Journal of Applied Informatics and Computing (JAIC)
Vol.5, No.2, Desember 2021, pp. 202~207
e-ISSN: 2548-6861

Analysis of Security Guard Scheduling System Using Genetic Algorithm and Tournament Selection (Case Study: Institut Teknologi Sumatera)

Ilham Firman Ashari [1]*, Ardi Gaya Manalu [2]*, Rahmat Setiawan[3]*,
Mugi Praseptiawan[4]*, Dita Alviuni P[5]*, Sisilia Juli A [6]*
* Teknik Informatika, Insitut Teknologi Sumatera
firman.ashari@if.itera.ac.id [1], ardi.118140088@student.itera.ac.id [2], rahmat.118140088@student.itera.ac.id [3],
mugi.prasetptiawan@if.itera.ac.id [4], dita.118140168@student.itera.ac.id [5], sisilia.118140088@student.itera.ac.id [6]

Article Info

Article history:

Received 2021-10-02
Revised 2021-11-24
Accepted 2021-12-04

Keyword:

Schedule,
Genetic,
Algorithm, Tournament

ABSTRACT

Institut Teknologi Sumatera (ITERA) is one of the new state universities on the island of Sumatra. ITERA has developed and has a large area and many buildings, of course it requires a lot of security guards to maintain security and order in the campus environment. The working hours of security guards at ITERA are from morning to night. ITERA is required to make a watch or shift schedule for security guards, where currently the scheduling is still done manually and has not been systemized automatically. Another problem caused is that the addition of buildings and security guards will make future scheduling more difficult, ineffective, and efficient. Genetic Algorithm can be used in the scheduling process automatically and optimally by going through several stages. The security guard data used are 21 people and the number of buildings is 6 buildings. The result of the research is that the automatic scheduling system was successfully built, from the security data used as many as 16 data obtained scheduling from Monday to Sunday along with its working hours.

I. INTRODUCTION

Information technology is used to make it easier for humans to get information, such as tourism information, scheduling information, and educational information [1]. To make it easier for humans to get information, a system is made which provides information in accordance with the desired content. One of the information that can be obtained is information related to scheduling. Scheduling is a series of plans that regulate the allocation of work time to the division of tasks from limited resources based on a certain time span [2]. The division of tasks is usually done by considering the capacity of the available resources to be able to do a job and the efforts to make optimal decisions for an organization.

Institut Teknologi Sumatera (ITERA) is one of the state universities on the island of Sumatra, precisely in the province of Lampung. On a state campus that is currently growing and has a large area and many buildings, this certainly requires a lot of security personnel to maintain security and order in the campus environment. Currently, the total buildings in ITERA that are used to support the activities of the academic community are buildings A, B, C, D, E, F, and kuliah umum building (GKU) on a land area of approximately 300 hectares. Security guards also must be on guarding day and night, therefore ITERA is obliged to make a watch schedule or shift for security guards. ITERA itself already has a schedule for security guards, but it is still done manually and there is no automatic scheduling system. This scheduling is certainly not effective and efficient, considering the number of buildings continues to grow and the number of security guards is also increasing, therefore it is necessary to schedule using an automatic system so that it can accommodate this problem.

The algorithms commonly used for scheduling optimization are the tabu search algorithm, ant colony, and genetic algorithm [3][4]. Based on research conducted by Tiandini et al, it was found that the genetic algorithm is more optimal than the taboo search algorithm [4]. The advantage of genetic algorithms is that they can perform optimization for

more complex problems with a larger space. The problems that can be solved by the taboo algorithm are limited, and mostly just combination problems. Where in terms of time or complexity the taboo algorithm is faster, while the genetic algorithm takes longer computational time due to the need for regeneration. In addition to the taboo algorithm, you have to think about the size of the taboo list [4].

