DOI: https://doi.org/%2010.34123/jurnalasks.v16i2.796

Jurnal Aplikasi Statistika & Komputasi Statistik
Vol. 16 No. 2, December 2024
e- ISSN: 2615-1367, p-ISSN:2086-4132

Available online at https://jurnal.stis.ac.id/

# Implementation of a RESTful API-Based Evolutionary Algorithm in a Microservices Architecture for Course Timetabling

**Zuhdi Ali Hisyam[1*], Farid Ridho[2], Arbi Setiyawan[3]**

*[1,2]Politeknik Statistika STIS, East Jakarta, Indonesia, [3]School of Management, Jiangsu University, China*
**Corresponding Author: E-mail address: 222011642@stis.ac.id*

**ARTICLE INFO**

***Article history:***
Received 3 September, 2024
Revised 29 October, 2024
Accepted 9 November, 2024
Published 31 December, 2024

***Keywords:***
Course Timetabling; Evolutionary Algorithm; (1+1) Evolutionary Strategies; RESTful API; Microservices; Cost Function; Black Box Testing

## Abstract

**Introduction/Main Objectives:** Implement an evolutionary algorithm within a RESTful API for a course timetabling system that employs a microservices architecture. **Background Problems:** The current course timetabling at Politeknik Statistika STIS uses the third-party application (aSc Timetables), which lacks a generator as a service, resulting in its inefficiency due to the lack of integration with SIPADU NG. **Novelty:** The evolutionary algorithm is built as a service (RESTful API) within a microservices architecture and supports custom constraints for timetables. **Research Methods:** One of the evolutionary algorithm families, the (1+1) evolutionary strategy, is implemented and used to create a course timetable 1000 times. Each course timetable created will have its cost calculated to assess the goodness of the algorithm implementation. The developed RESTful API is also evaluated through black box testing. **Finding/Results:** For the odd semester data, 40.5% of the trials yielded a cost value between 4 and 5, while for the even semester, all trials produced a cost value below 1. The resulting cost value is close to 0, which indicates that the timetable created has minimal violations. Additionally, black box testing concluded that the service operates as expected, delivering the anticipated output.

## 1. Introduction

Timetabling or scheduling is one of many issues institutions deal with, and the academic field is no exception [1]. This issue is usually called the University Course Timetabling Problem (UCTTP). In a big institution like a university, timetabling is a worthwhile task that needs to be resolved efficiently. In the context of lectures (courses), timetabling is a way to combine lecturers, groups of students, subjects, time of the lectures, and classroom availability with some sort of rules, which results in a flexible and conflictless course timetable. Flexible means the timetable can accommodate its resources, for example, a classroom preference from some lecturer, a time preference of the lecturers, or a forbidden time for lectures.

Politeknik Statistika STIS is a college in the Badan Pusat Statistik (BPS) environment. While Politeknik Statistika STIS is functionally built by the Head of BPS, it is still technically built by the Indonesian Ministry of Research, Technology and Higher Education [2]. Politeknik Statistika STIS, as a college institution, faces the same problem as many other colleges/universities, which is related to course scheduling or timetabling.

**Copyright:** © 2024 by the authors.
This an open access article distributed under the terms and conditions of the CC BY-NC 4.0.

Courses timetabling problems can be resolved manually or automatically by using a system. However, making a manual timetable can take much time and effort, making it difficult, especially for a larger institution with many entities like Politeknik Statistika STIS. That is why the utilization of an algorithm-based scheduling system can be done to solve timetabling problems.

At this moment, Politeknik Statistika STIS is utilizing a third-party application (aSc TimeTables) to build its timetables. On aSc TimeTables official site, this application does not provide the generator as a service and resulted in a problem in which the timetables result of this application cannot be integrated with SIPADU NG (*Sistem Informasi Perkuliahan Terpadu Next Generation*). Building timetables using a third-party application can impact inefficiency because of the need to manually input timetable results into SIPADU NG. Also, users need to define constraints themselves, whether lecturer or general constraints. Lecturer constraint contains the availability of teaching time and lecture room (classroom) preferences desired by the lecturer. These constraints must be defined one by one, which, of course, takes a long time. General constraint is a limitation that is inherent in many entities, be it a few lecturers, some classes of students, or all entities. These, too, must be defined one by one. After completing the schedule, the timetable must still be inputted into SIPADU NG again. This procedure raises the opportunity to create a timetabling system that is integrated with SIPADU NG to be more efficient.

Creating a timetabling system requires allocating each lecture containing lecturers, students, and lecture rooms to a specific time so there is no conflict between the three. In addition, timetables are also required to follow the rules (constraints) by the needs of the Politeknik Statistika STIS. As a result, a timetabling algorithm must be implemented to address this need since manually creating timetables is both time-consuming and inefficient. Studies [3], [4], [5] demonstrate that the timetabling problem can be solved using algorithms or formulas.

The course timetabling system that will be created is divided into three modules: the front-end web development module, the constraints or back-end rule provider module, and the timetabling module. This research will focus solely on the timetabling module, namely how to manage the available resources to produce the best timetable possible. These resources include the availability of time and preferences for lecturers' rooms, lecture rooms, lecture meetings, and other constraints.

Course timetabling problems are classified as non-polynomial time (NP) and combinatorial optimization problems (COP), indicating that they can be addressed using optimization algorithms to generate the desired optimal timetable [1]. Many algorithms can be used to address course timetabling problems, one of which is the evolutionary algorithm (EA). Evolutionary Algorithms are a subset of Evolutionary Computations (EC) and fall within modern heuristic-based search methods. Thanks to their flexibility and robust characteristics derived from Evolutionary Computation, they are a practical problem-solving approach for a wide range of global optimization challenges. EAs have been successfully applied in a wide range of problems [6], [7], [8].

Additionally, studies [9], [10], [11] implement evolutionary algorithms to address course timetabling, demonstrating that timetabling problems can be solved quickly, efficiently, and effectively. However, as done in this study, only some studies implement the algorithm into a service. Moreover, the algorithm in this study accommodates custom constraints, making it more flexible when creating course timetables.

The algorithm applied to the timetabling system of the Politeknik Statistika STIS in this study is one type of evolutionary algorithm, namely (1+1) evolutionary strategies (ES). Because this algorithm works randomly, several solutions will be made by multiprocessing in creating schedules. One of the best solutions will be selected from several solutions due to this service.

In developing a timetabling system, the concept of microservices is used. Therefore, the timetable module will later be added to the service. This RESTful API-based service will generate timetables based on existing data. Other modules will later use the results, especially the front-end module, to display to users. That way, users can see the lecture schedules that have been created and allow users to change some lecture schedules if needed.

## 2. Material and Method

### *2.1. Data*

Several methods were used to collect data for this study.

1. Interview

The interview method involves direct communication with relevant parties to gather data. Interviews were conducted with the IT Unit of the Politeknik Statistika STIS, the department responsible for creating the semester timetable. These interviews aimed to discuss the current course timetabling business process at Politeknik Statistika STIS, identify any existing problems or challenges, and understand the specific needs of the subject matter experts.

Additionally, interviews were conducted with the *Bagian Administrasi, Akademik, dan Kemahasiswaan*, or Academic Affairs Division, who created the semester timetable. These interviews aimed to identify the usual constraints imposed each semester, providing a guideline for developing a system to address these limitations. Furthermore, these interviews were conducted to request course data that would be used for testing purposes.

