CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

Here is the pseudocode for genetic algorithms

```
1. ReadScheduleCSV()
2. initialize reader
3. while current read line != null
4.     if first line
5.           set the schedule title
6.           initialize constraints fitness value
7.     else
8.           add activity data into schedule
9.     endif
10. endwhile
```

The pseudocode above has function to read the schedule.csv file that contains all church activities and divisions with the number of members needed for a month and load them into memory.

Line 2 is reader initialization to read txt files. Lines 3 to 10 are the while looping process for reading lines one by one in the txt file. Lines 5 and 6 are used to read the schedule title and initialize the total fitness value and fitness value of each constraint. Line 8 are used to read the name, date, start time, and time of completion of an activity, and what divisions are included / scheduled into the activity.

```
1. RandomizeMember()
2. for i=0 until i < number of activities in the schedule
3.     for j=1 until j < number of divisions in the activity
4.           get division name and number of members needed
5.           for k=0 until k < number of members needed
6.                 add members from database to the division
                   randomly
7.           end for k
8.     end for j
9. end for i
```

The pseudocode above has function to input members of a division registered in the database randomly to an activity in a schedule. This pseudocode use 3 layers of looping, the first layer (i) is to process each activity in a schedule, the second layer (j) is to process each division and the third layer (k) is to process each member in a division.

Line 4 is used to read the division names and the number of members to be scheduled. Line 5 to 7 are the data retrieval process in the form of member names in a division from the database to be scheduled to an activity according to the number of members needed and the member names are taken based on random numbers obtained from Math.random function.

```
1. EvaluateSchedule()
2.     initialize all constraints fitness value to 0
3.     //constraint 1
4.     for i=0 until i < number of activities in the schedule
5.         for j=1 until j < number of divisions in the activity
6.             check if there is repeated names in the division
7.             if repeated names found
8.                 constraint 1 fitness value +1
9.             endif
10.        end for j
11.    end for i
12.
13.    //constraint 2
14.    for i=1 until i < number of activities in the schedule
15.        for j=1 until j < number of divisions in the activity
16.            for k=j+1 until k < number of divisions in the
   activity
17.                check if there is member present in both
   division j and k
18.                if there is member present in both
   division j and k
19.                    constraint 2 fitness value +1
20.                endif
21.            end for k
22.        end for j
23.    end for i
24.
25.    //constraint 3
26.    for i=1 until i < number of activities in the schedule - 1
27.        for j=i+1 until j < number of activities in the
   schedule
28.            check if collision occurs between activity i and
   j
29.            if activity collision occurs
```

```
30.                     check if there is member that serve in
  both activity i and j
31.                     if there is member that serve in both
  activity i and j
32.                           constraint 3 fitness value +1
33.                     endif
34.               endif
35.         endfor j
36.   endfor i
37.
38.   //constraint 4
39.   initialize temp (2D ArrayList)
40.   for i=1 until i < number of activities in the schedule
41.         for j=1 until j < number of divisions in the activity
42.               check if the division exists in the temp
43.               if(division exists in the temp)
44.                     add all members in the division to temp
45.               else
46.                     add the new division name and its members
  to temp
47.               endif
48.         endfor j
49.   endfor i
50.
51.   for i=0 until i < number of divisions in the temp
52.         get number of total slots of the division available in
  the schedule
53.         get number of members of the division registered in
  the database
54.         set maxserve = number of total slots available /
  number of members of the division registered in the database +
  1
55.         for j=0 until j < number of members in the division
56.               check member frequency
57.               if member frequency > maxserve
58.                     constraint 4 fitness value +1
59.               endif
60.         endfor j
61.   endfor i
62.   set totalfitness = constraint 1 fitness value + constraint 2
  fitness value + constraint 3 fitness value + constraint 4
  fitness value
```

The pseudocode above has function to evaluate a schedule with 4 predetermined constraints. Each time the schedule does not meet the requirements of any of the 4 constraints, the fitness value (error value) of that schedule will increase.

Line 2 has function to initialize the fitness value of the schedule, which is 0. Line 4 to 11 are the checking process for constraint 1, that is in a division in an activity, there should be no member names that are repeated, line 14 to 23 is a checking process for constraint 2 that is in an activity, there should be no members that are scheduled in 2 divisions at once, line 26 to 36 are the process of checking the schedule with constraint 3, that is there should be no members that are scheduled in 2 activities at the same time, and line 39 to 61 is the process of checking the schedule with constraint 4, that is each member cannot be scheduled exceeds the maximum limit of serve (maximum limit = the total number of slots of a division available in a schedule / total number of members registered in the division). Line 62 has function to calculate the total of the fitness values of the 4 constraints that have been obtained from the previous processes.

```
1. CrossOverMutationSchedule()
2.     initialize maxcrossover, divcount, crossovercount,
  mutationrate, random
3.     set 2 schedule from population with best fitness value as
  parent
4.     initialize 2 new schedule as child
5.     set child1 = parent 1 (deep copy)
6.     set child2 = parent 2 (deep copy)
7.     set divcount = total number of divisions in the child
  schedule
8.     set maxcrossover = Random * divcount
9.     while crossovercount < maxcrossover
10.          crossover random divisions in both childs
11.    endwhile
12.
13.    set mutationrate = 30
14.    for i=0 until i < number of activities in the child schedule
15.          for j=1 until j < number of divisions in the activity
16.                for k=1 until k < number of members in the
  division
17.                      set random1 = Random * 100
18.                      if random1 < 30
19.                            replace the current member to other
  member registered in the database (mutation)
20.                      endif
21.                endfor k
22.          endfor j
23.    endfor i
24.    add both child schedules into the population
```

The pseudocode above is part of a genetic algorithm that is the crossover process to produce new individuals / schedules from 2 parents who have the best fitness value and the mutation process for the 2 new individuals obtained from the crossover.

Line 3 is the processs to select the population to get 2 individuals with the best fitness value as the parent. Line 4 to 6 has function to copy 2 parents who have been selected with the deep copy method to get 2 new individuals / child. Line 7 and 8 has function to determine the crossover frequency to be run randomly (frequency = random of the total number of divisions in a schedule). Line 9 to 11 has function to perform a crossover process in divisions in a schedule randomly so that 2 child / new schedules that are obtained will be different from the parent.