From research conducted by Kurniati, et al using the ant colony algorithm, it was found that scheduling using ant colony produces a smaller fitness value than scheduling using genetic algorithms. So scheduling with genetic algorithms is more optimum than ant colony [5]. The next related research is from Razali and Geraghty, where they compare the selection method on genetic algorithms [6]. The selection methods compared are roulette wheel, tournament, and rank. From the test results, it is found that the tournament selection method has better efficiency for the solution obtained because it has fewer generations and better iteration times [6].

Therefore, the researchers decided to choose a genetic algorithm for scheduling optimization accompanied by the tournament selection method. Several studies related to scheduling optimization with genetic algorithms have been carried out by several researchers including Dwi Oktarina et al, Restie Maya et al, Rifqy Rosyidah, and Tri Handoyo et al.

Based on research conducted by Dwi Oktarina and Alyauma Hajjah, it was found that the proposal scheduling system and thesis trial seminar can speed up the process of scheduling activities and information obtained from the web can make it easier for students and lecturers to find the proposed schedule and list of lecturer exam schedules [7]. By applying genetic algorithms to the seminar proposal scheduling system and thesis testing, errors and process delays can be minimized.

Another research conducted by Puspita, et al. They developed an application that can solve the problem of planning activities by generating the best plan for the activity. The use of genetic algorithm methods in application scheduling activities can simplify the scheduling process, if in the process a solution is given for the efficiency of scheduling data processing, be it time, energy or other resources, it can be said to be optimal, and there are no schedule errors with the same room [8].

Subsequent research conducted by Rifqy Rosyidah Ilmi, et al. In this study, genetic algorithms are applied to solve problems related to nurse scheduling. Permutation representation of integers with a length of chromosome 360 that is used for each gene number represents the nurse id number [9]. The crossover method used is the one-point intersection, the reciprocal exchange mutation mutation method and selected by selection elitism. From the test results, the optimal parameters obtained are the population of 200 individuals with moderate fitness from 0.80094, 150 creates an average fitness of 2.13674 and the combination of cr = 0.5 and mr 0.5 with an average fitness of 3.4266. The end result is the scheduling of nurses in the ICU for 1 month [9].

Another research conducted by Tri Handoyo, related to the Lesson Scheduling System, at Muhammadiyah 1 Middle School, Magelang City with Genetic Algorithms. This web-based scheduling system is designed using Adobe Dreamweaver CC 2014 and MySQL database. From this research, it was found that lesson scheduling is more optimal, easy, and structured by using genetic algorithms [10].

A genetic algorithm is a technique for finding solutions using the principle of random (natural) selection. The genetic algorithm process begins with the selection of a set or data set to be used, this data will then be implemented in the form of a chromosome called a population [11]. Solutions contained in a population will be made to form a new population, where the selection depends on the physical value obtained. Therefore, the new population is expected to be better than the previous one. This process is repeated until certain conditions are met. The use of this algorithm method is used in order to find efficient scheduling and to simplify the process of scheduling security guards in every building in ITERA, which was previously still done computerized and then the system will be developed again automatically.

II. RESEARCH METHOD

In the study, several stages were carried out sequentially, so that the output of the research was obtained.

A. Data Collection and Observation

In this phase, the researcher conducts a survey and collects data on security guards who work in the ITERA environment. In this study using some data as a test sample. The data used in this study are the name of the security guard, the name of the building, working hours, and the schedule of working days. The name of the building is the place where the building will be monitored by the security guard (A, B, C, D, E, F, GKU), security guard hours (8.00-16.00), where each shift works for 1 hour. The security guard's work schedule is Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday (SLRKJBM). The security guard data can be seen in table 1 below.

Table 1. Data of Security Gurads at ITERA

No	Security Guard Names	Building Name	Work Hours	Working Day
1	Eko	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
2	Tommy	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
3	Haikal	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
4	Michael	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
5	Budi	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
6	Krisna	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
7	Joko	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
8	Robby	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM

No	Security Guard Names	Building Name	Work Hours	Working Day
9	Putra	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
10	Bima	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
11	Jono	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
12	Tober	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
13	Daffa	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
14	Suryanto	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
15	Alex	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
16	Yogi	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
17	Muharram	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
18	Dodi	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
19	Rusdi	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
20	Irawan	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM
21	Lutfi	Building A, B, C, D, E, F, GKU	8-16	SLRKJBM

B. Selection of Methods

After getting the data, a method is chosen, namely using a genetic algorithm. The flow of the genetic algorithm can be seen in the following figure [12].