2. Observation

   *Observation* is a data collection method that involves directly observing the research object. This method involves observing the requirements for creating a timetable, such as data on all student classes, lecturers, classrooms, available time slots, and, most importantly, observing the constraints of the course timetabling at the Politeknik Statistika STIS. This activity is necessary for the researcher to translate these findings into a system.

3. Literature Study

   The objective of this method is to conduct an in-depth study of theories related to course timetabling and system development by gathering literature from relevant research papers. The references collected are related to timetabling algorithm theories, system implementation theories, and so on. The sources obtained are varied, ranging from scientific journals, books, the Internet, and other valid sources.

## *2.2. Timetabling*

The word "scheduling" is regarded as a broad term that includes a range of issues, such as timetabling, sequencing, and rostering. Research [1] defines timetabling as the process of allocating resources to activities over time and space, subject to a set of constraints. This problem has gained prominence in various domains, including education. Research [12] further classifies timetabling problems into three categories: school, exam, and course timetabling. This study focuses on the course timetabling.

Course timetabling entails assigning courses, lecturers, and students to specific time slots and rooms, subject to a multitude of constraints. As identified by research [3], [12], these constraints can be categorized into hard and soft constraints. Hard constraints are mandatory requirements that must be met, while soft constraints are desirable but not strictly necessary, aiming to optimize the overall timetable. The specific constraints for course timetabling at the Politeknik Statistika STIS may vary from semester to semester.

At the Politeknik Statistika STIS, the academic timetable is divided into time blocks. A standard day consists of 9-time blocks, and classes are held 5 days a week, totaling 45 blocks per week. The duration of each course is determined by the number of credits associated with the course. For instance, a 3-credit course will occupy three consecutive time blocks, while a 2-credit course will occupy 2. Using the same concept as [13], we can define a set of working days from Sunday to Friday as $D = \{1, 2, 3, 4, 5\}$, where Sunday is one and Friday is five. Each day $d \in D$ consist of $P$ block time which $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Course assignment is represented as x $x_{d,p,l,c,r,s}$, where it equal to one if a subject $c \in C$, taken by student group $s \in S$, is scheduled to be taught by a lecturer $l \in L$ on a day $d \in D$, at a block time $p \in P$ in a room $r \in R$.

The main purpose of timetabling is to create a timetable so that there is no conflict between lecturers, student groups, and classes. At most, one subject must be assigned to only one lecturer in one room at each time block. Hence, a lecturer could only attend one course at a time, as shown in (2). A subject is taught once per time block/room, as shown in (1), and could not be taught in two different rooms simultaneously, as shown in (3). More models can be seen in [13].

$$\sum_{l \in L} \sum_{c \in C} \sum_{r \in R} x_{d,p,l,c,r,s} \leq 1 \tag{1}$$

$$\sum_{c\in C}\sum_{r\in R}\sum_{s\in S} x_{d,p,l,c,r,s} \leq 1 \quad (2)$$

$$\sum_{l\in L}\sum_{c\in C}\sum_{s\in S} x_{d,p,l,c,r,s} \leq 1 \quad (3)$$

The timetabling system created in this study will implement an evolutionary algorithm. The Academic, Administration, and Student Affairs Division of Politeknik Statistika STIS will use this system to create a timetable every semester (six months).

## 2.3. Evolutionary algorithm: (1+1) Evolutionary strategies

Evolutionary strategies are a type of search algorithm classified as evolutionary algorithms. These algorithms, in general, continuously evolve solutions through mutation until an optimal solution is found or the maximum number of iterations is reached [14]. Research [6] mapped several search algorithms, as shown in Figure 1.

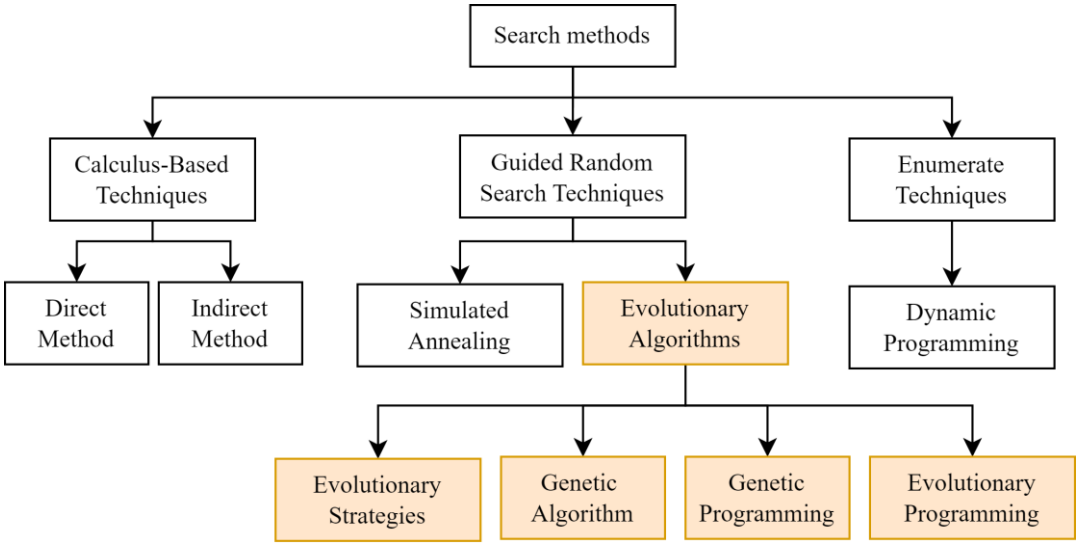


**Figure. 1**. Search algorithms and their family.

Generally, evolutionary algorithms operate on a population of individuals. These individuals undergo crossover, a process similar to genetic recombination, as one of the mechanisms of evolution. However, the algorithm used in this study differs. This research employs (1+1) evolutionary strategies, the simplest form of evolutionary strategies. As the name suggests, it operates on only two individuals: one parent and one child produced by mutating the parent. If the child is better than the parent, it replaces the parent in the next generation [14]. According to Fernandes [9], a cost function determines whether the child is better than the parents. The cost function will be further explained in the following equation.

The algorithm implementation is built on a microservices architecture, and a RESTful API is implemented. The service's functionality will be tested using black-box testing to verify if it can provide the expected responses. Additionally, the implementation of the evolutionary algorithm will be evaluated based on the resulting cost value using the Politeknik Statistika STIS's course data. The algorithm implementation is summarized in Figure 2.

## 2.4. RESTful API

An API (Application Programming Interface) is a mechanism for exchanging data between two software applications, adhering to specific rules and protocols. A RESTful API is an API that implements the REST (Representational State Transfer) architectural style. The REST architecture was first introduced by Roy Fielding in 2000. Derived from network-based architectures, REST incorporates

concepts such as client-server and layered systems [15]. This procedure allows each component of the architecture to be isolated while still enabling communication. Communication between the client and server in REST utilizes HTTP and employs request methods such as GET, PUT, POST, DELETE, PATCH, and others.

