APPENDIX

Here is the tables of running results of genetic algorithm

1st Run(3841s)		2nd Run(2315s)		3rd Run(4414s)		4th Run(8589s)		5th Run(11221s)	
Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness
1	48	1	49	1	48	1	47	1	50
28	47	3	46	2	46	3	42	22	49
38	46	42	44	60	45	33	41	40	47
167	45	43	43	74	44	48	40	132	46
301	44	127	42	108	43	240	39	161	45
329	43	137	40	145	42	445	38	274	44
373	42	150	38	146	41	465	37	380	43
463	41	176	37	206	40	556	36	396	42
483	40	190	36	312	38	759	35	504	41
500	39	393	35	462	37	787	34	518	40
523	38	400	33	479	36	874	33	531	39
583	37	558	32	538	35	1195	32	619	38
617	36	724	31	620	34	1333	31	903	37
709	35	788	30	627	33	1486	30	944	36
793	34	826	28	852	32	1625	29	989	35
911	33	1200	27	861	31	2020	28	1000	34
1011	32	1608	26	1023	30	2093	27	1041	33
1106	30	1924	24	1102	29	2214	26	1063	32
1274	29	1954	21	1187	28	2478	25	1166	31
1463	28	2217	20	1259	27	2740	24	1196	30
1488	27	2658	19	1395	26	3021	23	1431	29
1741	26	3249	18	1816	25	3368	22	1482	28
1743	25	4190	17	1956	24	4361	21	1485	27
1790	24	5692	16	3302	23	4668	20	1853	25
1928	23	5909	15	3515	22	5071	19	1940	24
2007	22	6063	14	3543	21	5193	18	2265	23
2268	21	6920	13	4796	20	5958	17	2321	22
2789	20	8826	12	4805	19	6235	16	2891	21
3185	19	11929	11	5171	18	7188	15	2892	20
3858	18	22259	10	5869	16	8747	14	3291	19

4337	17	25295	9	5994	15	9517	13	3608	18
4752	16	26113	8	6706	14	11360	12	4398	17
4818	15	30689	7	6841	13	11736	11	4667	16
5086	14	47220	6	7015	12	12008	10	5045	15
5215	13	48602	5	8298	11	12499	9	6448	14
5509	12	54817	4	9193	10	12671	8	6843	13
6293	11	87579	3	12473	9	14118	7	9321	12
7455	10	140981	2	13455	8	15534	6	10857	11
15416	9	315207	1	17946	7	18519	5	11604	10
26001	8	566179	0	19142	6	29087	4	17092	9
40048	7			28588	5	35145	3	20228	8
44390	6			34774	4	39484	2	24122	7
69681	5			41194	3	76001	1	25249	6
81126	4			76969	2			33867	5
97476	3			81113	1			42940	4
153149	2			1079442	0			86259	3
308828	1							362810	2
937243	0							717387	1

6th Run(16056s)		7th Run(9329s)		8th Run(10664s)		9th Run(1254s)		10th Run(11864s)	
Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness
1	54	1	49	1	52	1	44	1	49
5	51	4	48	2	51	24	43	8	48
20	49	58	46	75	50	33	42	81	47
54	48	72	44	94	49	121	41	108	46
55	47	106	43	109	48	149	40	138	44
74	46	114	42	157	47	155	39	263	43
137	45	230	41	184	46	186	37	323	42
138	44	245	40	233	45	234	36	377	41
375	43	258	39	313	44	254	35	389	40
379	42	447	38	355	43	267	34	395	39
566	41	512	37	356	41	387	33	439	38
767	40	642	36	387	40	1386	32	517	37
866	39	680	35	394	39	1411	31	577	36
954	38	713	34	395	38	1432	30	636	35
983	37	853	33	616	37	1469	29	716	34
1076	36	1108	31	655	36	2370	27	730	33
1187	35	1371	30	667	35	2393	26	804	32
1268	33	1376	29	850	33	2566	25	891	31
1373	32	1471	28	911	32	2965	24	1158	30
1397	31	1759	27	1324	31	3322	22	1411	29
1614	30	2300	26	1625	30	3883	21	1864	28
1644	29	2515	24	1764	29	3937	20	1955	27
1754	28	2622	23	1860	27	4391	19	2120	26
1815	27	2765	22	1919	26	4932	18	2344	25
2317	26	2977	21	2057	25	4994	17	2443	23
2630	25	3049	20	2377	24	5797	16	2778	22
2668	24	3144	19	2502	23	6344	15	3067	21
2817	23	3297	18	2679	22	6377	14	3351	20
3039	22	4580	17	3859	21	8622	13	4292	19
3770	21	4969	16	4875	20	9013	12	6087	18
3847	20	7157	15	5051	19	9608	11	9154	17
3972	19	7674	14	5168	18	12689	10	9279	16
4225	18	7699	13	7135	17	15412	9	9516	15

5130	17	9887	12	7176	16	15609	8	9874	14
5872	16	10473	11	7892	15	26402	7	9919	13
5924	15	11569	10	11031	14	32109	6	10467	12
6284	13	20689	9	11659	13	34006	5	11615	11
7353	12	24635	8	14749	12	41214	4	13457	10
7824	11	24926	7	15777	11	58154	3	14653	9
14825	1	25578	6	15849	10	77593	2	16563	8
15371	9	62582	5	27868	9	146782	1	20302	7
19583	8	66988	4	29421	8	308715	0	28075	6
27281	7	87186	3	40548	7			33518	5
40362	6	90610	2	46974	6			35036	4
41298	5	259019	1	52290	5			59067	3
49785	4			76448	4			73272	2
78948	3			77482	3			543286	1
228590	2			128782	2				
1934277	1			2036733	1				
				2600645	0				

Here is the tables of running results of the SAHC algorithm

1st Run(12871s)		2nd Run(14653s)		3rd Run(10280s)		4th Run(10144s)		5th Run(9584s)	
Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness
1	48	1	52	1	48	1	49	1	50
6	47	7	48	8	43	2	45	5	49
11	44	9	45	67	42	12	42	12	47
57	43	12	43	197	41	97	41	103	44
99	42	23	41	269	40	432	40	309	43
146	40	187	40	437	38	478	39	660	42
269	39	249	39	484	36	991	37	2185	41
2295	38	331	38	9964	35	1150	36	3864	40
2806	37	1080	37	22065	34	3865	35	4839	39
4144	33	1363	36	90371	33	28066	34	29423	38
16722	32	3448	33	137186	32	95299	33	32149	37
214553	31	72051	30	216477	31	146797	32	126532	33
243582	30	977928	29						
798255	26								

6th Run(10140s)		7th Run(10472s)		8th Run(15653s)		9th Run(16298s)		10th Run(13936s)	
Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness	Generation	Fitness
1	50	1	46	1	52	1	47	1	51
3	48	44	42	4	50	32	42	3	46
6	42	105	37	5	45	430	41	72	45
63	41	2353	35	15	41	1336	40	91	44
99	40	20251	33	308	39	2234	39	177	40
332	38	51183	32	6696	38	21211	38	5179	39
1377	36	178242	31	97475	37	74846	37	8165	38
6074	35			208801	36	89426	36	54809	37
17142	34			309990	35	168440	35	143545	36
31271	33			422531	34	505166	34	613984	35
34511	31			1219042	33	876979	33	957843	31
122644	30					1391471	32		