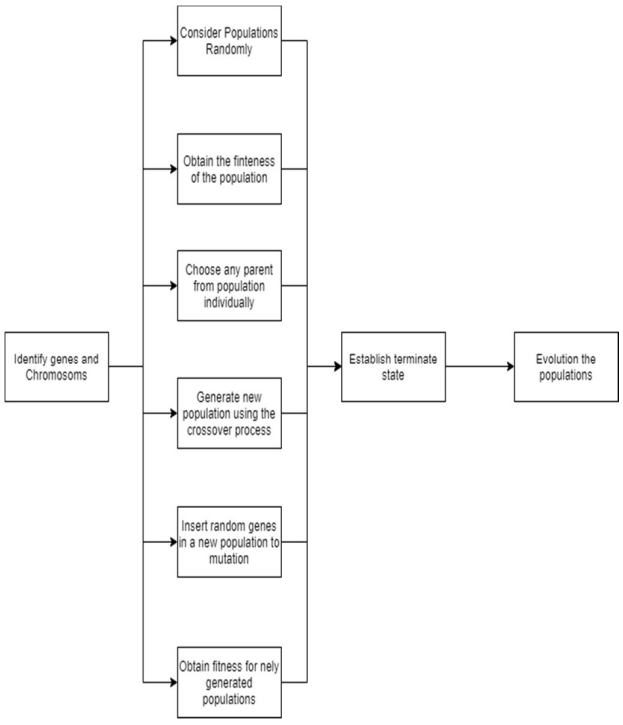

Figure 1. Genetic Algorithm

The stages to be carried out are cycles of the following phases:

1. Determination of the initial population

Determination of the initial population based on the security guard table data in table 1. So, there is the name of the security guard, the building, the hours of guard and the day of guard. Where the rules for working days and hours have been determined, for the hours of 8 am to 4 pm and working days Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday. In mapping individuals to populations using binary data.

2. Fitness Evaluation

Where in this evaluation section a maximum score will be determined to determine the best individual quality. The variables used are the name of the security guard, the building, and the working time. To calculate the fitness value can be seen in formula 1.

$$score = satpam_{inc} * num_{satpam} + gedung_{inc} * 2 * num_{satpam} + 1 \quad (1)$$

3. Selection

The selection function is used to determine the best parent candidate from the population. This is by looking at the value of the fitness evaluation that has been done. The selection used in this study is to use the tournament selection technique. Tournament selection is done by randomly selecting n individuals in the population with the highest fitness value in the population, where these individuals will compete with one another. This method can be seen as in the following picture [6].

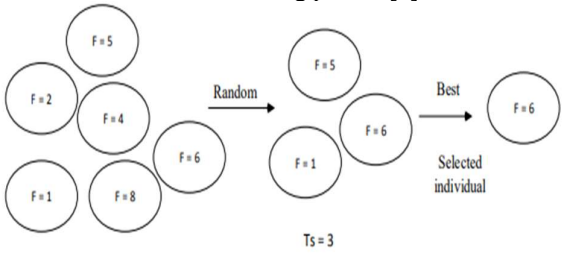

Figure 2. Tournament Selection Method

4. Reproduction

In this section, reproduction will be carried out by means of crossover and mutation. Mutations are carried out by exchanging gene values with inverse values. Mutations used are binary mutations. The crossover used is a two point crossover.

C. Implementation

The program is made using the Python programming language using Google Collaboration tools in the implementation. At the implementation stage, there is no GUI display.

D. *Analysis and Evaluation of Results*

After completing the program, testing is carried out by running the program to find out whether the results are appropriate or whether there are errors in the program flow that has been designed.