Line 13 to 23 has function to perform the mutation process for 2 child that are obtained from crossover result in the previous process, mutations are done by changing the names of existing scheduled members with the other members registered in the database randomly with a possible mutation rate of 30% so that the 2 child will be further different from the parent. Line 24 is the process to add the 2 child that are obtained from the result of crossover and mutation process into the population.

```
1. FilterPopulation()
2. for i=0 until i < 2
3.     remove schedule with worst fitness value from the population
4. endfor i
```

The pseudocode above has function to filter the population, which is to delete or eliminate 2 individuals with the worst fitness value so that the number of individuals in the population will return to its original number.

```
1. GeneticAlgorithm()
2. initialize complete = false
3. initialize loop, noupdate, currentfitness, bestfitness
4. while complete == false
5.     loop++
6.     evaluate all schedule in the population (EvaluateSchedule())
7.     set currentfitness = best fitness value of a schedule in the
   population
8.     if currentfitness is better than bestfitness
9.           set bestfitness = currentfitness
10.          set noupdate = 0
11.    else
12.          noupdate++
13.    endif
14.    if there is a schedule in the population with fitness value
   of 0 or loop > 5000000 or noupdate > 1000000
15.          complete = true
16.          break
17.    else
18.          crossover and mutation schedules in the population
   (CrossOverMutationSchedule)
19.          filter the population (FilterPopulation())
20.    endif
21. endwhile
```

Pseudocode above is the whole genetic algorithm process. Line 2 and 3 are initialization of the helping variables. Line 4 to 21 is the while looping which is the core of the genetic algorithm that is if the boolean status complete == false, then this process will continue until complete == true, these processes are at line 5 is the command to add the value of the counter variable loop to record the number of loops that have been done, at line 6 is to call the EvaluateScheduleGenetic() method to evaluate all individuals or schedules in the population. Line 10 has function to reset the value of the noupdate counter variable to 0 if the new individual is found with the fitness value better than other individuals in the population. Line 12 is a command to add the noupdate counter variable value each time the genetic algorithm process does not make any progress (the discovery of new individuals with better fitness values). Line 14 to 20 is the stopping conditions of the genetic algorithm that is if an individual or schedule has been found with a fitness value of 0 or if the while looping has reached 5 million generations or if the noupdate variable has reached 1 million, then this genetic

algorithm will be stopped. As long as the three conditions above have not been fulfilled, the crossover, mutation and filtering process will be carried out until one of the three conditions above is fulfilled.

```
1. SAHCAlgorithm()
2. initialize optimized = false
3. initialize loop, noupdate, bestfitness, currentfitness
4. generate initial schedule
5. evaluate the initial schedule (EvaluateSchedule())
6. if initial schedule fitness value == 0
7.     set optimized = true
8. endif
9. while optimized == false
10.    loop++
11.    set current schedule = initial schedule
12.    evaluate the current schedule (EvaluateSchedule())
13.    if current schedule fitness value == 0 or loop > 5000000 or
   noupdate > 1000000
14.          set optimized = true
15.          break
16.    endif
17.    generate new schedule
18.    evaluate the new schedule (EvaluateSchedule())
19.    if current schedule fitness value is better than new
   schedule fitness value
20.          noupdate++
21.    else
22.          set current schedule = new schedule
23.          set noupdate = 0
24.    endif
25.endwhile
```

Pseudocode above is the whole process of the steepest ascent hill climbing algorithm. Line 2 and 3 is the initialization of the helping variables. Line 4 is the command to generate the initial schedule randomly and the schedule will be evaluated on line 5 by calling the EvaluateSchedule() method. Line 6 to 8 has function to determine the fitness value of the initial schedule, if the initial schedule already has fitness value of 0 (optimal), then the SAHC algorithm is immediately finished or stopped, but if it is not, then it will continue with the looping process in line 9 to 24, these processes are, on line 10 is a command to add the value of the loop counter variable to record the number of loops that have

been done. Line 11 is to set the initial schedule as the current schedule. Line 12 has function to evaluate the current schedule by calling the EvaluateSchedule() method. Line 13 to 16 is the stopping conditions of this SAHC algorithm, that is if a schedule with the fitness value of 0 has been found or if the while loop has reached 5 million times or if the noupdate variable has reached 1 million, then the SAHC algorithm will be stopped. Line 17 is the command to generate a new schedule randomly and the new schedule will be evaluated on line 18 by calling the EvaluateSchedule() method. Line 19 to 24 has function to get the fitness value of the new schedule that has been previously evaluated and compare the fitness value of the new schedule with the current schedule fitness, if the fitness value of the current schedule is better than the new schedule, then there is no progress and in line 20 the value of the noupdate counter variable will increase, but if the fitness value of the new schedule is better than the current schedule, then set the new schedule as the current schedule by using deep copy method on line 22 and on line 23 the noupdate counter variable will be reset again to 0.

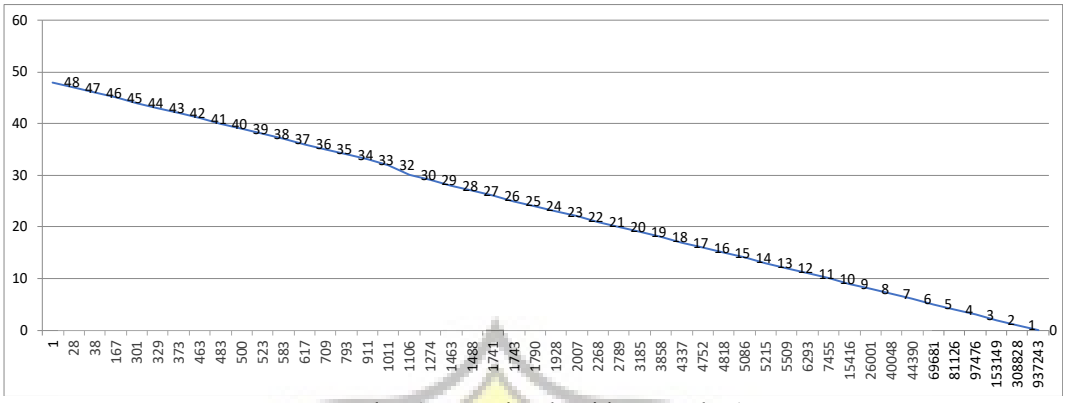
5.2 Testing

Testing is done by running Java programs with genetic algorithm and SAHC algorithm 10 times to produce 10 variants of schedule for church activities in December 2018 with the following series of events / activities