Here is the full source code that used in this project

Schedule.java

```
1. import java.util.ArrayList;
2. import java.util.Set;
3. import java.util.HashSet;
4. import java.util.Collections;
5. import java.lang.Math;
6. import java.io.File;
7. import java.io.BufferedReader;
8. import java.io.FileReader;
9. import java.io.BufferedWriter;
10.    import java.io.FileWriter;
11.    import java.io.IOException;
12.    import java.time.*;
13.    import java.time.format.DateTimeFormatter;
14.
15.    class Schedule
16.    {
17.          private ArrayList<ArrayList<ArrayList<ArrayList>>>
   schedules = new ArrayList<ArrayList<ArrayList<ArrayList>>>();
18.          private ArrayList<ArrayList<ArrayList>> schedule,
   child, currentstate, bestschedule;
19.          private ArrayList<ArrayList> activity;
20.          private ArrayList division, temp, randomnumbers,
   checkedmember, cons1, cons2, cons3, cons4;
21.
22.          Database database = new Database();
23.          BufferedReader reader;
24.          String line, join;
25.          String[] record;
26.          int random, random2, membercount, fitness, newfitness,
   schedulecount, parents, maxserve, overserve, divmembercount,
   activitycount, selectedfitness, selectedfitnessind,
   selectedfitnessindex, divcount, maxcrossover, crossovercount,
   mutationrate, bestfitness, bestfitnessindex, currentfitness,
   noupdate, currentstatefitness, newstatefitness, delete,
   deleteindex, loop, c,i,j,k,l,m,n,o,p,f;
27.          Boolean found, complete, optimized;
28.
29.          DateTimeFormatter dateformat =
   DateTimeFormatter.ofPattern("yyyy-MM-dd");
30.          DateTimeFormatter datetimeformat =
   DateTimeFormatter.ofPattern("yyyy-MM-dd HH_mm_ss");
31.          LocalDate date;
32.          LocalTime time;
33.          LocalDateTime activity1, activity2, endactivity1,
   startactivity2, now, dbcsvtime, schcsvtime, runtime;
34.          long starttime, finishtime, timeelapsed,
   totaltimeelapsed;
35.
36.          String dbfile = "database.csv";
37.          String schfile = "schedule.csv";
```

```
38.
39.         //this is a default constructor
40.         Schedule()
41.         {
42.
43.         }
44.
45.         //methods begin here
46.
47.         public void ReadScheduleCSV()
48.         {
49.               try
50.               {
51.                     //read the schedule.csv
52.                     reader = new BufferedReader(new
  FileReader(schfile));
53.                     while((line = reader.readLine()) !=
  null) //as long as the current line is not null do ...
54.                     {
55.                           record = line.split(","); //store
  the activity records in array record[]
56.                           activity = new
  ArrayList<ArrayList>();
57.                           division = new ArrayList<String>();
58.                           if(record.length == 2) //for the
  first line that contains schedule title
59.                           {
60.                                 division.add(record[1]);
  //store the schedule title to division(0) activity(0)
61.                                 division.add("0");
  //initialize total fitness value and store it to division(1)
  activity(0)
62.                                 division.add("0"); //fitness
  value constraint 1
63.                                 division.add("0"); //fitness
  value constraint 2
64.                                 division.add("0"); //fitness
  value constraint 3
65.                                 division.add("0"); //fitness
  value constraint 4
66.                                 activity.add(division);
67.                           }
68.                           else //for the second line onwards
69.                           {
70.                                 division.add(record[0]);
71.                                 division.add(record[1]);
72.                                 division.add(record[2]);
73.                                 division.add(record[3]);
74.                                 activity.add(division);
75.                                 for(i=4; i<record.length; i++)
76.                                 {
77.                                       division = new
  ArrayList<String>();
78.                                       division.add(record[i]);
```

```
79.                                         activity.add(division);
80.                                   }
81.                             }
82.                             schedule.add(activity);
83.                       }
84.                       reader.close();
85.                 }
86.                 catch(Exception e)
87.                 {
88.                       System.out.println("Error : " + e);
89.                 }
90.           }
91.
92.           public void RandomizeMember()
93.           {
94.                 for(i=0; i<schedule.size(); i++)
95.                 {
96.                       for(j=1; j<schedule.get(i).size(); j++)
97.                       {
98.                             line =
   (String)schedule.get(i).get(j).get(0);
99.                             record = line.split("[.]");
100.                            membercount =
   Integer.parseInt(record[1]);
101.                            randomnumbers = new
   ArrayList<String>();
102.                            for(k=0; k<membercount; k++)
103.                            {
104.                                  random = (int)
   (Math.random()*database.GetDivisionSize(record[0]));
105.
      while(randomnumbers.contains(random))
106.                                  {
107.                                        random = (int)
   (Math.random()*database.GetDivisionSize(record[0]));
108.                                  }
109.                                 randomnumbers.add(random);
110.                                 if(random == 0)
111.                                 {
112.                                       random = random + 1;
113.                                 }
114.

   schedule.get(i).get(j).add(database.GetDivisionMember(record[0]
   , random));
115.
      Collections.sort(schedule.get(i).get(j).subList(1,
   schedule.get(i).get(j).size()));
116.                            }
117.                      }
118.                }
119.          }
120.
```

```
121.        public ArrayList<ArrayList<ArrayList>>
  GenerateRandomSchedule()
122.        {
123.              schedule = new
  ArrayList<ArrayList<ArrayList>>();
124.              ReadDatabaseCSV();
125.              ReadScheduleCSV();
126.              RandomizeMember();
127.              return schedule;
128.        }
129.
130.        public ArrayList<ArrayList<ArrayList>>
  EvaluateSchedule(ArrayList<ArrayList<ArrayList>> schedule)
131.        {
132.              for(f=1; f<6; f++)
133.              {
134.                    schedule.get(0).get(0).set(f,
  Integer.toString(0)); //initialize all fitness value to 0
135.              }
136.
137.              //Constraint 1 : In the same activity and same
  division, no double names allowed
138.              cons1 = new ArrayList<String>();
139.              temp = new ArrayList<String>();
140.              for(i=1; i<schedule.size(); i++)
141.              {
142.                    for(j=1; j<schedule.get(i).size(); j++)
143.                    {
144.                          Set<String> set = new
  HashSet<String>(schedule.get(i).get(j));
145.                          if(set.size() <
  schedule.get(i).get(j).size())
146.                          {
147.                                newfitness =
  Integer.parseInt((String)schedule.get(0).get(0).get(2)) + 1;
148.                                schedule.get(0).get(0).set(2,
  Integer.toString(newfitness));
149.
      cons1.addAll(schedule.get(i).get(j));
150.                                temp.addAll(set);
151.                                for(f=0; f<temp.size(); f++)
152.                                {
153.
      cons1.remove(temp.get(f));
154.                                }
155.                          }
156.                    }
157.              }
158.
159.              //Constraint 2 : In the same activity, no member
  that can serve in 2 division
160.              cons2 = new ArrayList<String>();
161.              for(i=1; i<schedule.size(); i++)
162.              {
```

```
163.                     for(j=1; j<schedule.get(i).size()-1; j++)
164.                     {
165.                           for(k=j+1; k<schedule.get(i).size()-
  1; k++)
166.                           {
167.                                 temp = new
  ArrayList<String>(schedule.get(i).get(j));
168.
      temp.retainAll(schedule.get(i).get(k));
169.                                 if(temp.size() != 0)
170.                                 {
171.                                       newfitness =
  Integer.parseInt((String)schedule.get(0).get(0).get(3)) + 1;
172.
      schedule.get(0).get(0).set(3, Integer.toString(newfitness));
173.
      Collections.addAll(cons2, temp);
174.                                 }
175.                           }
176.                     }
177.               }
178.
179.               //Constraint 3 : A member cannot serve in two
  activities at the same time
180.               cons3 = new ArrayList<String>();
181.               for(i=1; i<schedule.size()-1; i++)
182.               {
183.                     for(j=i+1; j<schedule.size(); j++)
184.                     {
185.                           activity1 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(i).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(i).get(0).get(2)));
186.                           //
  System.out.println(schedule.get(j).get(0).get(1));
187.                           //
  System.out.println(schedule.get(j).get(0).get(2));
188.                           activity2 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(j).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(j).get(0).get(2)));
189.                           if(activity1.isBefore(activity2))
190.                           {
191.                                 endactivity1 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(i).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(i).get(0).get(3)));
192.                                 startactivity2 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(j).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(j).get(0).get(2)));
193.                           }
194.                           else
195.                           {
```

```
196.                                    endactivity1 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(j).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(i).get(0).get(3)));
197.                                    startactivity2 =
  LocalDateTime.of(LocalDate.parse((String)schedule.get(i).get(0)
  .get(1), dateformat),
  LocalTime.parse((String)schedule.get(i).get(0).get(2)));
198.                                }
199.                                //check if the activity collides...
200.
      if(endactivity1.isBefore(startactivity2))
201.                                {
202.                                    //no activity collsion,
  ignore...
203.                                }
204.                                else
205.                                {
206.                                    //activity collision, check
  the members...
207.                                    ArrayList members1 = new
  ArrayList<String>();
208.                                    ArrayList members2 = new
  ArrayList<String>();
209.                                    for(k=1;
  k<schedule.get(i).size(); k++)
210.                                    {
211.                                        ArrayList temp = new
  ArrayList<String>();
212.
      temp.addAll(schedule.get(i).get(k));
213.                                        temp.remove(0);
214.                                        members1.addAll(temp);
215.                                    }
216.                                    for(k=1;
  k<schedule.get(j).size(); k++)
217.                                    {
218.                                        ArrayList temp = new
  ArrayList<String>();
219.
      temp.addAll(schedule.get(j).get(k));
220.                                        temp.remove(0);
221.                                        members2.addAll(temp);
222.                                    }
223.                                    members1.retainAll(members2);
224.                                    if(members1.size() != 0)
225.                                    {
226.                                        //there are members that
  serve in the two activities at the sametime, update fitness
  value
227.                                        newfitness =
  Integer.parseInt((String)schedule.get(0).get(0).get(4)) + 1;
228.
      schedule.get(0).get(0).set(4, Integer.toString(newfitness));
```

```
                                        cons3.addAll(members1);
                                    }
                                }
                            }
                        }

                        //Constraint 4 : Each member can only serve a certain number of activity
                        //Formula : max serve = (number of slots in the division * number of activity) / number of members available in the division

                        // Step 1 : Store each division and its members in every activity to a new temporary 2D arraylist
                        ArrayList<ArrayList> checkmaxserve = new ArrayList<ArrayList>(); //temporary 2D arraylist
                        for(i=1; i<schedule.size(); i++)
                        {
                            for(j=1; j<schedule.get(i).size(); j++)
                            {
                                found = false;
                                line = (String)schedule.get(i).get(j).get(0);
                                record = line.split("[.]");
                                for(k=0; k<checkmaxserve.size(); k++) //check if a division exists in the checkmaxserve
                                {

                                    if(checkmaxserve.get(k).contains(record[0])) //if the division exists, add the members
                                    {
                                        found = true;
                                        ArrayList temp = new ArrayList<String>();

                                        temp.addAll(schedule.get(i).get(j));
                                        temp.set(0, record[0]);

                                        checkmaxserve.get(k).addAll(temp);
                                        break;
                                    }
                                }
                                if(found == false) //if the division doesn't exists, add the new division
                                {
                                    ArrayList temp = new ArrayList<String>();

                                    temp.addAll(schedule.get(i).get(j));
                                    temp.set(0, record[0]);
                                    checkmaxserve.add(temp);
                                }
                            }
                        }
```

```
268.
269.              //Step 2 : Check every member frequency in each
  division in the checkmaxserve
270.              //if frequency > max serve, increase the fitness
  value
271.              //store every checked member to arraylist
  checkedmember, so every member will only checked one time
272.              cons4 = new ArrayList<String>();
273.              checkedmember = new ArrayList<String>();
274.              for(i=0; i<checkmaxserve.size(); i++)
275.              {
276.                   activitycount =
  Collections.frequency(checkmaxserve.get(i),
  checkmaxserve.get(i).get(0));
277.                   divmembercount =
  database.GetDivisionSize((String)checkmaxserve.get(i).get(0)) -
  1;
278.                   // maxserve =
  (int)Math.ceil((checkmaxserve.get(i).size() - activitycount) /
  (float)divmembercount);
279.                   maxserve = (checkmaxserve.get(i).size() -
  activitycount) / divmembercount + 1;
280.                   if(maxserve > 0)
281.                   {
282.
283.                   }
284.                   else
285.                   {
286.                        maxserve = 1;
287.                   }
288.                   for(j=0; j<checkmaxserve.get(i).size(); j+
  +)
289.                   {
290.
      checkedmember.add(checkmaxserve.get(i).get(0));
291.
      if(checkedmember.contains(checkmaxserve.get(i).get(j)) ==
  true)
292.                        {
293.                             //member already checked,
  skip...
294.                        }