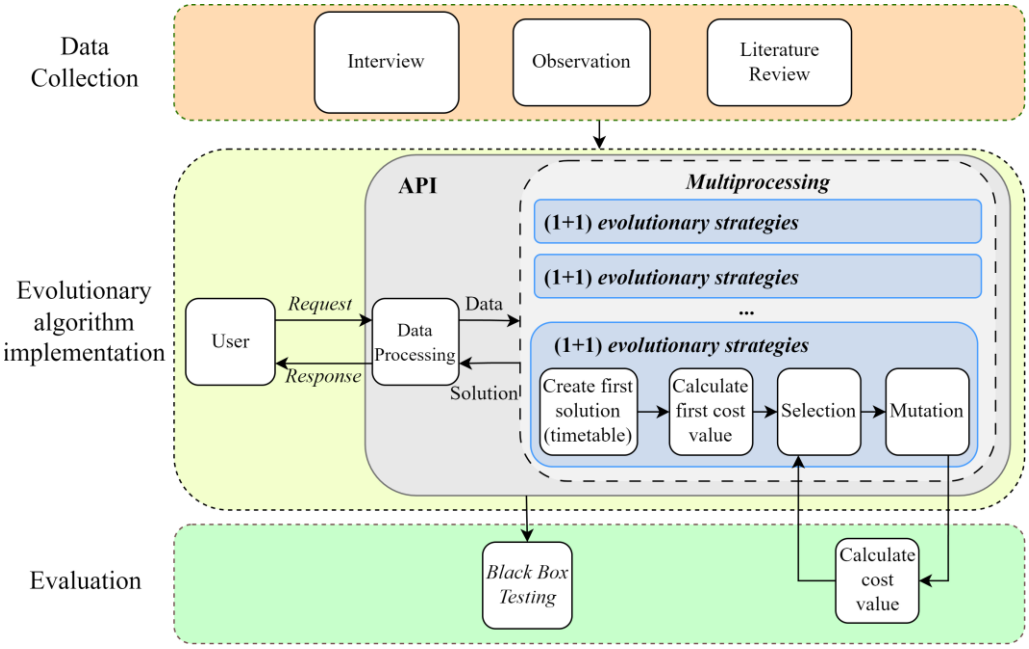


**Figure. 2**. Research method.

Research [16] indicates that a service created using RESTful API is easier to use without relying on the same operating system with another service, programming languages, or databases because the service communicates using a standard data format using JSON. Besides, REST architecture has several advantages compared to one of the well-known network communication architectures, namely SOAP [17]. This architecture is why evolutionary algorithms are implemented into a RESTful API in this study.

After the RESTful API is created, it is evaluated using testing. Testing is used to ensure the RESTful API's quality in general. The importance of testing has led developers to develop methodologies and approaches that support the testing process. The RESTful API is tested using black box testing, which ignores the internal structure and implementation of the component being tested, focusing solely on its inputs and outputs [18].

## *2.5. Evaluation Method*

This research will be evaluated using two tests. First, we tested the RESTful API using black-box testing, checking if the system gives the correct output for every input. Second, we tested the (1+1) evolutionary algorithm by creating 1,000 different course timetables using data from the Politeknik Statistika STIS for the 2023/2024 academic year. We will then calculate a cost value for each timetable. A lower cost means a better timetable. The cost function used in this study will be explained later.

## *2.6. Microservices Architecture*

Microservices is an architectural and organizational approach to software development in which software is composed of small, independent services that communicate through well-defined APIs [19]. Microservices architecture have several benefits, including increased modularity, flexible configuration, easier development, easier maintenance, and increased productivity [20]. The adoption of microservices simplifies system development as each service can be built using different tools (flexible), ensuring that errors in one service do not impact others. Microservices also enable faster system development. Multiple services can be developed simultaneously by different teams without having to wait for other services to be completed.

# 3. Result and Discussion

## *3.1. Data Preparation*

Before implementing (1+1)-ES, several data are needed to create a course timetable. The data are:

1. Room data that can be used for courses. Room data is categorized based on building or floor.
2. Lecturer data contains information about the lecturer's name as well as their preferred teaching time and room.
3. Course meeting data combines lecturer, students, and subjects. It also contains information about the duration of the course (two blocks or three blocks) and the type of course (theory or practical).
4. The user defines constraints data or limitations that must be applied to the course timetable.
5. Configuration data contains information about the maximum number of iterations in the implementation of (1+1) evolutionary strategies, the maximum number of courses in a day for lecturers and students along with their priorities, the number of timetabling trials to be used in multiprocessing, the number of candidate solutions to be returned in the response, and the list of blocks that can be used for the start of courses, both with a duration of 2 blocks and three blocks.

Data preparation involves processing the data into a specific format to facilitate running the (1+1)-ES algorithm. The initial data preparation focused on time availability for lecturers, students, and classrooms, which were stored in an array. The possible value of the time availability array is: '0' meaning available, '99' meaning not available, and '999' meaning optional. This array has a length of 45, corresponding to the courses at the Politeknik Statistika STIS, which consist of 5 days with nine blocks each day. The representation of the array index with the day and course block is shown in Table 1.

**Table 1.** Representation of Day and Time block into index array

| Day | Time blocks of course | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Monday | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Tuesday | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Wednesday | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| Thursday | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| Friday | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 |

The next step in data preparation is to apply constraints related to prohibited blocks, mandatory blocks, and required rooms as defined by the user. Specifically, the required room constraint is limited to courses only. Block-related constraints are applied by modifying the values in the time availability array. Meanwhile, room-related constraints are applied by creating a list of courses and possible rooms.

The following data preparation step involves listing the possible time blocks for all lecturers, both for 2-credit and 3-credit courses. This step matches the lecturer's availability with the template for 2-credit and 3-credit courses. Figure 3 illustrates the process of obtaining possible time blocks with a duration of 3 credits for lecturer Ach001. The exact process is applied to 2-credit courses and all lecturers.

In Figure 3, the green color means the time block is free and can be filled with a course, while the red color means the time block is prohibited for courses, and the yellow color means that a course can start from this block because there will be no violation of the constraints if it starts from this block. Other data preparations include handling courses with theory and practical components, team teaching (courses taught by more than one lecturer), and creating course lists for lecturers and students.

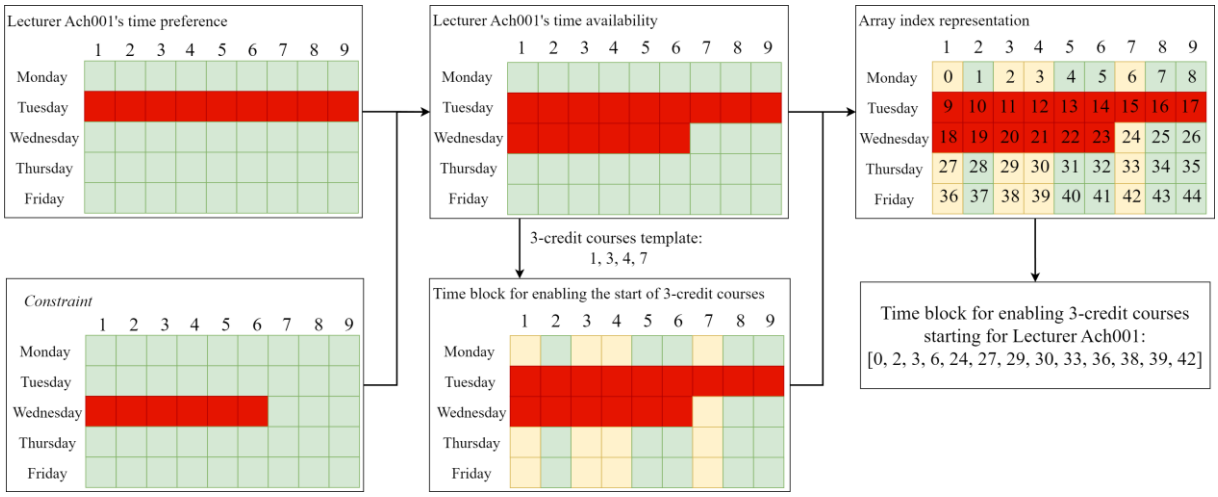


**Figure. 3**. An example calculation of representing possible course start time blocks into an array index.

## *3.2. Implementation of (1+1) Evolutionary Strategies*

The implementation of (1+1)-ES works using a chromosome. The chromosome contains all the resources needed to create a timetable. In this study, the chromosome is defined as an array of length 7. Each array index stores information about:

1. All of the course meetings (Figure 5a) are an array.
2. The array contains all constraints, time preference, time availability (Figure 5b), time blocks for enabling 3-credit courses and 2-credit courses starting, and a list of courses (Figure 5b) belonging to each lecturer.
3. The array contains all-time availability (Figure 5d) belonging to each classroom.
4. The array contains all constraints, time availability (Figure 5c), and a list of courses (Figure 5c) belonging to each group class.
5. The array contains all plotting of subjects and time blocks.
6. Configuration of timetable, including maximal iteration ("iterasi"), number of multiprocessing to be performed ("jml_percobaan"), number of timetable candidate solution to be returned ("jml_kandidat_solusi"), number of maximal courses in one day ("maks_matkul"), weight penalty of maximal courses in one day ("maks_matkul_prioritas"), list of courses that needs special rooms ("ruang_wajib"), and time blocks for starting lectures for three credits ("blok_mulai_3_sks") and two credits ("blok_mulai_2_sks"). An example of configuration is below in Table 4.
7. An array of custom constraints is defined by the user (Explained later in Table 4).

The flowchart of (1+1)-ES implementation can be seen in Figure 4. The implementation process of (1+1)-ES involves the following steps:

1. Creation of initial chromosome as the first solution

   Initial chromosome generation is conducted by selecting a room and a starting block time for each course meeting. Room selection is naturally limited to rooms included in the lecturer's room preferences. This selection begins by randomly choosing a building. Afterward, a course room is randomly selected from the chosen building. The selected building and room are stored in the course meeting with the keys "lantai_terpilih" (selected building) and "ruang_terpilih" (selected room).

   Meanwhile, the selection of the starting block uses the index list that has been created, as shown in Figure 3. This selection of time and room aims to avoid selecting times or rooms that violate constraints or are outside the lecturer's room preferences. After a course meeting has been assigned a time and room, changes will occur in the time availability and course list array. An illustration of the changes that occur in each selection of time and room for each course meeting can be seen in Figure 5.