III. Result And Analysis

A. Result

Security guard scheduling is made within a week's time span by considering the attributes that include the name of the security guard, the name of the building, the duration, and the working days of the week. The results of scheduling processing can be seen in Figure 3.

```
mutasi complete
[ Eko, Gedung E, dt{16, SRJ},
  Tommy, Gedung GKU, dt{14, LK},
  Haikal, Gedung C, dt{8, SRJ},
  Michael, Gedung C, dt{12, BM},
  Budi, Gedung GKU, dt{15, SRJ},
  Krisna, Gedung GKU, dt{8, BM},
  Joko, Gedung F, dt{13, BM},
  Robby, Gedung B, dt{13, BM},
  Putra, Gedung C, dt{15, SRJ},
  Bima, Gedung B, dt{11, LK},
  Jono, Gedung GKU, dt{9, LK},
  Tober, Gedung GKU, dt{8, SRJ},
  Daffa, Gedung GKU, dt{15, BM},
  Suryanto, Gedung A, dt{10, SRJ},
  Alex, Gedung C, dt{10, BM},
  Yogi, Gedung F, dt{15, BM},
  Muharram, Gedung A, dt{8, BM},
  Dodi, Gedung E, dt{8, SRJ},
  Rusdi, Gedung F, dt{11, BM},
  Irawan, Gedung A, dt{14, SRJ},
  Lutfi, Gedung A, dt{11, LK}]
```

Figure 3. The most optimum scheduling results

From Figure 3, information is obtained for the name of the security guard, work building, working time and working day, with details of the binary mutation results in Figure 4.

```
{ 'Gedung A': { 'B': [1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                'J': [0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
                'K': [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
                'L': [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
                'M': [1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                'R': [0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
                'S': [0, 0, 1, 0, 0, 0, 1, 0, 0, 0]},
  'Gedung B': { 'B': [0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
                'J': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'K': [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
                'L': [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
                'M': [0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
                'R': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'S': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]},
  'Gedung C': { 'B': [0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
                'J': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                'K': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'L': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'M': [0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
                'R': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                'S': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0]},
  'Gedung E': { 'B': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'J': [1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                'K': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'L': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'M': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'R': [1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                'S': [1, 0, 0, 0, 0, 0, 0, 0, 1, 0]},
  'Gedung F': { 'B': [0, 0, 0, 1, 1, 1, 1, 1, 1, 0],
                'J': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'K': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'L': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'M': [0, 0, 0, 1, 1, 1, 1, 1, 1, 0],
                'R': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                'S': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]},
  'Gedung GKU': { 'B': [1, 1, 0, 0, 0, 0, 0, 1, 1, 0],
                  'J': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                  'K': [0, 1, 1, 0, 0, 0, 1, 1, 0, 0],
                  'L': [0, 1, 1, 0, 0, 0, 1, 1, 0, 0],
                  'M': [1, 1, 0, 0, 0, 0, 0, 1, 1, 0],
                  'R': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                  'S': [1, 0, 0, 0, 0, 0, 0, 1, 0, 0]}}
```

Figure 4. Security guard scheduling results with binary mutation

It can be concluded from the results, all security guards have been properly scheduled from Monday to Sunday.

B. The Analysis of Algorithm

At this stage, the security scheduling mechanism will be explained one by one using a genetic algorithm, starting from the variable initialization stage, determining the fitness function, selection, mutation, and cross over. An initial stage is needed which consists of 2 classes, namely the datetime and shift classes. This can be seen in Figure 5.

```
class DateTime:
    days_list = ["SRJ", "LK", "BM"]
    duration_mapping = {"SRJ":50 , "LK": 70, "BM": 70}

    def __init__(self, start_time, days):
        self.start_time = start_time
        self.days = days
        self.duration = DateTime.duration_mapping.get(self.days)

    def __repr__(self):
        return "dt{%d, %s}" % (self.start_time, self.days)


class shift:

    def __init__(self, satpam_id, gedung_num, date_time):
        self.satpam_id = satpam_id
        self.gedung_num = gedung_num
        self.date_time = date_time

    def mutasi_satpam_id(self, satpam_id):
        self.satpam_id = satpam_id

    def mutasi_gedung(self, gedung_num):
        self.gedung_num = gedung_num

    def mutasi_date_time(self, date_time):
        self.date_time = date_time

    def __repr__(self):
        return "%s, %s, %s" % (self.satpam_id, self.gedung_num, self.date_time)
```