295.                        else
296.                        {
297.
      if(Collections.frequency(checkmaxserve.get(i),
  checkmaxserve.get(i).get(j)) > maxserve)
298.                             {
299.                                  newfitness =
  Integer.parseInt((String)schedule.get(0).get(0).get(5)) + 1;
300.
      schedule.get(0).get(0).set(5, Integer.toString(newfitness));
301.
      checkedmember.add(checkmaxserve.get(i).get(j));
```

```
302.
      cons4.add(checkmaxserve.get(i).get(j));
303.                                    }
304.
      checkedmember.add(checkmaxserve.get(i).get(j));
305.                               }
306.                         }
307.                }
308.                //Count total fitness value
309.                newfitness = 0;
310.                for(f=2; f<6; f++)
311.                {
312.                      newfitness = newfitness +
  Integer.parseInt((String)schedule.get(0).get(0).get(f));
313.                }
314.                schedule.get(0).get(0).set(1,
  Integer.toString(newfitness));
315.                return schedule;
316.          }
317.
318.          public void GenerateRandomPopulation(int population)
319.          {
320.                for(m=0; m<population; m++)
321.                {
322.                      schedule = GenerateRandomSchedule();
323.                      schedules.add(schedule);
324.                }
325.          }
326.
327.          public void EvaluateScheduleGenetic()
328.          {
329.                for(n=0; n<schedules.size(); n++)
330.                {
331.                      EvaluateSchedule(schedules.get(n));
332.                }
333.          }
334.
335.          public void CrossOverMutationSchedule()
336.          {
337.                //CROSSOVER
338.                //Step 1 : Select 2 schedules with the best
  fitness value as parent
339.                ArrayList fitnessvalue = new
  ArrayList<Integer>();
340.                ArrayList selectedfitnessindex = new
  ArrayList<Integer>();
341.                for(i=0; i<schedules.size(); i++)
342.                {
343.

  fitnessvalue.add(Integer.parseInt((String)schedules.get(i).get(
  0).get(0).get(1)));
344.                }
345.                schedulecount = schedules.size();
```

```
346.                if(schedulecount >= 4)
347.                {
348.                      parents = schedulecount / 2;
349.                      if(parents%2 != 0)
350.                      {
351.                            parents--;
352.                      }
353.                }
354.                else
355.                {
356.                      parents = 2;
357.                }
358.                while(selectedfitnessindex.size() < parents)
359.                {
360.                      selectedfitness =
  (int)Collections.min(fitnessvalue);
361.                      selectedfitnessind =
  fitnessvalue.indexOf(selectedfitness);
362.
      selectedfitnessindex.add((int)selectedfitnessind);
363.                      fitnessvalue.set(selectedfitnessind,
  1000); //set the current best fitness value to 1000 so it will
  not be selected again
364.                }
365.                System.out.println(selectedfitnessindex.size());
366.                System.out.println(selectedfitnessindex);
367.
368.                //Step 2 : Create childs schedule based on the
  parents
369.                ArrayList<ArrayList<ArrayList<ArrayList>>>
  childs = new ArrayList<ArrayList<ArrayList<ArrayList>>>();
370.
371.                for(c=0; c<parents; c++)
372.                {
373.                      child = new
  ArrayList<ArrayList<ArrayList>>();
374.                      // System.out.println("child : " + child);
375.                      for(i=0;
  i<schedules.get((int)selectedfitnessindex.get(c)).size(); i++)
376.                      {
377.                            activity = new
  ArrayList<ArrayList>();
378.                            for(j=0;
  j<schedules.get((int)selectedfitnessindex.get(c)).get(i).size()
  ; j++)
379.                            {
380.                                  division = new
  ArrayList<String>();
381.                                  for(k=0;
  k<schedules.get((int)selectedfitnessindex.get(c)).get(i).get(j)
  .size(); k++)
382.                                  {
383.
```

```
  division.add(schedules.get((int)selectedfitnessindex.get(c)).ge
  t(i).get(j).get(k));
384.                                    }
385.                                    activity.add(division);
386.                              }
387.                              child.add(activity);
388.                        }
389.                        childs.add(child);
390.                  }
391.
392.                  for(c=0; c<parents/2; c++)
393.                  {
394.                        //Step 3 : Count the number of divisions
  and set how many division will crossover
395.                        //Crossover level : division
396.                        divcount = 0;
397.                        for(i=0; i<childs.get(0).size(); i++)
398.                        {
399.                              divcount = divcount +
  childs.get(0).get(i).size();
400.                        }
401.                        maxcrossover = (int)
  (Math.random()*divcount);
402.
403.                        //Step 4 : Crossover those divisions in
  childs index 0 and 1
404.                        crossovercount = 0;
405.                        randomnumbers = new ArrayList<String>();
406.                        while(crossovercount != maxcrossover)
407.                        {
408.                              i = (int)
  (Math.random()*childs.get(0).size());
409.                              j = (int)
  (Math.random()*childs.get(0).get(i).size());
410.
411.                              //Rerandom if random result repeated
412.                              while(randomnumbers.contains(i + ","
  + j))
413.                              {
414.                                    i = (int)
  (Math.random()*childs.get(0).size());
415.                                    j = (int)
  (Math.random()*childs.get(0).get(i).size());
416.                              }
417.                              randomnumbers.add(i + "," + j);
418.
419.                              ArrayList temp = new
  ArrayList<String>();
420.
      temp.addAll(childs.get(0).get(i).get(j));
421.                              childs.get(0).get(i).get(j).clear();
422.
```

```
childs.get(0).get(i).get(j).addAll(childs.get(1).get(i).get(j))
;
                        childs.get(1).get(i).get(j).clear();

    childs.get(1).get(i).get(j).addAll(temp);
                        crossovercount++;
                    }
                    //
DisplaySchedule(schedules.get((int)selectedfitnessindex.get(0))
);
                    //
DisplaySchedule(schedules.get((int)selectedfitnessindex.get(1))
);
                    // DisplaySchedule(childs.get(0));
                    // DisplaySchedule(childs.get(1));

                    mutationrate = 30; //percent
                    //MUTATION for childs index 0
                    for(i=0; i<childs.get(0).size(); i++)
                    {
                        for(j=1;
j<childs.get(0).get(i).size(); j++)
                        {
                            for(k=1;
k<childs.get(0).get(i).get(j).size(); k++)
                            {
                                random2 = (int)
(Math.random()*100);
                                if(random2 <= 30)
                                {
                                    //mutation occurs
                                    line =
(String)childs.get(0).get(i).get(j).get(0);
                                    record =
line.split("[.]");
                                    random = (int)
(Math.random()*database.GetDivisionSize(record[0]));
                                    if(random == 0)
                                    {
                                        random =
random + 1;
                                    }

    childs.get(0).get(i).get(j).set(k,
database.GetDivisionMember(record[0], random));
                                }
                            }

    Collections.sort(childs.get(0).get(i).get(j).subList(1,
childs.get(0).get(i).get(j).size()));
                        }
                    }
                    //MUTATION for childs index 1
                    // for(i=0; i<childs.get(1).size(); i++)
```

```
459.                    // {
460.                    //    for(j=1;
  j<childs.get(1).get(i).size(); j++)
461.                    //    {
462.                    //          for(k=1;
  k<childs.get(1).get(i).get(j).size(); k++)
463.                    //          {
464.                    //                random2 = (int)
  (Math.random()*100);
465.                    //                if(random2 <= 30)
466.                    //                {
467.                    //                      //mutation occurs
468.                    //                      line =
  (String)childs.get(1).get(i).get(j).get(0);
469.                    //                      record =
  line.split("[.]");
470.                    //                      random = (int)
  (Math.random()*database.GetDivisionSize(record[0]));
471.                    //                      if(random == 0)
472.                    //                      {
473.                    //                            random =
  random + 1;
474.                    //                      }
475.                    //
      childs.get(1).get(i).get(j).set(k,
  database.GetDivisionMember(record[0], random));
476.                    //                }
477.                    //          }
478.                    //
     Collections.sort(childs.get(1).get(i).get(j).subList(1,
  childs.get(1).get(i).get(j).size()));
479.                    //    }
480.                    // }
481.                    //ADD both childs into the population
482.                    schedules.add(childs.get(0));
483.                    schedules.add(childs.get(1));
484.                    //DELETE both childs in index 0 and 1 so
  the next 2 child will have index 0 and 1 again
485.                    childs.remove(0);
486.                    childs.remove(0);
487.              }
488.        }
489.
490.        public void FilterPopulation()
491.        {
492.              for(l=0; l<parents; l++) //Looping 2 times to
  delete 2 schedules with worst fitness value
493.              {
494.                    ArrayList fitnessvalue = new
  ArrayList<Integer>();
495.                    for(m=0; m<schedules.size(); m++)
496.                    {
497.
```

```
  fitnessvalue.add(Integer.parseInt((String)schedules.get(m).get(
  0).get(0).get(1)));
498.                    }
499.                    delete =
  (int)Collections.max(fitnessvalue);
500.                    deleteindex =
  fitnessvalue.indexOf(delete);
501.                    schedules.remove(deleteindex);
502.              }
503.        }
504.
505.        public void GeneticAlgorithm()
506.        {
507.              complete = false;
508.              loop = 0;
509.              noupdate = 0;
510.              bestfitness = 1000; //for evaluation
511.              currentfitness = 1000; //for evaluation
512.              runtime = LocalDateTime.of(LocalDate.now(),
  LocalTime.now()); //for evaluation
513.              //for evaluation
514.              try
515.              {
516.                    BufferedWriter writer = new
  BufferedWriter(new FileWriter("GeneticLog_" +
  runtime.format(datetimeformat) + ".txt" , true));
517.                    writer.write("Genetic Algorithm begins at
  " + runtime.format(datetimeformat) + "\n");
518.                    writer.close();
519.              }
520.              catch(Exception e)
521.              {
522.                    System.out.println("Error : " + e);
523.              }
524.              while(complete == false)
525.              {
526.                    loop++;
527.                    EvaluateScheduleGenetic();
528.                    ArrayList fitnessvalue = new
  ArrayList<Integer>();
529.                    for(o=0; o<schedules.size(); o++)
530.                    {
531.

  fitnessvalue.add(Integer.parseInt((String)schedules.get(o).get(
  0).get(0).get(1)));
532.                    }
533.                    System.out.println("Generation : " + loop
  + " Fitness : " + fitnessvalue);
534.                    currentfitness =
  (int)Collections.min(fitnessvalue); //for evaluation
535.
536.                    //IF BEST SCHEDULE IS ACCIDENTALLY
  REPLACED
```

```
537.                     if(bestfitness < currentfitness)
538.                     {
539.                           delete =
  (int)Collections.max(fitnessvalue);
540.                           deleteindex =
  fitnessvalue.indexOf(delete);
541.                           schedules.remove(deleteindex);
542.
543.                           schedule = new
  ArrayList<ArrayList<ArrayList>>();
544.                           for(i=0; i<bestschedule.size(); i++)
545.                           {
546.                                 activity = new
  ArrayList<ArrayList>();
547.                                 for(j=0;
  j<bestschedule.get(i).size(); j++)
548.                                 {
549.                                       division = new
  ArrayList<String>();
550.                                       for(k=0;
  k<bestschedule.get(i).get(j).size(); k++)
551.                                       {
552.
    division.add(bestschedule.get(i).get(j).get(k));
553.                                       }
554.                                       activity.add(division);
555.                                 }
556.                                 schedule.add(activity);
557.                           }
558.                           schedules.add(schedule);
559.                     }
560.
561.                     //for evaluation
562.                     if(currentfitness < bestfitness)
563.                     {
564.                           now =
  LocalDateTime.of(LocalDate.now(), LocalTime.now());
565.                           bestfitness = currentfitness;
566.                           bestfitnessindex =
  fitnessvalue.indexOf(bestfitness);
567.
568.                           //BACKUP BEST SCHEDULE EVER FOUND
569.                           bestschedule = new
  ArrayList<ArrayList<ArrayList>>();
570.                           for(i=0;
  i<schedules.get(bestfitnessindex).size(); i++)
571.                           {
572.                                 activity = new
  ArrayList<ArrayList>();
573.                                 for(j=0;
  j<schedules.get(bestfitnessindex).get(i).size(); j++)
574.                                 {
575.                                       division = new
  ArrayList<String>();
```

```
576.                                        for(k=0;
  k<schedules.get(bestfitnessindex).get(i).get(j).size(); k++)
577.                                        {
578.
  division.add(schedules.get(bestfitnessindex).get(i).get(j).get(
  k));
579.                                        }
580.                                        activity.add(division);
581.                                  }
582.                                  bestschedule.add(activity);
583.                            }
584.
585.                            ArrayList constfitnessvalue = new
  ArrayList<Integer>();
586.                            for(i=2; i<6; i++)
587.                            {
588.
  constfitnessvalue.add(Integer.parseInt((String)schedules.get(be
  stfitnessindex).get(0).get(0).get(i)));
589.                            }
590.
591.                            noupdate = 0;
592.
      WriteDetailedLog(runtime.format(datetimeformat), "Genetic",
  now.format(datetimeformat), loop, bestfitness,
  constfitnessvalue);
593.                      }
594.                      noupdate++;
595.                      if(fitnessvalue.contains(0) || loop >
  5000000 || noupdate > 2000000)
596.                      {
597.                            complete = true;
598.                            break;
599.                      }
600.                      else
601.                      {
602.                            CrossOverMutationSchedule();
603.                            FilterPopulation();
604.                      }
605.                }
606.          }
607.
608.          public void SAHCAlgorithm()
609.          {
610.                optimized = false;
611.                loop = 0;
612.                noupdate = 0;
613.                bestfitness = 1000; //for evaluation
614.                runtime = LocalDateTime.of(LocalDate.now(),
  LocalTime.now()); //for evaluation
615.                currentstate = new