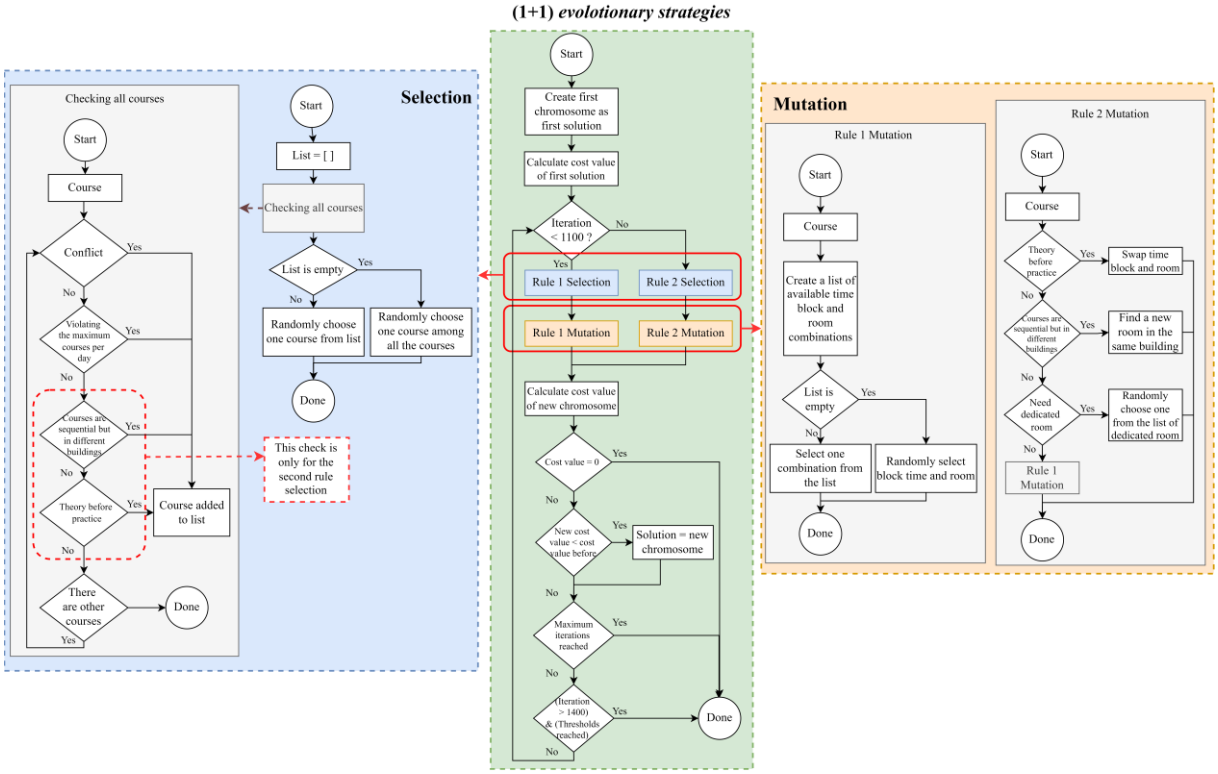


**Figure. 4**. Flowchart (1+1) Evolutionary strategies implementation.

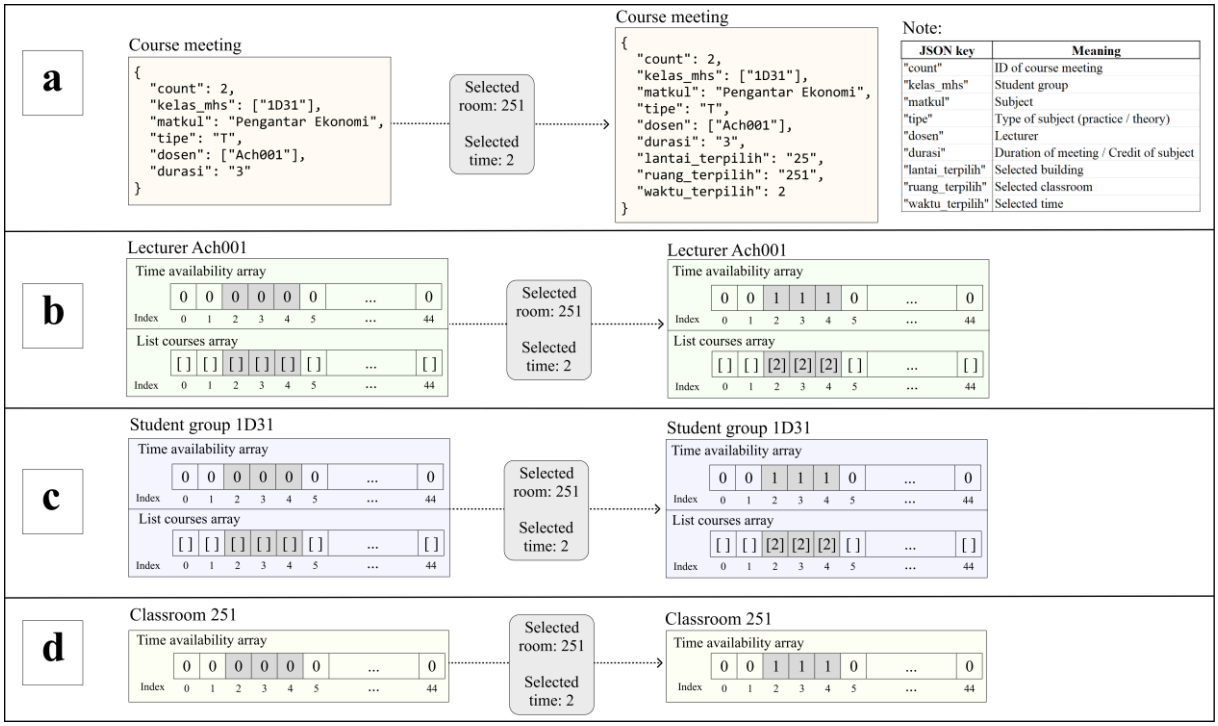


**Figure. 5**. Modifications to the scheduled time and room for course meetings that result in alterations to the array of: (a) course meetings, (b) lecturers, (c) student groups, and (d) classrooms. The alternations are visually represented by a grey background.

2. Cost assessment

The cost value is calculated using the following cost function:

$$Cost(chromosome) = \sum_{i=1}^{R} C_i w_i \quad (4)$$

The variable $R$ signifies the total quantity of constraints. At the same time, $C_i$ denotes the number of times the $i$-th constraint is violated, and $w_i$ represents the penalty (weight) associated with each violation of the $i$-th constraint. Based on equation (4), a lower cost value implies a more optimal timetabling solution.

The preparation of previous data has made the cost calculation easier. The cost value will increase as violations of certain constraints occur. Due to the well-designed data preparation and the selection of time and room for each course meeting that avoids prohibited times, violations of user-defined hard constraints or violations of the lecturer's time and room preferences will never occur. The remaining possible violations are course conflicts, the maximum number of courses in one day, and soft constraint violations.

The time availability arrays of lecturers, students, and classrooms can be checked to identify conflicting timetables. If any value in the time availability array is within the range of 2-98, there is a course conflict. The course list array is examined to determine the number of courses in one day. If there are more course IDs in one day than the maximum number of courses allowed in one day, then there is a violation of the maximum number of courses in one day. Finally, the time availability arrays of lecturers, students, and classrooms are checked to identify soft constraint violations. If a value in the array is more significant than 999, it means there is a soft constraint violation.

The constraint weights used in this study refer to [9] with some modifications. In addition, the weights for user-defined constraints can be adjusted as needed. A higher weight indicates that a constraint has a higher priority than another constraint with a lower weight. If the user does not define the constraint weight, the value will be set to the default weight. Table 2 details the list of constraints and their default weights used in this study.

**Table 2.** Default weight of each constraint

| Violation | Type | Default weight |
|---|---|---|
| Timetabling conflicts among lecturers, students, or classrooms | Hard | 1.00 |
| Violation of lecturer’s time preferences | Hard | 1.00 |
| Violation of lecturer’s room preferences | Hard | 1.00 |
| Maximum number of courses per day for a lecturer or student | Hard | 1.00 |
| | Soft | 0.02 |
| Theoretical courses precede practical courses. | Hard / Soft | 0.02 |
| Other violation hard constraints | Hard | 1.00 |
| Other violation soft constraints | Soft | 0.02 |

3. Selection

The selection process uses two different rules, but both begin by creating a list of courses that need improvement. The first rule is used when the iteration is still below 1100 because it focuses more on handling conflicting courses. This study selects the number 1100 because, after several trials using real course data from two semesters, the selection and mutation rules following the first rule no longer produce better chromosomes after 1100 iterations.

In the selection process using the first rule, a course is categorized as a course that needs to be improved when there is a conflicting course or when there is a violation of the number of courses in a single day, while the selection process using the second rule also considers theory courses that precede practical sessions and if there are consecutive courses but in different buildings from the lecturer's perspective. This checking process is done by looking at the time availability array and the course list array. From the created list, one course will be randomly selected for mutation. If the list is empty, one will be randomly selected from all courses.

4. Mutation

Mutation is done by changing the time and/or room of the selected course in the selection process. The mutation process also has two different rules, but both start by removing the relationship between the time availability array and the course list array. After that, a list of

combinations of available time blocks and rooms will be created. One of the combinations from the list created will be used as a replacement for the time and/or room of the course. If no combinations meet the requirements, then the time and room of the course will be randomly selected again and separately.

In mutations with the second rule, after the relationship between the course, the time availability array and the course list array is removed, several checks will be performed on the course. If this course has a pair (theory-practice) and violates the theory preceding practice, it will be swapped in time and room with its pair. If this does not happen, the check continues by looking at whether it is a consecutive course in a different building from the lecturer's point of view. If the course meets this condition, a new room in the same building as the previous course will be randomly selected. The next check for the course is whether the subject is a subject that must be placed in a specific room only. If it falls under this condition, then the course meeting will be assigned one of the rooms required to be occupied by this subject. The course-selected time will also be updated by selecting one of the block times where the course can start for the lecturer concerned (visualized in Figure 3). If the course passes all checks, the mutation that will be applied is the mutation with the first rule.

After the new time and room have been determined, the last step of the mutation with the first and second rules is to connect them to the time availability and course list array.

5. Last checking

The next step of implementation of (1+1)-ES is checking the threshold and iteration. The threshold is used to determine whether the algorithm can produce better chromosomes during a number of iterations. The algorithm will stop if the iteration exceeds 1400 and the threshold is reached. However, the algorithm will continue running if this condition is not met. This study selects the number 1400 because treatment with the second rule selection and mutation is expected to be carried out at least 300 times (from iteration 1100-1400). If, after the maximum iteration is reached, there are still violations, these violations will be reported to the user.

## *3.3. Multiprocessing*

In this research, multiprocessing is employed to create multiple timetable trials simultaneously. The configuration data determines the number of trials. For instance, only two timetables will be generated in parallel if the configuration specifies two trials. The service is limited to 8 CPU cores, so the maximum number of trials is capped at 8. The results of each trial are compared based on their cost values, and the trial with the lowest cost is selected as the final solution.

The 'concurrent.futures' library in Python does the multiprocessing. The library provides a high-level interface for asynchronously executing callables using the 'ThreadPoolExectutor' class or 'ProcessPoolExecutor' class. This study uses the second class.

When multiprocessing is activated, several processes run independently, indicated by high CPU usage on multiple cores. To verify this, we can examine the system monitor. Figure 6 presents a screenshot of the system monitor during multiprocessing.