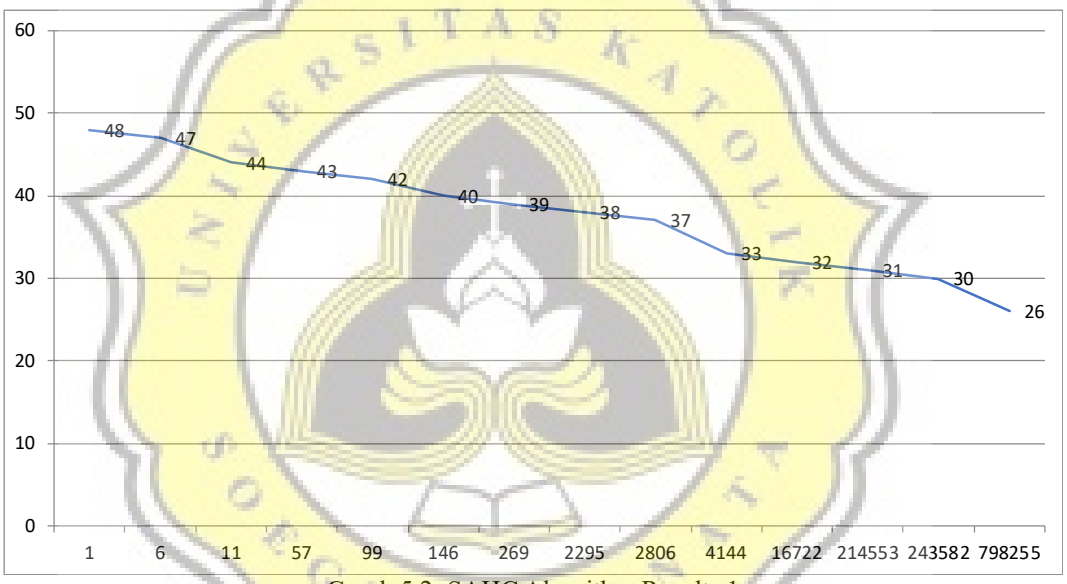
- Morning Public Worship 1 (December 2, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 2, 2018, 09.00 – 11.00)
- Evening Public Worship (December 2, 2018, 17.00 – 19.00)
- Morning Public Worship 1 (December 9, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 9, 2018, 09.00 – 11.00)
- Evening Public Worship (December 9, 2018, 17.00 – 19.00)
- Morning Public Worship 1 (December 16, 2018, 06.30 – 08.30)

- Morning Public Worship 2 (December 16, 2018, 09.00 – 11.00)
- Evening Public Worship (December 16, 2018, 17.00 – 19.00)
- Morning Public Worship 1 (December 23, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 23, 2018, 09.00 – 11.00)
- Evening Public Worship (December 23, 2018, 17.00 – 19.00)
- Christmas Night Worship (December 24, 2018, 18.00 – 20.00)
- Special Christmas Worship (December 25, 2018, 17.00 -19.30)
- Morning Public Worship 1 (December 30, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 30, 2018, 09.00 – 11.00)
- Evening Public Worship (December 30, 2018, 17.00 – 19.00)
- Ibadah Tutup Tahun (December 31, 2018, 22.00 – 24.00)

After running the program with the implementation of genetic algorithm, the most effective number of individuals in the population is 2 individuals, and here are the results of both genetic and SAHC algorithms

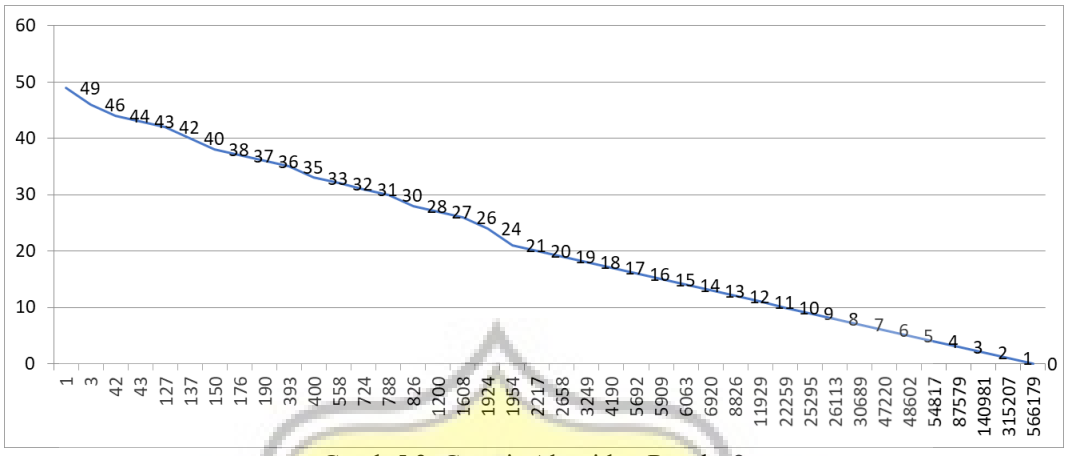


Graph 5.1: Genetic Algorithm Results 1

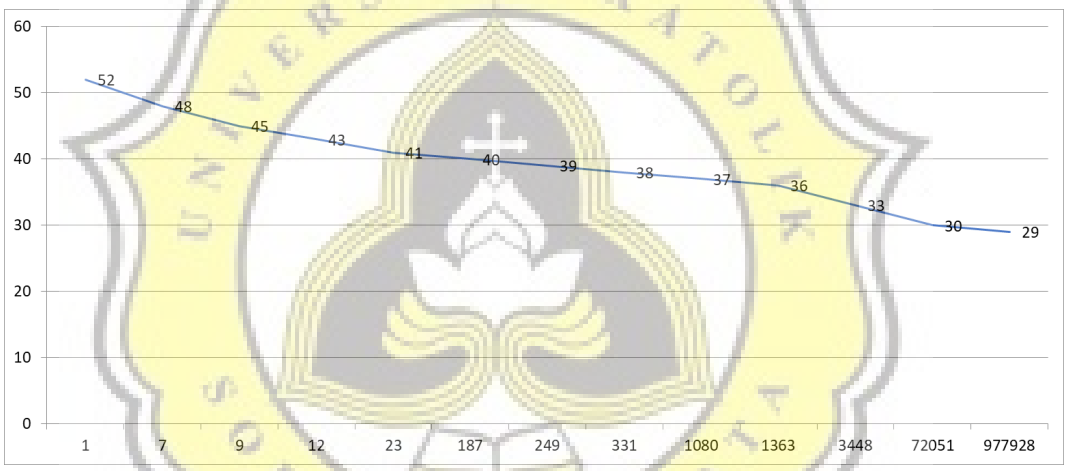


Graph 5.2: SAHC Algorithm Results 1

From the graphs above, both algorithms start with fitness value of 48. In the early stage, the SAHC algorithm can improve the fitness value from 48 to 39 in 269 generations, while the genetic algorithm requires 500 generations to reach that value. But after that, the genetic algorithm outperforms the SAHC algorithm, for example to reach the fitness value of 32, the genetic algorithm only need 1011 generations, whereas the SAHC algorithm requires 16722 generations. And in the end, the SAHC algorithm can only find a schedule with fitness value of 26 with 798255 generations, while the genetic algorithm can find the best optimal schedule with fitness value of 0 with 937243 generations.