  ArrayList<ArrayList<ArrayList>>(GenerateRandomSchedule());
```

```
616.              currentstate = EvaluateSchedule(currentstate);
617.              currentstatefitness =
  Integer.parseInt((String)currentstate.get(0).get(0).get(1));
618.              try
619.              {
620.                    BufferedWriter writer = new
  BufferedWriter(new FileWriter("SAHCLog_" +
  runtime.format(datetimeformat) + ".txt" , true));
621.                    writer.write("SAHC Algorithm begins at " +
  runtime.format(datetimeformat) + "\n");
622.                    writer.close();
623.              }
624.              catch(Exception e)
625.              {
626.                    System.out.println("Error : " + e);
627.              }
628.              if(currentstatefitness == 0)
629.              {
630.                    optimized = true;
631.              }
632.              while(optimized == false)
633.              {
634.                    loop++;
635.                    currentstate =
  EvaluateSchedule(currentstate);
636.                    currentstatefitness =
  Integer.parseInt((String)currentstate.get(0).get(0).get(1));
637.                    System.out.println("Generation : " + loop
  + " Fitness : " + currentstatefitness);
638.                    if(currentstatefitness == 0 || loop >
  5000000 || noupdate > 2000000)
639.                    {
640.                          optimized = true;
641.                          break;
642.                    }
643.                    ArrayList<ArrayList<ArrayList>> newstate =
  new ArrayList<ArrayList<ArrayList>>(GenerateRandomSchedule());
644.                    newstate = EvaluateSchedule(newstate);
645.                    newstatefitness =
  Integer.parseInt((String)newstate.get(0).get(0).get(1));
646.
647.                    if(currentstatefitness <= newstatefitness)
648.                    {
649.                          //current state is better than
  newstate, do nothing...
650.                          noupdate++;
651.                    }
652.                    else
653.                    {
654.                          currentstate.clear();
655.                          for(i=0; i<newstate.size(); i++)
656.                          {
657.                                activity = new
  ArrayList<ArrayList>();
```

```
658.                                    for(j=0;
  j<newstate.get(i).size(); j++)
659.                                    {
660.                                          division = new
  ArrayList<String>();
661.                                          for(k=0;
  k<newstate.get(i).get(j).size(); k++)
662.                                          {
663.
      division.add(newstate.get(i).get(j).get(k));
664.                                          }
665.                                          activity.add(division);
666.                                    }
667.                                    currentstate.add(activity);
668.                              }
669.                              //for evaluation
670.                              now =
  LocalDateTime.of(LocalDate.now(), LocalTime.now());
671.                              ArrayList constfitnessvalue = new
  ArrayList<Integer>();
672.                              for(i=2; i<6; i++)
673.                              {
674.
  constfitnessvalue.add(Integer.parseInt((String)currentstate.get
  (0).get(0).get(i)));
675.                              }
676.
677.                              noupdate = 0;
678.
      WriteDetailedLog(runtime.format(datetimeformat), "SAHC",
  now.format(datetimeformat), loop, newstatefitness,
  constfitnessvalue);
679.                         }
680.                    }
681.               }
682.
683.          public void
  DisplaySchedule(ArrayList<ArrayList<ArrayList>> schedule)
684.          {
685.               System.out.println("Schedule Title : " +
  schedule.get(0).get(0).get(0));
686.               System.out.println("Fitness Value : " +
  schedule.get(0).get(0).get(1));
687.               for(i=1; i<schedule.size(); i++)
688.               {
689.
  System.out.println("==========================================
  ==========");
690.                    System.out.println();
691.                    System.out.println("Activity Title : " +
  schedule.get(i).get(0).get(0));
```

```
692.                     System.out.println("Date : " +
  schedule.get(i).get(0).get(1));
693.                     System.out.println("Start Time : " +
  schedule.get(i).get(0).get(2));
694.                     System.out.println("Finish Time : " +
  schedule.get(i).get(0).get(3));
695.                     for(j=1; j<schedule.get(i).size(); j++)
696.                     {
697.                           line =
  (String)schedule.get(i).get(j).get(0);
698.                           record = line.split("[.]");
699.                           System.out.println(record[0] +
  " :");
700.                           for(k=1;
  k<schedule.get(i).get(j).size(); k++)
701.                           {
702.                                 System.out.println(k + ". " +
  schedule.get(i).get(j).get(k));
703.                           }
704.                           System.out.println();
705.                     }
706.                     System.out.println();
707.               }
708.         }
709.
710.         public void DisplayPopulation()
711.         {
712.               System.out.println("Number of Schedule(s) : " +
  schedules.size());
713.               for(l=0; l<schedules.size(); l++)
714.               {
715.                     DisplaySchedule(schedules.get(l));
716.               }
717.         }
718.
719.         public void DisplayScheduleSAHC()
720.         {
721.               DisplaySchedule(currentstate);
722.         }
723.
724.         public void GeneticAlgorithmMultiple(int population,
  int numofschedules)
725.         {
726.               for(p=0; p<numofschedules; p++)
727.               {
728.                     schedules.clear();
729.                     starttime = System.currentTimeMillis();
730.                     GenerateRandomPopulation(population);
731.                     GeneticAlgorithm();
732.                     finishtime = System.currentTimeMillis();
733.                     timeelapsed = finishtime - starttime;
734.                     totaltimeelapsed = totaltimeelapsed +
  timeelapsed;
735.                     WriteLog("Genetic", 1, timeelapsed);
```

```
736.                     WriteErrorLog("Genetic", cons1, cons2,
  cons3, cons4);
737.                     SaveToTxtFileGenetic();
738.               }
739.         }
740.
741.         public void SAHCAlgorithmMultiple(int numofschedules)
742.         {
743.               for(p=0; p<numofschedules; p++)
744.               {
745.                     starttime = System.currentTimeMillis();
746.                     SAHCAlgorithm();
747.                     finishtime = System.currentTimeMillis();
748.                     timeelapsed = finishtime - starttime;
749.                     totaltimeelapsed = totaltimeelapsed +
  timeelapsed;
750.                     WriteLog("SAHC", 1, timeelapsed);
751.                     WriteErrorLog("SAHC", cons1, cons2, cons3,
  cons4);
752.                     SaveToTxtFileSAHC();
753.               }
754.         }
755.
756.         public void SaveToTxtFileGenetic()
757.         {
758.               ArrayList fitnessvalue = new
  ArrayList<Integer>();
759.               now = LocalDateTime.of(LocalDate.now(),
  LocalTime.now());
760.               for(i=0; i<schedules.size(); i++)
761.               {
762.
  fitnessvalue.add(Integer.parseInt((String)schedules.get(i).get(
  0).get(0).get(1)));
763.                     selectedfitness =
  (int)Collections.min(fitnessvalue);
764.                     selectedfitnessind =
  fitnessvalue.indexOf(selectedfitness);
765.               }
766.               try
767.               {
768.                     BufferedWriter writer = new
  BufferedWriter(new FileWriter("GeneticLog_" +
  runtime.format(datetimeformat) + ".txt", true));
769.                     writer.write("\nSchedule Title : " +
  schedules.get(selectedfitnessind).get(0).get(0).get(0) + "\n");
770.                     writer.write("Fitness Value : " +
  schedules.get(selectedfitnessind).get(0).get(0).get(1) + "\n");
771.                     for(j=1;
  j<schedules.get(selectedfitnessind).size(); j++)
772.                     {
773.
```

```
writer.write("================================================
====\n");
                        writer.write("\n");
                        writer.write("Activity Title : " +
schedules.get(selectedfitnessind).get(j).get(0).get(0) + "\n");
                        writer.write("Date : " +
schedules.get(selectedfitnessind).get(j).get(0).get(1) + "\n");
                        writer.write("Start Time : " +
schedules.get(selectedfitnessind).get(j).get(0).get(2) + "\n");
                        writer.write("Finish Time : " +
schedules.get(selectedfitnessind).get(j).get(0).get(3) + "\n");
                        writer.write("\n");
                        for(k=1;
k<schedules.get(selectedfitnessind).get(j).size(); k++)
                        {
                            line =
(String)schedules.get(selectedfitnessind).get(j).get(k).get(0);
                            record = line.split("[.]");
                            writer.write(record[0] + " :\
n");
                            for(l=1;
l<schedules.get(selectedfitnessind).get(j).get(k).size(); l++)
                            {
                                writer.write(l + ". " +
schedules.get(selectedfitnessind).get(j).get(k).get(l) + "\n");
                            }
                            writer.write("\n");
                        }
                        writer.write("\n");
                    }
                    writer.close();
                }
                catch(Exception e)
                {
                    System.out.println("Error : " + e);
                }
        }

        public void SaveToTxtFileSAHC()
        {
                try
                {
                    now = LocalDateTime.of(LocalDate.now(),
LocalTime.now());
                    BufferedWriter writer = new
BufferedWriter(new FileWriter("SAHCLog_" +
runtime.format(datetimeformat) + ".txt", true));
                    writer.write("\nSchedule Title : " +
currentstate.get(0).get(0).get(0) + "\n");
                    writer.write("Fitness Value : " +
currentstate.get(0).get(0).get(1) + "\n");
                    for(j=1; j<currentstate.size(); j++)
                    {
```

```
811.
  writer.write("==============================================
  ====\n");
812.                          writer.write("\n");
813.                          writer.write("Activity Title : " +
  currentstate.get(j).get(0).get(0) + "\n");
814.                          writer.write("Date : " +
  currentstate.get(j).get(0).get(1) + "\n");
815.                          writer.write("Start Time : " +
  currentstate.get(j).get(0).get(2) + "\n");
816.                          writer.write("Finish Time : " +
  currentstate.get(j).get(0).get(3) + "\n");
817.                          writer.write("\n");
818.                          for(k=1;
  k<currentstate.get(j).size(); k++)
819.                          {
820.                               line =
  (String)currentstate.get(j).get(k).get(0);
821.                               record = line.split("[.]");
822.                               writer.write(record[0] + " :\
  n");
823.                               for(l=1;
  l<currentstate.get(j).get(k).size(); l++)
824.                               {
825.                                    writer.write(l + ". " +
  currentstate.get(j).get(k).get(l) + "\n");
826.                               }
827.                               writer.write("\n");
828.                          }
829.                          writer.write("\n");
830.                     }
831.                     writer.close();
832.                }
833.                catch(Exception e)
834.                {
835.                     System.out.println("Error : " + e);
836.                }
837.           }
838.
839.           public void WriteLog(String algorithm, int
  numofschedules, long timeinms)
840.           {
841.                try
842.                {
843.                     now = LocalDateTime.of(LocalDate.now(),
  LocalTime.now());
844.                     dbcsvtime =
  LocalDateTime.ofInstant(Instant.ofEpochMilli(new
  File(dbfile).lastModified()), ZoneId.systemDefault());
845.                     schcsvtime =
  LocalDateTime.ofInstant(Instant.ofEpochMilli(new
  File(schfile).lastModified()), ZoneId.systemDefault());
```

```
846.                    BufferedWriter writer = new
  BufferedWriter(new FileWriter(algorithm + "Log_" +
  runtime.format(datetimeformat) + ".txt", true));
847.                    writer.write(now.format(datetimeformat) +
  " : " + "Created " + numofschedules + " variants of schedules
  with " + algorithm + " algorithm based on database.csv (updated
  : " + dbcsvtime + ") and schedule.csv (updated : " + schcsvtime
  + "). Time elapsed : " + timeinms + "ms \n");
848.                    writer.close();
849.             }
850.             catch(Exception e)
851.             {
852.                    System.out.println("Error : " + e);
853.             }
854.       }
855.
856.       public void WriteDetailedLog(String runtime, String
  algorithm, String time, int generation, int fitnessvalue,
  ArrayList constfitnessvalue)
857.       {
858.             if(algorithm == "Genetic")
859.             {
860.                    try
861.                    {
862.                           BufferedWriter writer = new
  BufferedWriter(new FileWriter("GeneticLog_" + runtime +
  ".txt" , true));
863.                           writer.write(time + " : " + "
  Generation = " + generation + " Fitness Value = " +
  fitnessvalue + " " + constfitnessvalue + "\n");
864.                           writer.close();
865.                    }
866.                    catch(Exception e)
867.                    {
868.                           System.out.println("Error : " + e);
869.                    }
870.             }
871.             if(algorithm == "SAHC")
872.             {
873.                    try
874.                    {
875.                           BufferedWriter writer = new
  BufferedWriter(new FileWriter("SAHCLog_" + runtime + ".txt",
  true));
876.                           writer.write(time + " : " + "
  Generation = " + generation + " Fitness Value = " +
  fitnessvalue + " " + constfitnessvalue + "\n");
877.                           writer.close();
878.                    }
879.                    catch(Exception e)
880.                    {
881.                           System.out.println("Error : " + e);
882.                    }
883.             }
```

```
884.        }
885.
886.        public void WriteErrorLog(String algorithm, ArrayList
  cons1, ArrayList cons2, ArrayList cons3, ArrayList cons4)
887.        {
888.              if(algorithm == "Genetic")
889.              {
890.                    try
891.                    {
892.                          BufferedWriter writer = new
  BufferedWriter(new FileWriter("GeneticLog_" +
  runtime.format(datetimeformat) + ".txt" , true));
893.                          writer.write("\nErrors : \n" +
  "Constraint 1 : " + cons1 + "\n" + "Constraint 2 : " + cons2 +
  "\n" + "Constraint 3 : " + cons3 + "\n" + "Constraint 4 : " +
  cons4 + "\n");
894.                          writer.close();
895.                    }
896.                    catch(Exception e)
897.                    {
898.                          System.out.println("Error : " + e);
899.                    }
900.              }
901.              if(algorithm == "SAHC")
902.              {
903.                    try
904.                    {
905.                          BufferedWriter writer = new
  BufferedWriter(new FileWriter("SAHCLog_" +
  runtime.format(datetimeformat) + ".txt" , true));
906.                          writer.write("\nErrors : \n" +
  "Constraint 1 : " + cons1 + "\n" + "Constraint 2 : " + cons2 +
  "\n" + "Constraint 3 : " + cons3 + "\n" + "Constraint 4 : " +
  cons4 + "\n");
907.                          writer.close();
908.                    }
909.                    catch(Exception e)
910.                    {
911.                          System.out.println("Error : " + e);
912.                    }
913.              }
914.        }
915.
916.        //method to read database.csv
917.        public void ReadDatabaseCSV()
918.        {
919.              database.ReadDatabaseCSV();
920.        }
921.  } //end of methods
```