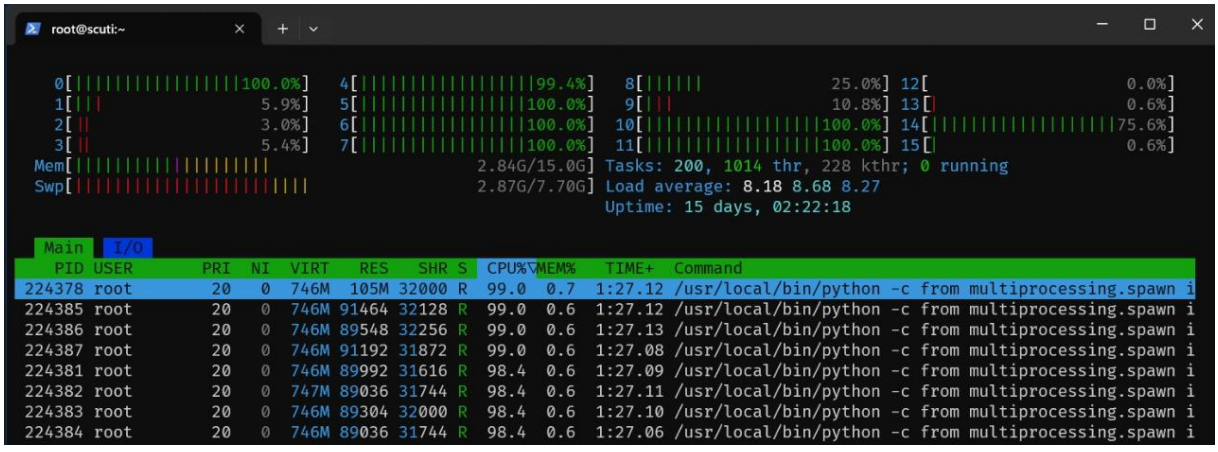


**Figure. 6**. Server system monitoring while multiprocessing is working

## *3.4. RESTful API*

No specific requirements exist for creating a timetable algorithm as a Representational State Transfer (RESTful) Application Programming Interface (API). The algorithm needs to be written using a programming language that supports the development of RESTful API (native or using library/framework). Utilizing the FastAPI framework facilitates the development of a RESTful API in this research. FastAPI is a modern web framework designed to streamline the creation of APIs in Python. By harnessing the power of Python 3.8+ and the asynchronous capabilities of Uvicorn, FastAPI offers developers a robust, efficient, and the best tool for building high-performance web applications [21], [22]. Benchmark comparisons conducted by TechEmpower have placed FastAPI among the top performers, showcasing its ability to rival the speed and efficiency of Node.js and Go, with Starlette and Uvicorn being the only frameworks to achieve slightly higher benchmarks. Moreover, FastAPI provides

The process commences with the instantiation of an object employing the "fastapi" library. The instantiated object possesses methods that are congruent with the Hypertext Transfer Protocol (HTTP), such as `get()`, `post()`, `put()`, `delete()`, and others. Each method necessitates a string-typed argument that will subsequently represent the endpoint of the constructed RESTful API. After declaring an endpoint, a function must be defined to process any request directed toward the specified endpoint.

This research has developed three endpoints with specific functions. Table 3 shows a detailed list of the implemented endpoints.

**Table 3.** Endpoint list

| Endpoint | Method | Utility |
|---|---|---|
| `/generatemultiproc` | POST | Create course timetable |
| `/validate` | POST | Course data validation |
| `/update-perkuliahan` | POST | Check timetable results |

Despite serving distinct utilities, the endpoints `/generatemultiproc` and `/validate` necessitate an identical request body. These endpoints enforce that each incoming request encapsulates the resources essential for timetabling within the request body. The resources related to the data required for (1+1)-ES, as described in the data preparation (Section 3.1). The enclosed request body must conform to a comprehensible format for the system. A detailed encoding for constraint definition is provided in Table 4.

**Table 4.** Custom Constraint Specification within the Request Body

| Key name | Note | Possible value | Value representation |
|---|---|---|---|
| Keterangan | Information about constraint | Free | - |
| jenis_target_1 | Target constraint | 1<br>2<br>3<br>4<br>5 | Lecture<br>Student<br>Subject<br>Classroom<br>All |
| jenis_constraint | Constraint types | 1<br>2<br>3 | Forbidden block time<br>Mandatory block time<br>Mandatory classroom |
| prioritas | Priority | 1<br>2 | Hard constraint<br>Soft constraint |
| is_all | Flag to signify universal applicability | True or False | - |
| data_1 | Constrained object | According to jenis_target_1 | - |
| daftar_blok | Constrained block time | 0-44 | - |
| daftar_ruang | Constrained classroom | Classroom | - |
| bobot | Weight constraint | 0-1 | - |

Examples of request body for both endpoints can be seen in the following code snippet.

```
{
 "ruang_kelas": {
  "25": ["251", "252", "253", "254", "255", "256", "257"],,
  ...
 },
 "dosen": [
  {
  "nama": "Ach001",
  "nama_lengkap": "Nama Dosen Ach001",
  "pref_waktu": [
   0, 0, 0, 0, 0, 0, 0, 0, 0, 99, 99, 99, 99, 99, 99, 99, 99, 99, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  ],
  "pref_ruangan": {
    "25": ["251", "252", "253", "254", "255", "256", "257"]
  }
 },
  ...
 ],
 "perkuliahan": [
  {
   "count": 2,
   "kelas_mhs": ["1D31"],
   "matkul": "Pengantar Ekonomi",
   "tipe": "T",
   "dosen": ["Ach001"],
   "durasi": "3"
  },
  ...
 ],
 "constraints": [
  {
   "keterangan": "Jum'atan",
   "jenis_target_1": 5,
   "jenis_constraint": 1,
   "prioritas": 1,
   "is_all": true,
   "data_1": [],
   "daftar_blok": [39, 40, 41]
  },
  ...
  ],
  "config": {
    "iterasi": 2000,
    "maks_matkul": 2,
    "maks_matkul_prioritas": 1,
    "jml_percobaan": 8,
    "jml_kandidat_solusi": 3,
    "blok_mulai_2_sks": [1, 3, 5, 7],
    "blok_mulai_3_sks": [1, 3, 4, 7]
  }
}
```

For the `/generatemultiproc` endpoint, a valid request body will be forwarded for data processing (data preparation) before finally entering the (1+1)-ES implementation. The response generated by this endpoint is a list of courses with assigned times and rooms, a list of violations, if any, a list of costs generated during the algorithm execution, a list of candidate course timetable solutions, and additional information such as an array of available times and an array of course lists. Meanwhile, for the `/validate` endpoint, a valid request body will only undergo data validation. If the data can be used to create a course timetable, a message indicating the data is ready to be used will appear in the response. Conversely, if the data cannot be used to create a course timetable, a list of error messages in the sent data will appear. The `/update-perkuliahan` endpoint requires a request body similar to the other two. The difference is that in the course definition, the "waktu_terpilih" (selected time) and "ruang_terpilih" (selected room) keys must be added. The response generated by this endpoint is the same as the `/generatemultiproc` endpoint.

## 3.5. Microservices architecture

The development of the course timetabling system at Politeknik Statistika STIS employs a microservices architecture. This architecture divides the system into four services, each with its specific function. These four services are the front end, back end, data storage, and timetabling service. The focus of this research is on the timetabling service. The microservices architecture and its relationships are shown in Figure 7.