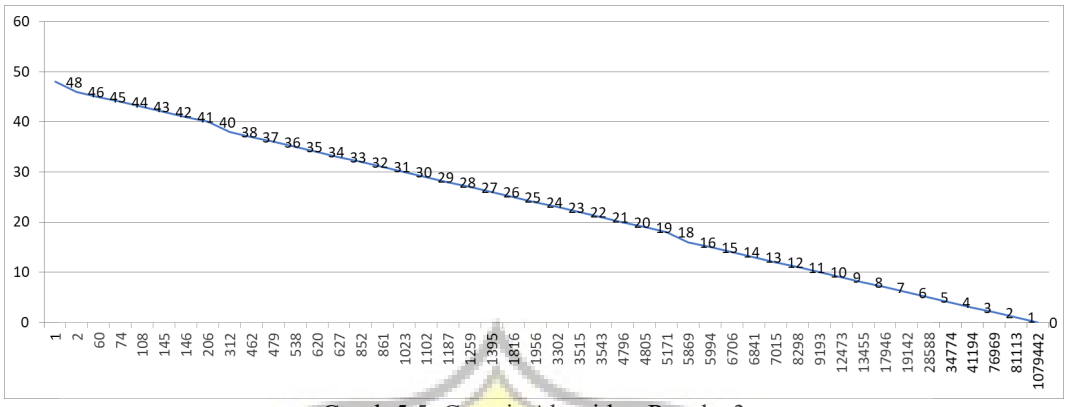


Graph 5.3: Genetic Algorithm Results 2

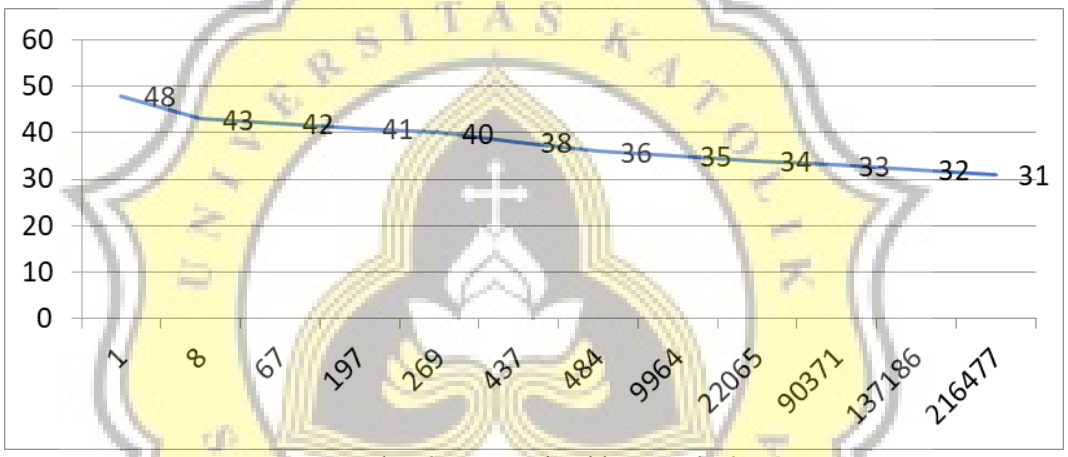


Graph 5.4: SAHC Algorithm Results 2

From the graphs above, the genetic algorithm starts with the fitness value of 49 and the SAHC algorithm starts with the fitness value of 52. In the early stages, the SAHC algorithm performs better than the genetic algorithm, it can improve the fitness value from 52 to 41 only in 23 generations, while the genetic algorithm needs 127 generations to get the fitness value of 42. But in the middle until the end of the process, the genetic algorithm outperforms the SAHC algorithm, it can find the best optimal schedule with fitness value of 0 with 566179 generations whereas the SAHC algorithm can only find a schedule with fitness value of 29 with more generations required that is 977928 generations.