ScheduleMain.java

```
1. import java.util.Scanner;
2.
3. class ScheduleMain
4. {
5.     public static void main(String[] args)
6.     {
7.           Schedule schedule = new Schedule();
8.
9.           int choice = 10;
10.
11.                Scanner scannumber = new Scanner(System.in);
12.                Scanner scanstring = new Scanner(System.in);
13.
14.                while(choice != 0)
15.                {
16.
   System.out.println("==========================================
   ==========");
17.                     System.out.println("Select an option : ");
18.                     System.out.println("==================
   Schedule Menu ==================");
19.                     System.out.println("1  - GENETIC
   ALGORITHM");
20.                     System.out.println("2  - SAHC ALGORITHM");
21.                     System.out.println("0  - Exit");
22.
   System.out.println("==========================================
   ==========");
23.                     System.out.println();
24.                     System.out.println("Enter 1-12 or 0 to
   exit");
25.                     choice = scannumber.nextInt();
26.                     if(choice == 1)
27.                     {
28.                          System.out.println("Enter the number
   of population(s) : ");
29.                          int population =
   scannumber.nextInt();
30.                          System.out.println("Enter the number
   of schedule(s) : ");
31.                          int numofschedules =
   scannumber.nextInt();
32.
       schedule.GeneticAlgorithmMultiple(population,
   numofschedules);
33.                     }
34.                     if(choice == 2)
35.                     {
36.                          System.out.println("Enter the number
   of schedule(s) : ");
```

```
37.                             int numofschedules =
  scannumber.nextInt();
38.
      schedule.SAHCAlgorithmMultiple(numofschedules);
39.                         }
40.                 } //end of while
41.           } //end of main
42.    } //end of class
```

