.1109/wescon.1995.485447>
- Kraleva, R., Kraleva, V., Sinyagina, N., Koprinkova-Hristova, P., & Bocheva, N. (2018). Design and analysis of a relational database for behavioral experiments data processing. *International Journal of Online and Biomedical Engineering*, 14(2), 117–132. <https://doi.org/10.3991/ijoe.v14i02.7988>
- Kurnianda, N. R. (2018). Database Design for Customer Retention and Loyalty Administration Information System. https://www.academia.edu/37566052/Database_Design_for_Customer_Retention_and_Loyalty_Administration_Information_System
- Lee, H. (1995). Justifying database normalization: a cost/benefit model. *Information Processing and Management*, 31(1), 59–67. [https://doi.org/10.1016/0306-4573\(95\)80006-F](https://doi.org/10.1016/0306-4573(95)80006-F)

- Lim, S. M., Sultan, A. B. M., Sulaiman, M. N., Mustapha, A., & Leong, K. Y. (2017). Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, 7(1), 9–12. <https://doi.org/10.18178/ijmlc.2017.7.1.611>
- Mallig, N. (2010). A relational database for bibliometric analysis. *Journal of Informetrics*, 4(4), 564–580. <https://doi.org/10.1016/j.joi.2010.06.007>
- Mitrovic, A. (2002). NORMIT: A Web-enabled tutor for database normalization. In *Proceedings - International Conference on Computers in Education, ICCE 2002*, (pp. 1276–1280). IEEE. <https://doi.org/10.1109/CIE.2002.1186210>
- O'Mara, J., Meredig, B., & Michel, K. (2016). Materials Data Infrastructure: A Case Study of the Citrination Platform to Examine Data Import, Storage, and Access. *JOM*, 68(8), 2031–2034. <https://doi.org/10.1007/s11837-016-1984-0>
- Read, R. L., Fussell, D. S., & Silberschatz, A. (1992). A Multi-Resolution Relational Data Model. <https://www.vldb.org/conf/1992/P139.PDF>
- Riordan, R. M. (2005). *Designing effective database systems*. Addison-Wesley Professional.
- Satman, M. H., & Akadal, E. (2020). Machine Coded Compact Genetic Algorithms for Real Parameter Optimization Problems. *Alphanumeric Journal*, 8(1), 43–58. <https://doi.org/10.17093/alphanumeric.576919>
- Scrucca, L. (2013). GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4), 1–37. <https://doi.org/10.18637/jss.v053.i04>
- Scrucca, L. (2017). On Some Extensions to GA Package: Hybrid Optimisation, Parallelisation and Islands Evolution On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution. *The R Journal*, 9(1), 187–206. <https://doi.org/10.32614/RJ-2017-008>
- Soler, J., Boada, I., Prados, F., & Poch, J. (2006). A web-based problem-solving environment for database normalization. In *Proceedings of the 8th International Symposium on Computers in Education, SIIE 2006*, (pp. 86–93). ACME. <http://acme.udg.cat/articles/siie2006.pdf>
- Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of Relational Database Management Systems*. Springer. <https://doi.org/10.1007/978-3-540-48399-1>
- Suranauwarat, S. (2017). An Approach to Solving Technical Difficulties Facing Non-CS Students in a Database Class. *International Journal of Modern Education and Computer Science*, 9(2), 14–26. <https://doi.org/10.5815/ijmeecs.2017.02.02>
- Tessler, S. (2002). *Data Model and Relational Database Design for the New England Water-Use Data System (USGS Open-File Report 01-359)*. USGS. <https://pubs.usgs.gov/of/2001/ofr01359/>
- Verma, S. (2012). Comparing Manual and Automatic Normalization Techniques for Relational Database. <https://www.semanticscholar.org/paper/COMPARING-MANUAL-AND-AUTOMATIC-NORMALIZATION-FOR-Verma/769b6e0b6b6cc84ff7e4822133b99367ffcb0531>
- Yazici, A., & Karakaya, Z. (2007). JMathNorm: A database normalization tool using mathematica. In *Computational Science – ICCS 2007*, (pp. 186–193). Springer. https://doi.org/10.1007/978-3-540-72586-2_27
- Zhu, X. H., Zeng, Q. L., & Cao, Q. H. (2010). A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database. *International Journal of Digital Content Technology and its Applications*, 4(8), 182–192.

Editorial record: The article has been peer-reviewed. First submission received on 26 July 2022. Revision received on 25 August 2022. Accepted for publication on 8 September 2022. The editor in charge of coordinating the peer-review of this manuscript and approving it for publication was Zdenek Smutny .

Acta Informatica Pragensia is published by Prague University of Economics and Business, Czech Republic.

ISSN: 1805-4951