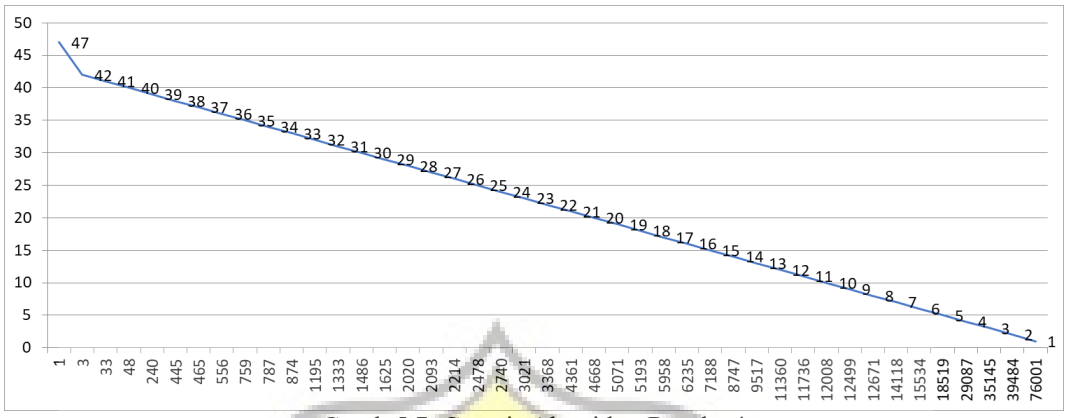


Graph 5.5: Genetic Algorithm Results 3

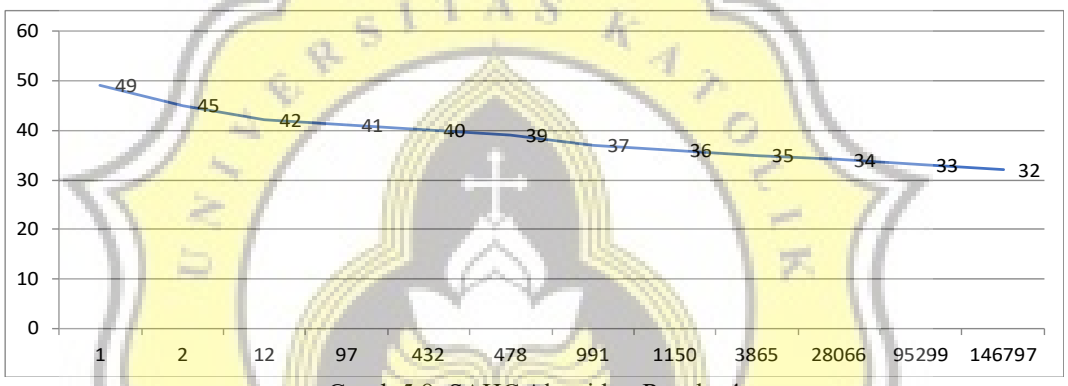


Graph 5.6: SAHC Algorithm Results 3

From the graphs above, both algorithms start with same fitness value that is 48. In the early process, the SAHC algorithm can improve the fitness value from 48 to 43 just in 8 generations, whereas the genetic algorithm requires 108 generations. In the middle of the process, the SAHC algorithm requires 9964 generations to get the fitness value of 35, while to reach that value, the genetic algorithm only needs 538 generations. And same as the previous results, the genetic algorithm outperforms the SAHC algorithm, it can find the best optimal schedule with fitness value of 0 with 1079442 generations whereas the SAHC algorithm cannot find the best optimal schedule, it can only find a schedule with fitness value of 31 with 216477 generations that the genetic algorithm can get it only in 861 generations.

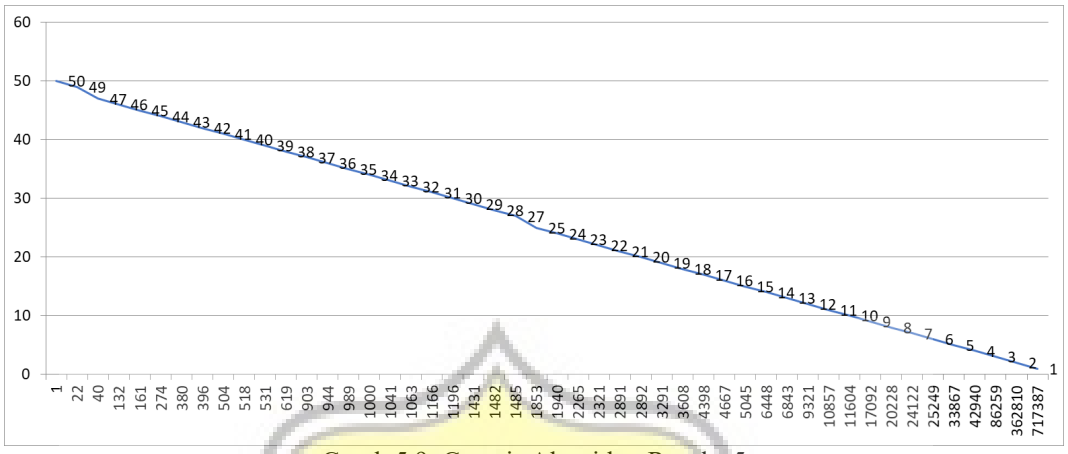


Graph 5.7: Genetic Algorithm Results 4

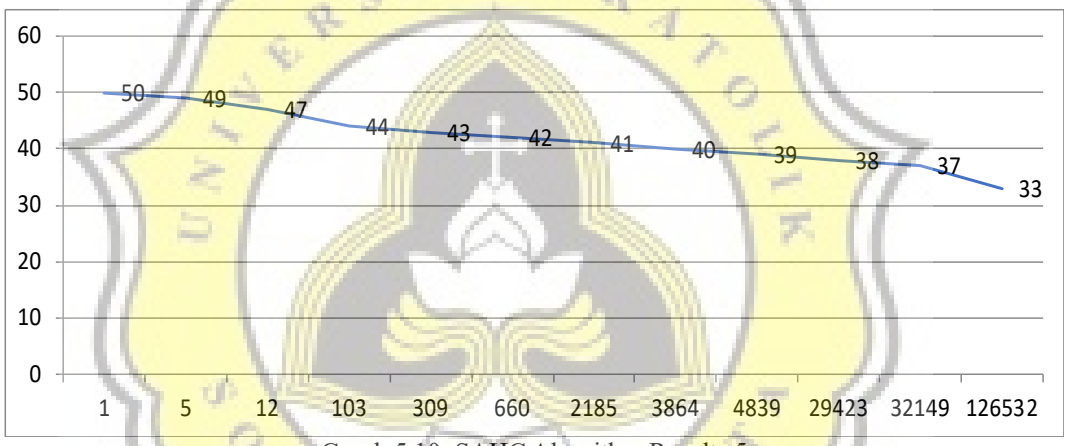


Graph 5.8: SAHC Algorithm Results 4

From the graphs above, the genetic algorithm starts with the fitness value of 47 and the SAHC algorithm starts with the fitness value of 49. In the beginning, both algorithms performs equally well, the genetic algorithm can improve the fitness value from 47 to 42 only in 3 generations, while the SAHC algorithm can improve the fitness value from 49 to 45 in 2 generations. But same as in the previous results, in the middle until the end of the process, the genetic algorithm performs better than the SAHC algorithm, although it can't find the best optimal schedule, but it can find a schedule with fitness value of 1 with only 76001 generations whereas the SAHC algorithm also can't find the best optimal schedule, but it only can find a schedule with fitness value of 32 with more generations needed that is 146797 generations.