FORMULIR SCAN ANTI PLAGIARISME 1,4 08 6

N a m a : Yohanes Wijaya

Alamat email : 15k10016@student.unika.ac.id

Fak. / Prodi : Ilmu Komputer / Teknik Informatika N I M : 15.K1.0016

berupa(**TESIS, TUGAS AKHIR, PROPOSAL, SKRIPSI,SUMMARY, LAPORAN KERJA PRAKTEK**)

dengan judul : Comparison Between Genetic And Hill Climbing Algorithms For Church Servant Scheduling In Hermon Community Church

Semarang, 25 Januari 2019

Petugas,	Yang Menyerahkan	Dosen Pembimbing
	Yohanes Wijaya	

NB. Laporan hasil scan terlampir

*untuk Yang bersangkutan **

Doc vs Internet + Library

98.6% Originality	1.4% Similarity	202 Sources

Web sources: 158 sources found

1. https://www.redblobgames.com/grids/hexagons 0.29%
2. http://docshare.tips/six-sigma_5888957cb6d87fe72e8b48c7.html 0.2%
3. https://repository.ipb.ac.id/bitstream/handle/123456789/52532/G11fin.pdf;sequence=1 0.2%
4. https://upcommons.upc.edu/bitstream/handle/2099.1/8931/3.Anexos.pdf?sequence=3&isAllowed=y 0.2%
5. https://apps.dtic.mil/dtic/tr/fulltext/u2/a421164.pdf 0.2%
6. https://vdocuments.site/documents/6-sigma-training-manual.html 0.2%
7. http://jtp.ub.ac.id/index.php/jtp/article/download/404/765 0.2%
8. https://allmychanges.com/p/soft/KomodoEdit 0.2%
9. http://blog.minitab.com/blog/statistics-and-quality-data-analysis/what-are-degrees-of-freedom-in-sta… 0.19%
10. http://www.machinelearning.ru/wiki/images/2/2a/Matrix-Gauss.pdf 0.19%
11. https://www.slideshare.net/konBoud/thesis-31765151 0.17%
12. http://asrjetsjournal.org/index.php/American_Scientific_Journal/article/download/1121/618 0.17%
13. https://arxiv.org/pdf/1804.04412 0.17%
14. http://vision.stanford.edu/documents/SavareseFei-Fei_ICCV2007.pdf 0.17%
15. http://www.clubcientificobezmiliana.org/blog/wp-content/uploads/2014/02/Galactosidasa.pdf 0.17%
16. https://wiryantodotblog.files.wordpress.com/2017/08/24-erwin-lim-eben-haezer-tanri-wijaya_final-p… 0.17%
17. https://www.slideshare.net/agpadilla/adecuaciones-curricularesbasadasenelanclajedelas7inteligen… 0.17%
18. http://www4.tecnun.es/asignaturas/tratamiento%20digital/tema6.pdf 0.17%
19. http://elib.spbstu.ru/dl/2234.pdf/download 0.17%
20. http://users.utcluj.ro/~mtrusca/Lab/English/lab9_ST2.pdf 0.17%
21. https://airccj.org/CSCP/vol3/csit3908.pdf 0.17%
22. https://mmspg.epfl.ch/files/content/sites/mmspl/files/shared/Semesterproject_tagpropagation.pdf 0.17%
23. http://www4.comp.polyu.edu.hk/~csajaykr/myhome/papers/ISBA2016.pdf 0.17%
24. https://arxiv.org/pdf/1802.07957.pdf 0.17%
25. https://www.tensorflow.org/xla/operation_semantics 0.17%
26. https://www.docsity.com/es/practicas-de-bioquimica/2765632 0.17%
27. https://stackoverflow.com/a/49071749 0.17%
28. http://ntuemc.tw/upload/file/201102212235523ca26.pdf 0.17%
29. https://hal.archives-ouvertes.fr/hal-01100992/document 0.17%
30. https://www.nvkc.nl/sites/default/files/NTKC/2004-1-p29-32.pdf 0.17%
31. http://folk.ntnu.no/skoge/prost/proceedings/ifac2014/media/files/1318.pdf 0.17%
32. https://patents.google.com/patent/US6165484A/en 0.17%
33. https://docplayer.nl/68468838-Autogene-en-autonome-zelfheling-van-beton.html 0.17%
34. https://arxiv.org/pdf/1801.07339 0.17%

Similarity
Similarity from a chosen source
abc Possible character replacement
Citation
References

35. https://es.slideshare.net/marylolymontanoespinoza/stress-calorico-41726554 0.17%
36. https://es.slideshare.net/JesusHidalgo4/bcr-presentacion-022015 0.17%
37. https://core.ac.uk/download/pdf/46769705.pdf 0.17%
38. https://joe-editor.sourceforge.io/4.6/man.html 0.14%
39. http://www.jimwrightonline.com/pdfdocs/cbaManual.pdf 0.13%
40. http://www.lac.lviv.ua/fileadmin/www.lac.lviv.ua/data/pidrozdily/Aspirantura/Rady/Spec_vchena_r… 0.13%
41. http://fs.onu.edu.ua/clients/client11/web11/pdf/regionalistyka_Yavor.pdf 0.13%
42. https://www.ucm.es/data/cont/docs/85-2013-11-29-Estudio%20permiso%20de%20paternidad.pdf 0.13%
43. https://www.slideshare.net/luisgenesis/solucion-de-la-agresividad-en-el-aula 0.13%
44. http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/497/3707V152.pdf?sequence=1 0.13%
45. https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/workingpapers/wp0024.pdf 0.13%
46. http://www.humankinetics.com/acucustom/sitename/K12DAM/c8d7b5cd-9e2e-4ede-968b-908830… 0.13%
47. http://www.cs.cmu.edu/~genetics/units/instructions/instructions-PGE.pdf 0.13%
48. https://www.slideshare.net/JuliaFussell/hcs-6931-capstone-final-julia-fussell 0.13%
49. http://tauja.ujaen.es/bitstream/10953.1/1038/7/TFG_SerranoRivera,SergioMiguel.pdf 0.13%
50. http://eprints.zu.edu.ua/17832/1/14.pdf 0.13%
51. https://www.ruminantia.it/wp-content/uploads/2018/06/WESTER-CANADIAN-DAIRY-SEMINAR-2… 0.13%
52. https://ag.purdue.edu/commercialag/Documents/Resources/Outlook/Livestock-Outlook/2014_02_… 0.13%
53. http://alcoy.san.gva.es/cas/hospital/sesclin/Bioequivalencia_Alcoy.pdf 0.13%
54. https://core.ac.uk/download/pdf/161233777.pdf 0.13%
55. http://www.nrmp.org/wp-content/uploads/2014/09/Charting-Outcomes-2014-Final.pdf 0.13%
56. https://pdpu.edu.ua/doc/vr/pavlovska/dis.pdf 0.13%
57. https://repositorio.uam.es/bitstream/handle/10486/678294/gonzalez_pizarro_patricio.pdf?sequenc… 0.13%
58. http://www.ijbmas.in/4.3.17/4307-4312%20V.SIVA%20SANKARAN.pdf 0.13%
59. http://www.akenergyauthority.org/Content/Programs/AEEE/Wind/WindResourceAssessment/Tan… 0.13%
60. http://repository.usu.ac.id/bitstream/handle/123456789/65581/Chapter%20III-V.pdf?sequence=3&… 0.13%
61. http://web.kpi.kharkov.ua/sopromat/wp-content/uploads/sites/29/2013/07/sopromat_Kyrkach_Roz… 0.11%
62. http://lib.chdu.edu.ua/pdf/metodser/188/1.pdf 0.11%
63. https://www.slideshare.net/Sanofi/201511-meet-sanofi-management 0.11%
64. http://www.onlineparentingcoach.com/2012 0.11%
65. https://en.wikipedia.org/wiki/Talk:Evolution/Archive_53 0.11%
66. https://business.baylor.edu/steve_green/green1.doc 0.11%
67. http://poippo.pl.ua/file/book/evaluation.pdf 0.11%
68. http://fs.onu.edu.ua/clients/client11/web11/metod/imem/usimchuk.pdf 0.11%
69. http://philosophy.karazin.ua/ua/nauka/rada/venevtseva_disser.pdf 0.11%
70. http://ief.org.ua/wp-content/uploads/2013/07/Dis_Guk.pdf 0.11%
71. http://nj.gov/health/rhc/documents/ed_report.pdf 0.11%
72. http://lib.iitta.gov.ua/166216/1/Tezy_IITZN_2016.4.PDF 0.11%
73. http://ir.nmu.org.ua/jspui/bitstream/123456789/3091/1/%D0%9D%D0%A2%D0%91451095.pdf 0.11%
74. http://eprints.covenantuniversity.edu.ng/687/1/PHD%20THESIS%20FOLA%20ADEGBIE%20CO… 0.11%
75. http://ej.kherson.ua/journal/economic_09-1/112.pdf 0.11%
76. http://historyscholars.weebly.com/uploads/1/4/7/8/1478974/geography_of_africa_pp_notes.pdf 0.11%
77. https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=910128JF.TXT 0.11%
78. https://collegereadiness.collegeboard.org/sites/default/files/psat_nmsqt_practice_test_1_answers… 0.11%
79. http://matesmatesymasmates.blogspot.com/2011/12/calculo-diferencial.html 0.11%
80. https://arxiv.org/pdf/1308.4675 0.11%