Figure 5. Class Initiation

In the first code block, there are 2 classes in the block. The first class is DateTime. Where in this class contains a list of days that will be used in scheduling. Then there is also duration_mapping which stores that those who work on Mondays, Wednesdays, Fridays work less than Tuesdays, Thursdays and Saturdays, Sundays. The shift class contains five functions that will be used in the permutation function in the second code block. The variables used for population formation, fitness evaluation, and selection can be seen in Figure 6.

```
import sys
import pprint
import copy
import random


nama_satpam = ["Alex", "Tommy", "Yogi", "Irawan", "Budi", "Jono", "Krisna", "Tober", "Putra", "Michael", "Suryanto",
          "Dodi", "Daffa", "Muharram", "Robby", "Bima", "Haikal", "Joko", "Rusdi", "Lutfi", "Eko"]

gedungITERA = ["Gedung A", "Gedung B", "Gedung C", "Gedung E", "Gedung F", "Gedung GKU"]

DAYS_OF_WEEK = "SLRKJBM"

GENOMES_SIZE = 5000

MIN_START_TIME = 8
MAX_START_TIME = 16

NUM_SATPAM = len(nama_satpam)
NUM_GEDUNG = len(gedungITERA)
TIME_SLICES = MAX_START_TIME - MIN_START_TIME + 2

SATPAM_INC = 2
GEDUNG_INC = 1

TOP_PERCENT = .05
PERCENT_UN_MUTATED = .01

random.seed()

pp = pprint.PrettyPrinter(indent=2)
```

Figure 6. Variable Initialitation

The nama_satpam variable is a variable that contains a collection of the names of working security guards, while the ITERA building variable is a variable that contains buildings on the ITERA campus. The variable DAYS_OF_WEEK is a variable that stores a string of the names of the days. S for Monday, L for Tuesday, R for Wednesday, K for Thursday, J for Friday, B for Saturday, and M for Sunday. The GENOMES_SIZE variable is a variable that stores how many genes will be produced at the selection stage to mutation, here 5000 genes are used. The MIN_START_TIME variable is a variable that stores the initial time the security guard works and the MAX_START_TIME variable is a variable that stores the last time the security guard works. Next, a process will be carried out to calculate the fitness value, which can be seen in Figure 7.

The function of evaluating the fitness value in this program is to determine whether an individual will be processed to the next stage. The fitness value obtained will be used as a reference in achieving the optimal value. If what you are looking for is the maximum value, then the value of fitness is the value of the function itself. But if the minimum value is required then the fitness value is an inverse of the value of the function itself.

```
def fitness(gene):
    global nama_satpam
    score = SATPAM_INC * NUM_SATPAM + GEDUNG_INC * 2 * NUM_SATPAM + 1
    hitung_satpam = {}
    for satpam_id in nama_satpam:
        hitung_satpam[satpam_id] = 0
    hitung_gedung = {}
    for gedung in gedungITERA:
        hitung_gedung[gedung] = {}
        for day in DAYS_OF_WEEK:
            hitung_gedung[gedung][day] = []
            for i in range(MIN_START_TIME, MAX_START_TIME + 2, 1):
                hitung_gedung[gedung][day].append(0)
    for satpam in gene:
        hitung_satpam[satpam.satpam_id] += 1
        for time_slice in range(satpam.date_time.start_time - MIN_START_TIME,
                                satpam.date_time.start_time + satpam.date_time.duration // 60 + 1 - MIN_START_TIME):
            for day in satpam.date_time.days:
                hitung_gedung[satpam.gedung_num][day][time_slice] += 1

    for hitung in hitung_satpam.values():
        if hitung == 1:
            score += SATPAM_INC
        else:
            score -= SATPAM_INC

    for days in hitung_gedung.values():
        for day in days.values():
            for hitung in day:
                if hitung != 0 and hitung != 1:
                    score -= GEDUNG_INC
    if score <= 0:
        print("score of %d not allowed, exiting..." % score)
        sys.exit(-1)
    return score
```