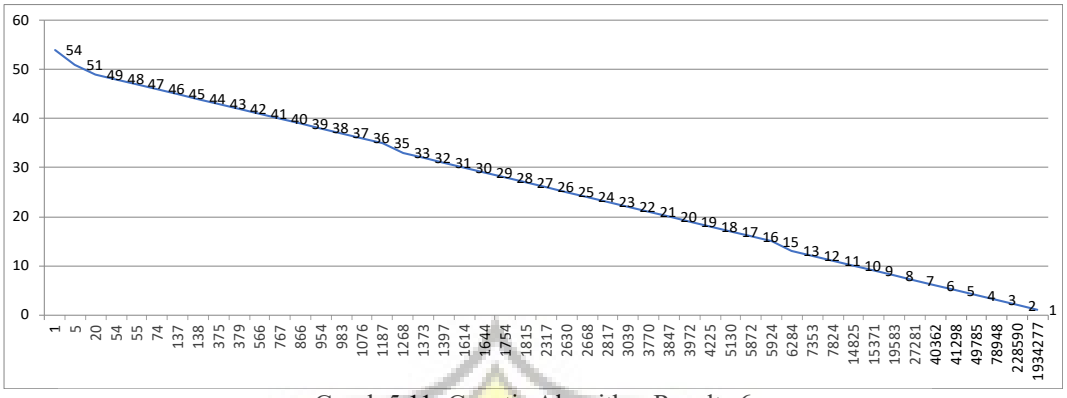


Graph 5.9: Genetic Algorithm Results 5

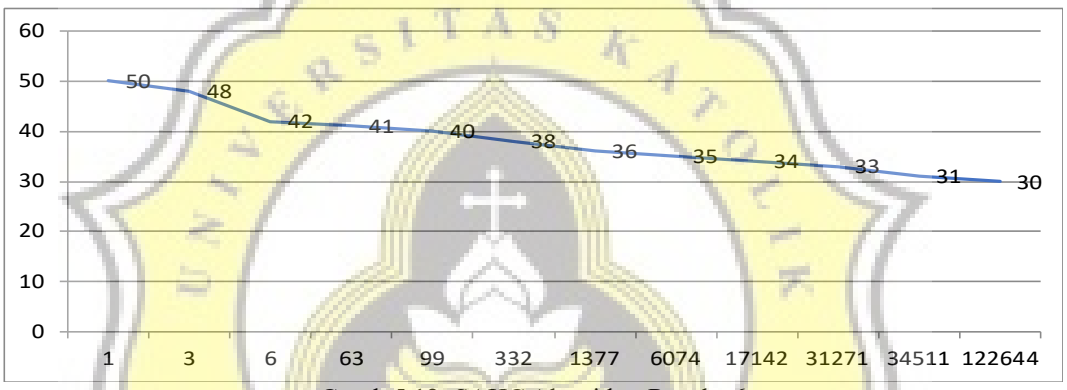


Graph 5.10: SAHC Algorithm Results 5

From the graphs above, both algorithms start with the same fitness value of 50. At first, the SAHC algorithm performs very well, it can improve the fitness value from 50 to 44 only in 103 generations, whereas the genetic algorithm requires 274 generations to get the fitness value of 44. But in the middle of the process, to get the fitness value of 40 the SAHC algorithm requires 3864 generations, while the genetic algorithm only needs 518 generations. And in the end, although both algorithms can't find the best optimal schedule, the genetic algorithm still performs better than the SAHC because it can find a schedule with the fitness value of 1 in 717387 generations whereas the SAHC algorithm can only find a schedule with fitness value of 33 in 126532 generations that the genetic algorithm can get it only in 1041 generations.

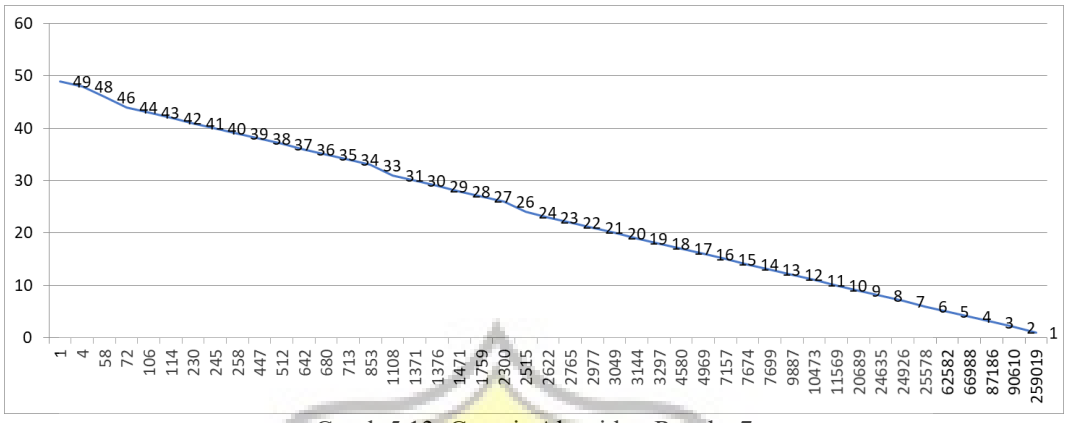


Graph 5.11: Genetic Algorithm Results 6

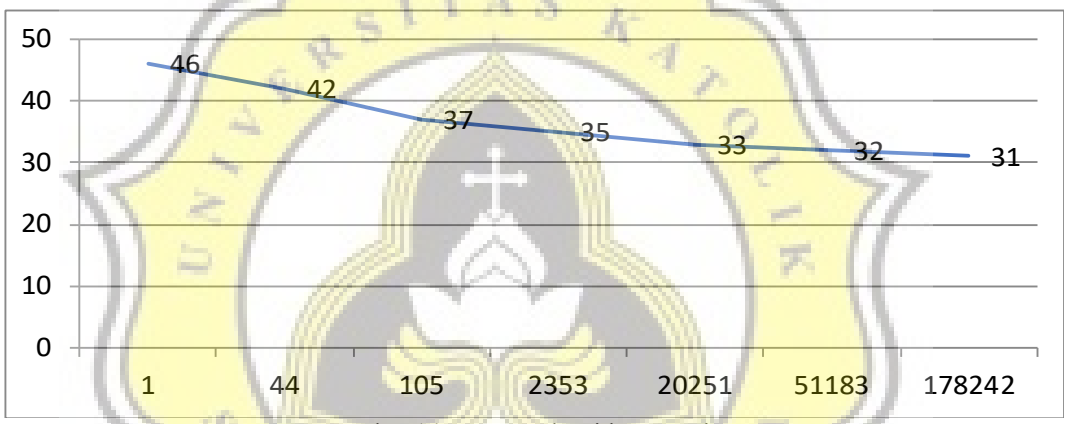


Graph 5.12: SAHC Algorithm Results 6

From the graphs above, the genetic algorithm starts with the initial fitness value of 60 and the SAHC algorithm starts with the initial fitness value of 50. In the early process, the SAHC algorithm performs much better than the genetic algorithm because it can improve the fitness value from 50 to 42 just in 6 generations, whereas the genetic algorithm requires 379 generations. But later, to get fitness value of 35 the genetic algorithm only needs 1187 generations, whereas the SAHC algorithm can get it after 6074 generations. And in the end the genetic algorithm can't find the best optimal schedule, but it can find a schedule with fitness value of 1 in 1934277 generations whereas the SAHC algorithm also can't find the best optimal schedule, it only can find a schedule with fitness value of 30 in 122644 generations that the genetic algorithm can get it only in 1614 generations.