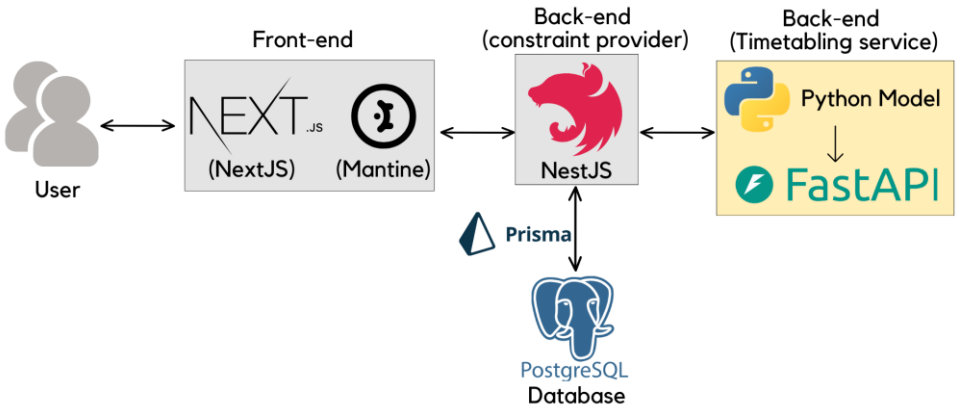


**Figure. 7**. Microservices architecture

The front-end module is a web page responsible for handling user interactions with the system. Users can interact directly through the webpage to modify timetabling data. Subsequently, this modification request will be sent from the front end to the back end for processing. Once the processing is complete, the back end will respond to the request. This response will be read by the front end and displayed to the user.

The back-end module is a service that interacts with all other services, including data storage. This module handles requests from the front end. If the request is for data modification, the back-end module will modify the data stored in the database using the ORM (Object Relational Mapping) technique and the Prisma framework. However, if the front-end requests to create a timetable, the back-end module will prepare course data from the database and send a request to the timetabling service with the course data as the request body. The response from the timetabling service will be stored in the database and returned to the front-end module.

This research focuses on the timetabling service. This service is built using the Python programming language and utilizes the FastAPI framework. It generates course timetables using the resources sent in the request body. In creating course timetables, the (1+1)-ES implementation is carried out within the service. The resulting timetable, along with any constraint violations, will be sent back to the back-end module.

## 3.6. Evaluation

As previously explained in the methodology section, the evaluation in this research is conducted through two tests: the cost-value to determine the goodness of the solution generated by the (1+1)-ES implementation and the black box testing to test the RESTful API. All tests were conducted on the service running on the server. The server specifications are as follows:

1) CPU: Intel Xeon 16 cores @ 2.2 GHz
2) RAM: 16GB
3) Storage: 92GB

### 3.6.1. Cost Value

The course data used in the testing is the course data of Politeknik Statistika STIS for the odd and even semesters of the academic year 2023/2024. The details of the data are presented in Table 5.

The course data was used in an experiment to generate course timetables 1000 times, with each experiment containing 8 different solutions using multiprocessing and aiming for 3 candidate solutions.

The maximum number of iterations used in the (1+1)-ES was 2000, and the cost value of each experiment was calculated.

**Table 5.** Course data of Politeknik Statistika STIS of the academic year 2023/2024

| Note | Semester | |
|---|---|---|
| | Odd | Even |
| Number of student groups | 60 | 50 |
| Number of lecturers | 103 | 88 |
| Number of classrooms | 41 | 41 |
| Number of course meetings | 328 | 281 |
| (2 credit) | 39 | 23 |
| (3 credit) | 289 | 258 |
| Number of constraints defined by user (custom constraints) | 9 | 9 |

a. Results of the cost value using odd semester course data

The estimated time to generate the odd semester course timetable is 110-160 seconds, averaging 130 seconds (2 minutes and 10 seconds). Table 6 reveals that most of the generated timetables have a cost value between 4 and 5. Given that a hard constraint violation incurs a cost of 1, it can be inferred that most timetables contain hard constraint breaches, specifically four timetable conflicts. However, the (1+1)-ES implementation in this study is designed to strictly adhere to user-defined hard constraints such as forbidden and mandatory blocks. Consequently, the potential violations in the generated timetables are limited to timetable conflicts, exceeding the daily course limit, and soft constraint breaches. Thus, most generated odd semester timetables likely contain four hard constraint violations (timetable conflicts), and the remaining violations are soft constraint breaches. Despite these minor violations, the relatively small cost values compared to the initial iterations suggest that the (1+1)-ES implementation for course timetabling is effective.

**Table 6.** Percentage distribution of cost values for the odd semester course data.

| Range of cost values | Amount | Percentage |
|---|---|---|
| $3 \leq Cost < 4$ | 198 | 19.8 % |
| $4 \leq Cost < 5$ | 405 | 40.5 % |
| $5 \leq Cost < 6$ | 254 | 25.4 % |
| $6 \leq Cost < 7$ | 109 | 10.9 % |
| $7 \leq Cost < 8$ | 29 | 2.9 % |
| $8 \leq Cost < 9$ | 4 | 0.4 % |
| $9 \leq Cost < 10$ | 1 | 0.1 % |
| Total | 1000 | 100.0 % |

Furthermore, Table 6 indicates that the minimum cost achieved from 1000 timetabling trials is 3. This outcome seems anomalous as the expected minimum cost should be 0. Upon investigation, this anomaly was attributed to three lecturers having ample time availability for their assigned courses. As a result, every generated timetable inevitably violates the daily course limit for these three lecturers. Despite the maximum daily course limit being set at 2, combining these lecturers' time preferences and associated constraints necessitates at least one day with more than two courses.

b. Results of the cost value using even semester course data

In contrast to the odd semester data, the even semester course timetabling experiments yielded significantly better results. Of the 1000 trials conducted, most generated timetables exhibited only two types of violations: consecutive courses in different rooms or buildings and deviations from lecturers' "medium" preferred time blocks. As these violations pertain to soft constraints, the (1+1)-ES algorithm could produce even semester timetables with a minimum cost below 1. Moreover, the average timetabling time was reduced to 110 seconds (1 minute and 50 seconds). These findings strongly suggest

that the (1+1)-ES implementation for timetabling courses at the Politeknik Statistika STIS has been highly successful. A detailed breakdown of cost values is presented in Table 7.

**Table 7.** Percentage of cost values for the even semester course data.

| Cost values | Amount | Percentage |
|---|---|---|
| Under 1 | 978 | 97.8 % |
| 0.02 | 594 | 59.4 % |
| 0.04 | 211 | 21.1 % |
| 0.06 | 134 | 13.4 % |
| 0.08 | 26 | 2.6 % |
| 0.1 | 6 | 0.6 % |
| 0.12 | 7 | 0.7 % |
| Above 1 | 22 | 2.2 % |
| 1 | 6 | 0.6 % |
| 1.02 | 6 | 0.6 % |
| 1.04 | 1 | 0.1 % |
| 1.06 | 5 | 0.5 % |
| 1.08 | 3 | 0.3 % |
| 2.04 | 1 | 0.1 % |
| Total | 1000 | 100.0 % |

### *3.6.2. Black Box Testing*

Each endpoint underwent black box testing, tailored to its specific request body. Postman, an API platform that streamlines the entire API lifecycle, was employed for these tests. The scenarios and outcomes of the RESTful API black box testing are summarized in Table 8. All test success indicates that the (1+1)-ES has been successfully implemented as a RESTful API.