Figure 7. Evaluation Fitness Function

After the fitness evaluation process has been carried out, the next step is to select the tournament which can be seen in Figure 8.

```
def selection(genomes, scores):
    children = []
    # select top 10% to be part of children
    children.extend([genomes[i[0]] for i in scores[:(int(GENOMES_SIZE * TOP_PERCENT))]])

    scores_sum = sum([i[1] for i in scores])
    proportional_selection = []
    index = 0
    for i in scores:
        proportional_selection.append(range(index, index + i[1]))
        index = index + i[1]

    for i in range(0, GENOMES_SIZE - int(GENOMES_SIZE * TOP_PERCENT)):
        selection1 = random.randint(0, scores_sum - 1)
        selection2 = random.randint(0, scores_sum - 1)

        parent1 = None
        parent2 = None

        for k in range(len(proportional_selection)):
            if selection1 in proportional_selection[k]:
                parent1 = genomes[scores[k][0]]
            if selection2 in proportional_selection[k]:
                parent2 = genomes[scores[k][0]]
            if parent1 is not None and parent2 is not None:
                break

        point1 = random.randint(1, NUM_SATPAM - 2)
        point2 = random.randint(point1, NUM_SATPAM - 1)

        children.append(copy.deepcopy(parent1[:point1] + parent2[point1:point2] + parent1[point2:]))
    print("selection complete")
    return children
```

Figure 8. Tournament Selection Function

The selection function is a function that is used to get the best parent candidate from the population. If the fitness value formed from the fitness function above is higher, it is likely that the individual will become the best parent for the next population. The technique used in the program that we created is by using the tournament technique where high scores will be continued. In this function there is also a crossover function that functions to produce new offspring by involving two newly formed parents in the process.

```
def mutasi(genomes, scores):
    global nama_satpam
    for i in range(int(GENOMES_SIZE * PERCENT_UN_MUTATED), GENOMES_SIZE):
        for c in range(NUM_SATPAM):
            if random.randint(1, NUM_SATPAM) == 1:
                hasil_mutasi = random.randint(0, 2)
                if hasil_mutasi == 0:
                    new_satpam_id = random.choice(nama_satpam)
                    genomes[scores[i][0]][c].mutasi_satpam_id(new_satpam_id)
                elif hasil_mutasi == 1:
                    new_gedung = random.choice(gedungITERA)
                    genomes[scores[i][0]][c].mutasi_gedung(new_gedung)
                elif hasil_mutasi == 2:
                    start_time = random.randint(MIN_START_TIME, MAX_START_TIME)
                    day_structure = random.choice(DateTime.days_list)
                    date_time = DateTime(start_time, day_structure)
                    genomes[scores[i][0]][c].mutasi_date_time(date_time)
    print("mutasi complete")
```

Figure 9. Mutation Function

The function of a mutation in a program is to exchange the value of a gene with its inverse value. Every individual who has been through crossover will experience a gene mutation with a predetermined probability of mutation. The above mutation process is done by giving an inverse value or shifting the gene value to the genes that have been selected.