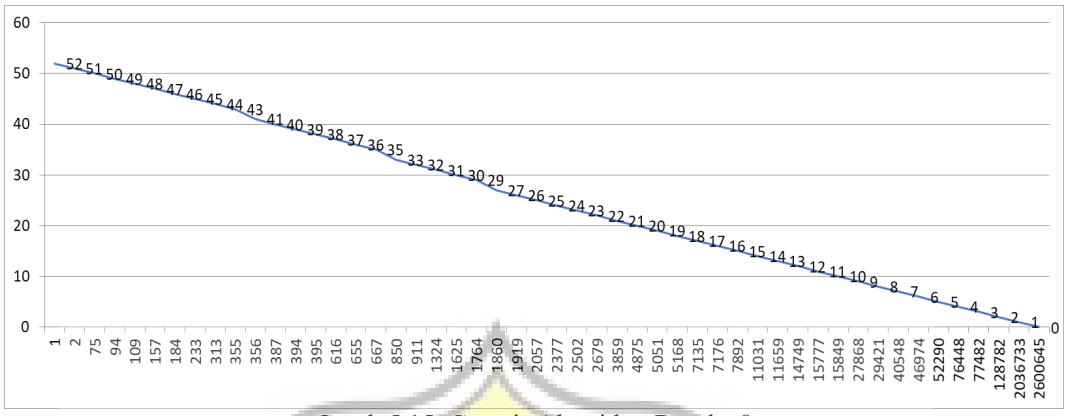


Graph 5.13: Genetic Algorithm Results 7

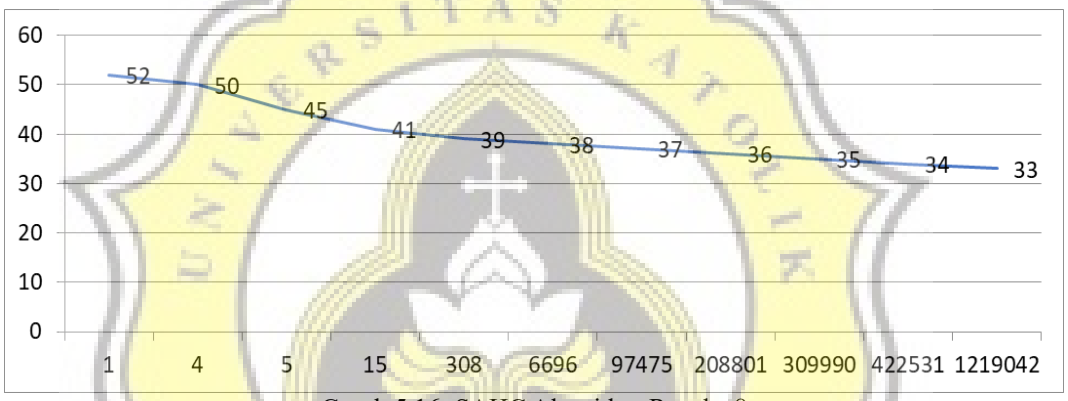


Graph 5.14: SAHC Algorithm Results 7

From the graphs above, the genetic algorithm starts with the fitness value of 49 and the SAHC algorithm starts with the fitness value of 46. In the beginning, the SAHC algorithm performs slightly better than the genetic algorithm, for example to get a schedule with fitness value of 37, the SAHC algorithm only needs 105 generations, while the genetic algorithm requires 512 generations. But in the middle until the end of the process, the genetic algorithm performs much better than the SAHC algorithm, although both algorithms can't find the best optimal schedule, but the genetic algorithm can find a schedule with fitness value of 1 after 259019 generations whereas the SAHC algorithm only can find a schedule with fitness value of 31 with 178242 generations that the genetic algorithm can get it only in 1108 generations.

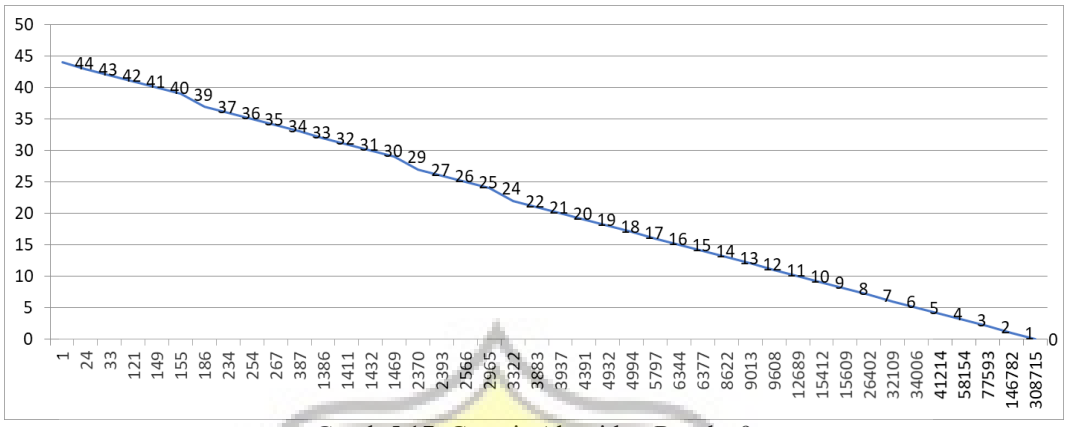


Graph 5.15: Genetic Algorithm Results 8

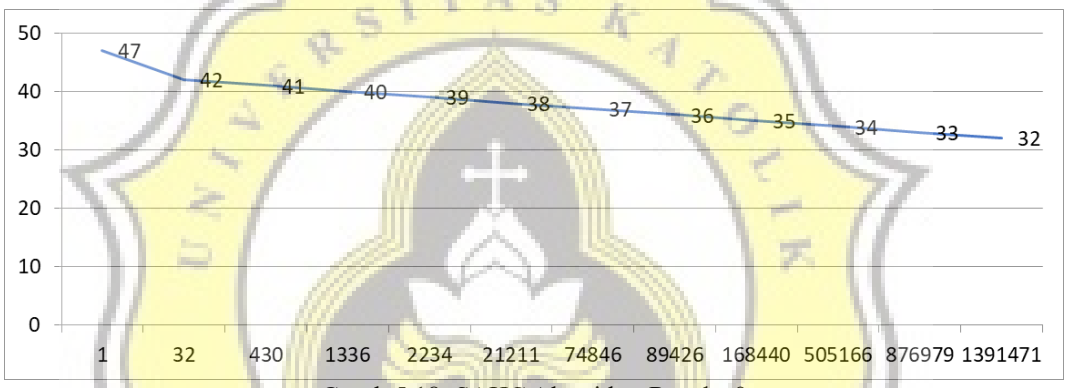


Graph 5.16: SAHC Algorithm Results 8

From the graphs above, both algorithms start with fitness value of 52. In the beginning, the SAHC algorithm performs much better than the genetic algorithm because it can improve the fitness value from 52 to 41 just in 15 generations, while to get the fitness value of 41, the genetic algorithm requires more that is 356 generations. But after that, the genetic algorithm outperforms the SAHC algorithm, for example to reach the fitness value of 36, the genetic algorithm only needs 655 generations, whereas the SAHC algorithm requires much more generations that is 208801 generations. And in the end, the genetic algorithm can find the best optimal schedule with fitness value of 0 after 2600645 generations, whereas the SAHC algorithm can only find a schedule with fitness value of 33 with 1219042 generations that the genetic algorithm can get it only in 850 generations.