**Table 8.** Scenarios and results of black box testing

| Tested endpoint | Test scenario | Expected output | Result |
|---|---|---|---|
| `/generatemultiproc` | Assigning values to the JSON attributes of classroom, lecturer, course, constraint, and config. | A course list is presented, including scheduled time and room, violation records, additional information, and a cost value history | Success |
| `/validate` | Assigning values to the JSON attributes of classroom, lecturer, course, constraint, and config. | A JSON response is generated with a "status" field set to "success" and a message confirming that the course data is prepared for creation. | Success |
| `/update-perkuliahan` | Assigning values to the JSON fields of classroom, lecturer, course, constraint, and config. Additionally, the course field is populated with both the selected time and the chosen room. | A course list is presented, including scheduled time and room, violation records, additional information, and a cost-value history | Success |

## 4. Conclusion

A (1+1) evolutionary strategy, a specific type of evolutionary algorithm, has been successfully implemented to generate course timetables for the Politeknik Statistika STIS. The service's adaptability

is improved through a user-configurable constraint system. Timetabling conflicts are identified and reported to the user for manual intervention. The algorithm's performance is deemed satisfactory given an average near-zero cost for the even semester data. The slight variation in the lowest achievable cost between semesters suggests that the specific characteristics of the course data have a notable impact on the optimization results. However, the evolutionary algorithm is a standard approach. Currently, there are many advancements in evolutionary algorithms that future researchers can implement into a service.

The RESTful API service has been successfully implemented and is functioning as intended. Black box testing of each endpoint has confirmed that the service is producing the correct outputs for all defined test cases. This finding means that (1+1)-ES has been successfully implemented as a RESTful API.

## Ethics approval

Not Required

## Acknowledgments

We would like to express our deepest gratitude to all parties who have helped in this research. Special thanks to Farid Ridho, SST., MT., who has provided many new insights and guidance during the research.

## Competing interests

All the authors declare that there are no conflicts of interest.

## Funding

This study received no external funding.

## Underlying data

Derived data supporting the findings of this study are available from the corresponding author on request.

## Credit Authorship

**Zuhdi Ali Hisyam:** Conceptualization, Software Development, Testing, Writing- original draft. **Farid Ridho:** Methodology, System Architecture, writing-review and editing, supervision. **Arbi Setiyawan:** Writing-review and editing, supervision

# References

[1] A. Bashab *et al.*, “Optimization Techniques in University Timetabling Problem: Constraints, Methodologies, Benchmarks, and Open Issues,” *Computers, Materials and Continua*, vol. 74, no. 3, pp. 6461–6484, 2023, doi: 10.32604/cmc.2023.034051.

[2] Politeknik Statistika STIS, “A Brief History of the Politeknik Statistika STIS”, (in Indonesian), Accessed: May 18, 2024. [Online]. Available: https://stis.ac.id/hal/16/sejarah-singkat

[3] C. H. Wong, S. L. Goh, and J. Likoh, “A Genetic Algorithm for the Real-world University Course Timetabling Problem,” in *2022 IEEE 18th International Colloquium on Signal Processing and Applications, CSPA 2022 - Proceeding*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 46–50. doi: 10.1109/CSPA55076.2022.9781907.

[4] M. V. Rane, V. M. Apte, V. N. Nerkar, M. R. Edinburgh, and K. Y. Rajput, "Automated timetabling system for university course," in *2021 International Conference on Emerging Smart Computing and Informatics, ESCI 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 328–334. doi: 10.1109/ESCI50559.2021.9396906.

[5] M. Zunino, S. V. del Valle, and L. Gatti, "An Automated Approach to University Course Timetabling Focused on Professor Assignment," in *2024 L Latin American Computer Conference (CLEI)*, IEEE, Aug. 2024, pp. 1–4. doi: 10.1109/CLEI64178.2024.10700361.

[6] Pradnya A. Vikhar, "Evolutionary Algorithms: A Critical Review and its Future Prospects," *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication*, 2016, doi: 10.1109/ICGTSPICC.2016.7955308.

[7] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," Aug. 01, 2020, *Springer*. doi: 10.1007/s00521-020-04832-8.

[8] T. Chugh, K. Sindhya, J. Hakanen, and K. Miettinen, "A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms," *Soft comput*, vol. 23, no. 9, pp. 3137–3166, May 2019, doi: 10.1007/s00500-017-2965-0.

[9] C. Fernandes, J. P. Caldeira, F. Melicio, and A. Rosa, "HIGH SCHOOL WEEKLY TIMETABLING BY EVOLUTIONARY ALGORITHMS," *Proceedings of the 1999 ACM symposium on Applied computing*, pp. 344–350, 1999, doi: 10.1145/298151.298379.

[10] I. A. Abduljabbar and S. M. Abdullah, "An evolutionary algorithm for solving academic courses timetable scheduling problem," *Baghdad Science Journal*, vol. 19, no. 2, pp. 399–408, 2022, doi: 10.21123/BSJ.2022.19.2.0399.

[11] S. Choudhary, S. Janarthanan, and P. Maurya, "A Study and Analysis of Timetable Generation using a Genetic Algorithm," in *Proceedings - IEEE 2023 5th International Conference on Advances in Computing, Communication Control and Networking, ICAC3N 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 700–703. doi: 10.1109/ICAC3N60023.2023.10541761.

[12] A. Schaerf, "A Survey of Automated Timetabling," *Artif Intell Rev*, vol. 13, pp. 87–127, 1999, doi: 10.1023/A:1006576209967.

[13] H. Algethami and W. Laesanklang, "A mathematical model for course timetabling problem with faculty-course assignment constraints," *IEEE Access*, vol. 9, pp. 111666–111682, 2021, doi: 10.1109/ACCESS.2021.3103495.

[14] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, "Evolutionary Algorithms," *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 4, no. 3, pp. 178–195, 2014, doi: 10.1002/widm.1124.

[15] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000.

[16] I. Ahmad, E. Suwarni, R. I. Borman, Asmawati, F. Rossi, and Y. Jusman, "Implementation of RESTful API Web Services Architecture in Takeaway Application Development," in *2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System, ICE3IS 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 132–137. doi: 10.1109/ICE3IS54102.2021.9649679.

[17] A. Soni and V. Ranga, "API features individualizing of web services: REST and SOAP," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 9 Special Issue, pp. 664–671, Jul. 2019, doi: 10.35940/ijitee.I1107.0789S19.

[18] Z. A. Hamza and M. Hammad, "Web and Mobile Applications' Testing using Black and White Box approaches," in *2nd Smart Cities Symposium (SCS 2019)*, 2019, pp. 1–4. doi: 10.1049/cp.2019.0210.

[19] D. Liu, C. Y. Li, Z. Jiang, R. Kong, L. Wu, and C. Ma, "Integrated Power Grid Management System based on Micro Service," in *Proceedings - 2020 8th International Conference on Advanced Cloud and Big Data, CBD 2020*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 37–41. doi: 10.1109/CBD51900.2020.00016.

[20] M. Söylemez, B. Tekinerdogan, and A. K. Tarhan, “Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review,” Jun. 01, 2022, *MDPI*. doi: 10.3390/app12115507.

[21] S. Ramírez, “FastAPI Documentation.” Accessed: Oct. 26, 2024. [Online]. Available: https://fastapi.tiangolo.com

[22] S. J. C. TRAGURA, *BUILDING PYTHON MICROSERVICES WITH FASTAPI build secure, scalable, and structured Python microservices from design concepts to infrastructure*. PACKT PUBLISHING LIMITED, 2022.