81. https://www.unicef.org/publications/files/Tracking_Progress_on_Child_and_Maternal_Nutrition_EN.. 0.11%
82. https://aceee.org/files/proceedings/2006/data/papers/SS06_Panel2_Paper25.pdf 0.11%
83. https://pt.slideshare.net/FelipeMago/metodologia-cientifica-25436835 0.11%
84. http://www.oecd.org/education/Global-competency-for-an-inclusive-world.pdf 0.11%
85. http://emoev.kpi.ua/wp-content/uploads/2015/09/12.pdf 0.11%
86. https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=9101H7D2.TXT 0.11%
87. https://residence-immobilien.ch/sites/default/files/ubs-real-estate-focus-2019-en_1.pdf 0.11%
88. http://www.onlineparentingcoach.com/2011/01 0.11%
89. https://ssq.github.io/2017/05/06/Coursera%20UIUC%20Data%20Mining%20Notebook 0.11%
90. http://e.lib.vlsu.ru/bitstream/123456789/5391/1/01575.pdf 0.11%
91. https://blogs.mathworks.com/pick/2007/08/20/matlab-basics-video 0.11%
92. https://infopedia.su/13x2c49.html 0.11%
93. http://www.readbag.com/sagepub-upm-data-2671-3asmnt02 0.11%
94. https://studopedia.su/8_49769_strunni-muzichni-instrumenti.html 0.11%
95. http://www.onlineparentingcoach.com/2008/02/house-rules-contract-cell-phones.html 0.11%
96. http://libertadpreciadotesoro.blogspot.com/2009_07_20_archive.html 0.11%
97. https://collegereadiness.collegeboard.org/pdf/psat-nmsqt-practice-test-1-answers.pdf 0.11%
98. https://en.wikipedia.org/wiki/Zygosity 0.11%
99. https://www.studyblue.com/notes/note/n/fawe-110-study-guide-2015-16-temple/deck/16711724 0.11%
100. http://www.onlineparentingcoach.com/2010/10 0.11%
101. http://rpitt.eng.ua.edu/Class/StormWaterManagement/M2%20problems%20Internet%20material… 0.11%
102. http://www.onlineparentingcoach.com/2009/05 0.11%
103. http://business.baylor.edu/steve_green/green1.doc 0.11%
104. http://www.onlineparentingcoach.com/2011/07 0.11%
105. http://dspace.ucuenca.edu.ec/bitstream/123456789/4108/1/MED56.pdf 0.11%
106. https://learn.ztu.edu.ua/mod/resource/view.php?id=4719 0.11%
107. http://www.onlineparentingcoach.com/2011/04/teaching-children-that-choices-have.html 0.11%
108. http://www.gobroomecounty.com/files/icp/reports/CommunitiesThatCareYouthSurvey.pdf 0.11%
109. http://www.wisp.umb.edu/Year_1_Evaluation.doc 0.11%
110. https://www.textbookmedia.com/Media/c03ca694-8969-4750-9bbc-365249600c33.pdf 0.11%
111. https://docplayer.nl/12337581-Masterproef-toepasbaarheid-van-effectieve-micro-organismen-in-e.. 0.11%
112. https://docplayer.es/6623440-C-analisis-del-sector-fortalezas-y-oportunidades.html 0.11%
113. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_quick_guide.htm 0.11%
114. https://docplayer.es/16301976-Introduccion-a-los-motores-de-combustion-interna-alternativos.html 0.11%
115. https://www.slideshare.net/yardog/internet-marketing-media-overview-ukraine-via-adproua 0.11%
116. http://stars.library.ucf.edu/cgi/viewcontent.cgi?article=3355&context=etd 0.11%
117. https://www.evs27.org/download.php?f=papers/EVS27-4790162.pdf 0.11%
118. https://vdocuments.site/refrigeration-562f968711538.html 0.11%
119. https://workstory.s3.amazonaws.com/assets/1105396/Mudrich_Prospectus_3.docx 0.11%
120. http://littleriverpondmill.com/en/dictionary.html 0.11%
121. https://www.slideshare.net/Kakie/apple-strategic-management-case-analysis 0.11%
122. http://www.mathematicallycorrect.com/pausd.htm 0.11%
123. http://darwin.eeb.uconn.edu/eeb348-notes/Lecture-Notes-in-Population-Genetics.pdf 0.11%
124. http://www.pondiuni.edu.in/sites/default/files/Research%20Methodology.pdf 0.11%
125. https://docplayer.nl/6549617-Colofon-willem-arntszkade-38-bis-laan-van-chartroise-70-3515-aj-ut.. 0.11%
126. https://docplayer.es/27860521-Universidad-central-del-ecuador-facultad-de-cultura-fisica.html 0.11%

127. https://students.ga.desire2learn.com/d2l/lor/viewer/viewFile.d2lfile/1798/12674/ecology01_print.h.. 0.11%
128. https://github.com/xkxx/algodb/blob/master/NEL/algolist.csv 0.11%
129. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/... 0.11%
130. https://www.slideserve.com/emily/inequality-poverty-and-welfare 0.11%
131. http://wels.open.ac.uk/research-project/caren/themes/care-costs 0.11%
132. http://www.readbag.com/courses-washington-css457-matlab-learning-matlab 0.11%
133. https://www.fac.es/files/descarga.php?t=prensa&doc=923&clave=5519abff6876a9da740809c88... 0.11%
134. https://www.huurderskoepel.com/uploads/images/wetenswaardigheden/2012%20huurtoeslag.pdf 0.11%
135. http://www.tesisenred.net/bitstream/handle/10803/6288/10CAPITOL6.pdf 0.11%
136. http://www.bcrp.gob.pe/docs/Publicaciones/Documentos-de-Trabajo/2015/documento-de-trabajo... 0.11%
137. http://www.onlineparentingcoach.com/2019/01/what-to-do-when-you-think-your-teen-is.html 0.11%
138. https://quizlet.com/120801604/3m071-cdc-svs-ures-flash-cards 0.11%
139. http://faculty.babson.edu/goldstein/Teaching/FIN3560Fall2013/Projects/Indian%20FX%20.pdf 0.11%
140. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/... 0.11%
141. http://newbiblereflections.blogspot.com/2015/07 0.11%
142. http://www.onlineparentingcoach.com/2010/10/high-school-students-and-academic.html 0.11%
143. https://www.bestbuy.ca/en-ca/product/d-link-d-link-powerline-av500-mini-adapter-starter-kit-dhp-3.. 0.11%
144. https://www.theseus.fi/bitstream/handle/10024/123598/Mariia_Bogatova_Thesis.pdf?sequence=1 0.11%
145. https://www.energiesparen.be/sites/default/files/atoms/files/Samenvattend_rapport_analyse_EP... 0.11%
146. https://core.ac.uk/download/pdf/148611421.pdf 0.11%
147. https://inthecornermumblinganddrooling.wordpress.com/2013/03/04/the-turn-to-disinflation-part-1.. 0.11%
148. https://annuario.isprambiente.it/sites/default/files/pdf/2009/en/Biodiversity.pdf 0.11%
149. https://oxfamintermon.s3.amazonaws.com/sites/default/files/documentos/files/economia-para-m... 0.11%
150. http://www.onlineparentingcoach.com/2007/10/too-much-tv.html 0.11%
151. http://www.iot.ntnu.no/users/fleten/students/tidligere_veiledning/OlimbOdegard_V10.pdf 0.11%
152. https://www.asdetur.com/files/informe-completo-2016.pdf 0.11%
153. https://core.ac.uk/download/pdf/70615092.pdf 0.11%
154. https://epic.awi.de/id/eprint/14504/1/Ove2006a.pdf 0.11%
155. https://www.nttdocomo.co.jp/english/binary/pdf/corporate/technology/rd/technical_journal/bn/vol1... 0.11%
156. http://troelslindgreen.dk/docs/31236_Intelligibility_of_speech.pdf 0.11%
157. https://zoek.officielebekendmakingen.nl/kst-33000-1.odt 0.11%
158. http://www.cshp.rutgers.edu/Downloads/7510.pdf 0.11%

Excluded as citation or reference Web sources: 3 sources found

1. http://www.readbag.com/nacua-documents-iirira 0.11%
2. https://www.law.cornell.edu/uscode/text/10/2302 0.11%
3. http://erepo.usiu.ac.ke/bitstream/handle/11732/2193/The%20Relationship%20between%20Perform.. 0.11%

Library sources: 44 sources found

[17.D3.0009] UTS- kasus sevel.docx 0.19%
14.I1.0073-Edwin Widjaja-15 FEB.docx 0.19%
ELANG-22 MARET.docx 0.19%
TIPMUTS_CAROLINE15.G1.0155.pdf.pdf 0.19%
Laurentius Yustika P_15.G1.0159_UTS TIPM.pdf.pdf 0.19%
15.I1.0015-Ian Ariel Handoyo-KP-21 MEI.docx 0.19%

Similarity
Similarity from a chosen source
abc Possible character replacement
Citation
References

14.I1.0029 Tan Rosana Evelyn-30 JUNI.docx	0.19%
14.I1.0020-Jessica Christianti Purwadi-2 JULI-S.docx	0.19%
14.I1.0029 Tan Rosana Evelyn-5 JULI S.docx	0.19%
14.D1.0184-MUHAMMAD IRSYAD ARKEN.docx	0.19%
13.07.0003 pieter.docx	0.19%
Jurnal (3).odt	0.17%
15.I1.0034 - Evan Fajar Alim-9 JUNI.docx	0.17%
15.I1.0034 - Evan Fajar Alim-6 JUNI.docx	0.17%
Jurnal (5).odt	0.17%
Laporan TA Teguh.odt	0.17%
jurnal (2).odt	0.17%
14.K1.0014 Teguh.odt	0.17%
Laporan TA Teguh (1).odt	0.17%
Laporan TA Teguh2.odt	0.17%
14.I1.0078-Adinda Khairunisa.docx	0.16%
14.I1.0078-Adinda Khairunisa Rev1.docx	0.16%
14.K1.0063_KARUNA_BODHI_SETIANTO.doc	0.14%
14.K1.0038 Roberto Rio Rakana Jaya.docx	0.14%
13020028_Perbaikan_19Jan.pdf	0.13%
Laporan Skripsi plagscan.docx.docx	0.13%
arauna imelda.docx	0.13%
Project_2017_13.02.0028.odt.odt	0.13%
13.02.0028-Stevanus Adi Ramli.doc	0.13%
ARAUNA-18 MEI.docx	0.13%
perbaikan_sebenarnya_13020028.pdf	0.13%
13.02.0067-Henricus Rendy.doc	0.13%
perbaikan_13020028.pdf	0.13%
13020028.pdf	0.13%
13020067.pdf	0.13%
PROJECT2017_13020067.pdf.pdf	0.13%
Skripsi_Plagscan Yosia.docx	0.11%
14.K1.0036-Edwin Listyo.docx	0.11%
14.K1.0036-Edwin Listyo.docx	0.11%
13.02.0013 Fernando Setiawan.doc	0.11%
13020062 Eric Victory (1).doc	0.11%
Nanda Isdian P. (13.12.0045) & Gerald Arsa A.A. (13.12.0061).doc	0.11%
14K20020.pdf	0.11%
14.K2.0020 Angela Indriana K.docx	0.11%

ABSTRACT

Scheduling has an important role in an organization, which regulates each member in carrying out an activity, especially for organizations that have a very large number of members and are divided into several divisions, scheduling is essential and cannot be taken lightly.