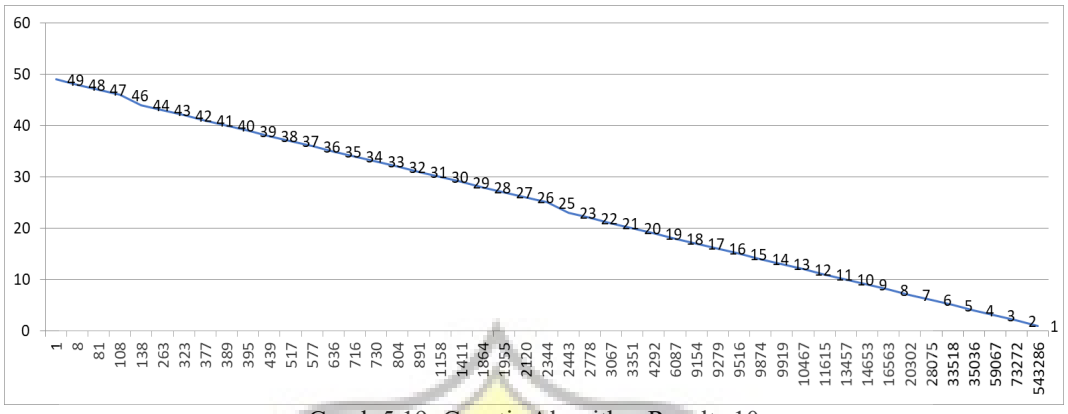


Graph 5.17: Genetic Algorithm Results 9

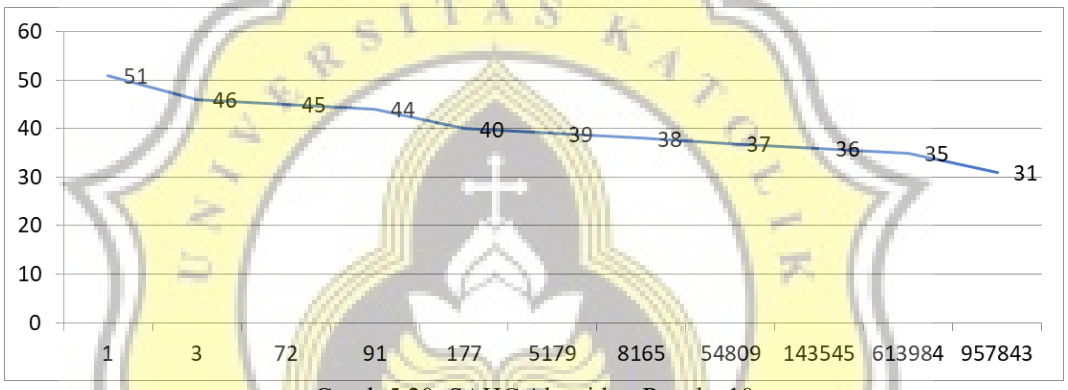


Graph 5.18: SAHC Algorithm Results 9

From the graphs above, the genetic algorithm starts with the fitness value of 44 and the SAHC algorithm starts with the fitness value of 47. At first, both algorithms performs equally well, the SAHC algorithm can improve the fitness value from 47 to 42 in 32 generations, and the genetic algorithm requires 33 generations to get that fitness value. But later, same as in the previous results, the genetic algorithm performs much better than the SAHC algorithm, the genetic algorithm can find a schedule with fitness value of 37 in only 186 generations whereas the SAHC algorithm can find it after 74846 generations. And in the end, the genetic algorithm can find the best optimal schedule with the fitness value of 0 after 308715 generations, whereas the SAHC algorithm can't find it and can only find a schedule with fitness value of 32 with more generations required that is 1391471 generations.



Graph 5.19: Genetic Algorithm Results 10



Graph 5.20: SAHC Algorithm Results 10

From the graphs above, the genetic algorithm starts with the fitness value of 49 and the SAHC algorithm starts with the fitness value of 51. In the early process, the SAHC algorithm performs better than the genetic algorithm, it can improve the fitness value from 51 to 46 only after 3 generations, while the genetic algorithm needs 108 generations to get it. But in the middle until the end of the process, although both algorithm can't find the best optimal schedule, the genetic algorithm performs much better than the SAHC algorithm because it can find a schedule with fitness value of 1 after 543286 generations whereas the SAHC algorithm can only find a schedule with fitness value of 31 with more generations needed that is 957843 generations that the genetic algorithm can get it only in 891 generations.

From the results and analysis above, it can be concluded that the genetic algorithm has been consistently outperforms the SAHC algorithm. If the two algorithms are compared in terms of results, genetic algorithms are able to find the optimal schedule solutions with the fitness value (error value) of 0 or 1 whereas the SAHC algorithm cannot find the optimal schedule solution, the SAHC algorithm can only find the schedule solution with the fitness value of approximately 30 that the genetic algorithm can find the schedule with fitness value of 30 only in approximately 1280 generations.

When viewed in terms of the complexity of the algorithm with Big-O notation, the genetic algorithm has complexity of O (CPM^2N^2), while the SAHC algorithm has a complexity of O (CM^2N^2), where C is the while loop (as long as one of the stopping conditions is not met, do the algorithm), M and N is the looping for schedule operations (evaluation, crossover, mutation, etc) and P is the number of individuals in the population.

Since this project only use 2 individuals for the population, then the P variable only has little impact in the algorithm, so in this project, both algorithm have same complexity that is O (CPM^2N^2). And when viewed in terms of processing speed, the genetic algorithm has processing speed of 240 generations / sec approximately, whereas the SAHC algorithm has processing speed of approximately 207 generations / sec.