V. CONCLUSION

From the research, it was found that an automatic scheduling system can be used to assist the scheduling of security guards at ITERA. Scheduling is done using security data as many as 21 people and the number of buildings is 6 buildings. With this system, it can avoid schedule collisions or scheduling errors. With this scheduling system, it can optimize the scheduling process, because there is no need to input data one by one, this can be useful for the future along with the increasing number of security guards and buildings in ITERA. From this scheduling system, there are still vacancies in the schedule, this is due to the lack of security guard data. This is because the working rules of security guards at ITERA are Monday, Wednesday, Friday the duration of work is less than Tuesday, Thursday, Saturday, and Sunday. Whereby comparison the duration of working hours for Monday, Wednesday, Friday is 50 working hours, while for Tuesday and Thursday it is 70 working hours, and Saturday and Sunday are 70 working hours.

REFERENCES

[1] I. F. Ashari, "Implementation of Cyber-Physical-Social System Based on Service Oriented Architecture in Smart Tourism," *J. Appl. Informatics Comput.*, vol. 4, no. 1, pp. 66–73, 2020, doi: 10.30871/jaic.v4i1.2077.

[2] R. K. Sihotang and M. A. Aditya Wirangga, ST., "Perencanaan Kapasitas Produksi Dengan Metode Capacity Requirement Planning Di Teaching Factory Manufacture Electronics Politeknik Negeri Batam," *J. Bus. Adm.*, vol. 1, no. 1, pp. 1–9, 2017.

[3] L. Paranduk, A. Indriani, M. Hafid, and Suprianto, "Sistem Informasi Penjadwalan Mata Kuliah Menggunakan Algoritma Genetika Berbasis Web," *Semin. Nas. Apl. Teknol. Inf.*, pp. E46–E50, 2018.

[4] N. Tiandini and W. Anggraeni, "Penerapan Metode Kombinasi Algoritma Genetika dan Tabu Search dalam Optimasi Alokasi Kapal Peti Kemas (Studi Kasus : PT. XYZ)," *J. Tek. ITS*, vol. 6, no. 1, 2017, doi: 10.12962/j23373539.v6i1.21255.

[5] N. I. Kurniati, A. Rahmatulloh, and D. Rahmawati, "Perbandingan Performa Algoritma Koloni Semut Dengan Algoritma Genetika – Tabu Search Dalam Penjadwalan Kuliah," *Comput. Eng. Sci. Syst. J.*, vol. 4, no. 1, p. 17, 2019, doi: 10.24114/cess.v4i1.11387.

[6] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategiesin solving TSP," *Proc. World Congr. Eng. 2011, WCE 2011*, vol. 2, no. August, pp. 1134–1139, 2011.

[7] D. Oktarina and A. Hajjah, "Perancangan Sistem Penjadwalan Seminar Proposal dan Sidang Skripsi dengan Metode Algoritma Genetika," *JOISIE (Journal Inf. Syst. Informatics Eng.*, vol. 3, no. 1, p. 32, 2019, doi: 10.35145/joisie.v3i1.421.

[8] R. M. Puspita, A. Arini, and S. U. Masrurah, "Pengembangan Aplikasi Penjadwalan Kegiatan Pelatihan Teknologi Informasi Dan Komunikasi Dengan Algoritma Genetika (Studi Kasus: Bprtik)," *J. Online Inform.*, vol. 1, no. 2, pp. 76–81, 2016, doi: 10.15575/join.v1i2.43.

[9] R. R. Ilmi, W. F. Mahmudy, and D. E. Ratnawati, "Optimasi Penjadwalan Perawat Menggunakan Algoritma Genetika," *Univ. Brawijaya*, vol. 5, no. 13, pp. 1–8, 2015.

[10] T. Handoyo, A. K. Rachmawati, and E. Prasetyo, "Sistem Penjadwalan Mata Pelajaran di SMA Muhammadiyah 1 Kota Magelang Dengan Algoritma Genetika," *Transformasi*, vol. 11, no. 1, pp. 14–19, 2015.

[11] J. Carr, *Introduction to genetic algorithms*. 2014.

[12] T. Alam, S. Qamar, A. Dixit, and M. Benaida, "Genetic algorithm: Reviews, implementations and applications," *Int. J. Eng. Pedagog.*, vol. 10, no. 6, pp. 57–77, 2021, doi: 10.3991/IJEP.V10I6.14567.