Hermon Community Church, a church which has hundreds of church servants that are divided in many divisions and scheduled in many church activities. This makes scheduling process very complicated, and of course there must be a good and optimal scheduling system, so that there will be no collision between activities, all church servants can get the same quota in their duties and church activities such as worship, community cell, and other events can run smoothly and effectively, so that the church can grow even more. This study aims to implement the genetic algorithm and steepest ascent hill climbing algorithm with the Java programming to carry out the process of scheduling for Hermon Community Church servants.

The result if this project is the genetic algorithm is more effective than the SAHC algorithm because it can find the best optimal schedules with the fitness value of 0 whereas the SAHC algorithm could only find schedules with the fitness value of 30 approximately.

***Keywords**: Scheduling, Genetic, Hill Climbing.*

PREFACE

This project mainly discusses about the implementation of the genetic and the steepest ascent hill climbing algorithm in the scheduling system. Chapter 1 discusses about the background, scope and purpose of the project. Chapter 2 discusses about the literature review or theoretical basis used in this project. Chapter 3 discusses about the steps taken in this project. Chapter 4 discusses about data analysis, the genetic and SAHC algorithm, and the design of this project. Chapter 5 discusses about the Pseudocode used in this project and the results from this project and its analysis. Chapter 6 is conclusion of this project.

CHAPTER 1 INTRODUCTION

1.1 Background

In a church, there are routine activities that are held, including public worship, youth-worship, Sunday school worship, etc. and each of these activities, there are church servants in charge of it. In the division of tasks for church servants, of course, a schedule that governs each church servant is needed, so that it can be known when and who is in charge of serving in an activity.

With the development of the Hermon Community Church today which has hundreds of congregations and no less than 200 people registered as church servants in various divisions, the process of scheduling church activities becomes increasingly complicated, of course a good schedule is needed to ensure there are no activities those who clash and share allotments serving each church servant must be balanced.

Therefore, this project discusses and provides solutions for this church scheduling system using computer technology, specifically using Genetic algorithms and Hill Climbing algorithms to help optimize the scheduling system in this church.

1.2 Scope & Problem Formulation

The scopes of this project are

1. This project only use Genetic and Hill Climbing algorithms to provide solution for the scheduling system of Hermon Community Church.
2. This project only use the data that obtained through survey in Hermon Community Church.

And the problem formulations of this project are

1. Can the Genetic algorithm be used to optimize the scheduling system for Hermon Community Church church servants?
2. Can the Hill Climbing algorithm be used to optimize the scheduling system for Hermon Community Church church servants?
3. How is the comparison of the effectiveness of the two algorithms in optimizing the scheduling system for Hermon Community Church church servants?

1.3 Objective

1. To help develop the scheduling system for the Hermon Community Church.
2. To find out the role and workings of Genetic algorithms in helping the Hermon Community Church church scheduling system.
3. To find out the role and workings of the Hill Climbing algorithm in helping the Hermon Community Church church scheduling system.
4. To compare which is more effective between the Genetic algorithm and the Hill Climbing algorithm in solving the problem of the Hermon Community Church scheduling system.

CHAPTER 2 LITERATURE STUDY

Denny Hermawanto (2007) write a module about the genetic algorithm and its applications and explain that the genetic algorithm is a computational algorithms inspired by Charles Darwin's evolutionary theory which is later adopted into computational algorithm to find solutions to problems in a more "natural" way. " Here the example of the application of genetic algorithm to solve the problem of equations a + 2b + 3c + 4d = 30, where the values of a, b, c, d are used as genes that form a chromosome whose value will be searched as a solution to the problem. After applying the genetic algorithm 50 times to the problem of this equation, the most perfect chromosome is found with a value of a = 7; b = 5; c = 3; d = 1.

Wiga Ayu P (2013) conducts a research on lecture scheduling for Information System department using genetic algorithm in website-based Sepuluh Nopember Institute of Technology University and explain that the application could produce schedules in accordance with the standard constraints (primary limitations) that must be met even though the software limits are not all fulfilled yet.

Nia Kurnia Mawaddah (2006) also conducts a research on optimization of exam scheduling by making a genetic model and explain that genetic algorithm can be used as an alternative solution to solve exam scheduling problems. The exam schedule is obtained from chromosomes that have the best fitness value with an optimal crossover probability level of 60%.

Shoffan Saifullah (2016) conducts a research on the development of the University of Yogyakarta's Faculty of Science and Technology lecture scheduling system with Delphi 7, MySQL database, and the Steepest Ascent Hill Climbing algorithm and explain that this development could help accelerate the lecture scheduling process and minimize the occurrence of clashes between schedules.

Putranto (2012) also conducts a study on the design and implementation of lecture scheduling using the website-based Steepest Ascent Hill Climbing algorithm and MySQL database at the SWCU Psychology Faculty and explain that the application can produce output in the form of an optimal lecture schedule, that is on the schedule produced, there are no collision between lecturers teaching hours, no collision in the lecture room, and each lecturer got a teaching quota that did not exceed the maximum teaching limit. The Steepest Ascent Hill Climbing algorithm has the advantage that all possible schedule solutions will be generated and will be examined one by one, so that the process is fast because the best solution will be obtained and the results are close to the expected optimization.

Aida Putri D (2017) also conducts a research on lecture scheduling using the hill climbing algorithm at the University of Technology in Yogyakarta and explain that the application of hill climbing algorithms on the system could produce optimal output in the form of a schedule of lectures and no lecturing schedule clashes were found between the lecturers. The application of the hill climbing algorithm has the advantage that all schedule solutions will be examined one by one, so that the process is fast because the best solution will be obtained and close to the expected optimization results.

Nuzulla (2012) conducts a study on the implementation of the Steepest Ascent Hill Climbing algorithm with Minimax optimization on the Android-based Tic Tac Toe game and conclude that the application of the steepest ascent hill climbing algorithm in the Java programming language can search for pathways that can be done on Tic Tac Toe and then the closest path to victory can be determined.

CHAPTER 3 RESEARCH METHODOLOGY

The steps taken for this project are

1. Data Collection

   Data collection was carried out by conducting a survey to Hermon Community Church to obtain data in the form of a list of church activities and a list of all divisions of church servants and their members.

2. Database Creation

The database is created with a .csv file that lists church activities and lists all church servants and is loaded into memory using a scheduling program with Arraylist data structures in Java.

3. Determining Constraints

   The next step is to determine the constraints that will be used for determining whether a schedule is an optimal or not.

4. Applying Genetic and SAHC Algorithm

   After the database is formed, genetic algorithms and SAHC are then applied to create an optimal schedule, which is a schedule that meets the specified conditions / contraints then save the results into memory.

5. Creating and Designing Output

   The optimized schedule that created by genetic and SAHC algorithm in memory are then written in the output in the form of a .txt file and also history of creating schedules of the scheduling program will be noted in a other .txt file and these two files will be used for analyzing in the next step.

6. Analyzing Output

   After an optimal schedule and history of schedule creation in the .txt file are created, then the two algorithms will be analyzed to compare which algorithm is more effective.

7. Creating Analysis Report

   The final step is to make a detailed report of analysis of comparison of the two algorithms and determine which algorithm is more effective.

CHAPTER 4 ANALYSIS AND DESIGN

4.1 Analysis

This project uses data obtained from the results of the Hermon Community Church survey, the data are as follows

Church Servants

Number of Divisions : 17 divisions consists of

- Worship Leader : 7 members
- Singer : 20 members
- Gitar : 5 members
- Bass : 5 members
- Keyboard : 6 members
- Drum : 4 members
- Usher : 25 members
- Sound : 1 member

- LCD : 6 members
- Tamborin : 11 members
- Banner : 11 members
- Persembahan : 67 members
- Perjamuan Kudus : 22 members

Church Activities in general

- Morning Public Worship 1 (06.30 – 08.30)
- Morning Public Worship 2 (09.00 – 11.00)
- Evening Public Worship (17.00 – 19.00)

From the data above, there will be many variants of the schedule that can be made. Suppose that a single worship requires 1 WL, 3 Singers, 2 Guitars, 1 Bass, 2 Keyboards, 1 Drum, 3 Usher, 4 Banners, 4 Tambourines, 1 Sound, 1 LCD,
8 Collections, if the permutation is used then

- 1 WL : 7P1 = 7! / (7-1)! = 7! / 6! = 7 variants
- 3 Singers : 20P3 = 20! / (20-3)! = 20! / 17! = 6840 variants
- 2 Guitars : 5P2 = 5! / (5-2)! = 5! / 3! = 20 variants
- 1 Bass : 5P1 = 5! / (5-1)! = 5! / 4! = 5 variants
- 2 Keyboards : 6P2 = 6! / (6-2)! = 6! / 4! = 30 variants
- 1 Drum : 4P1 = 4! / (4-1) = 4! / 3! = 4 variants
- 3 Ushers : 25P3 = 25! / (25-3)! = 25! / 22! = 13800 variants
- 4 Banners : 11P4 = 11! / (11-4)! = 11! / 7! = 7920 variants
- 4 Tambourines : 11P4 = 11! / (11-4)! = 11! / 7! = 7920 variants
- 1 Sound : 1P1 = 1! / (1-1)! = 1! / 0! = 1 variants
- 1 LCD : 6P1 = 6! / (6-1)! = 6! / 5! = 6 variants
- 8 Offering Collectors : 67P8 = 67! / (67-8)! = 67! / 59! = 2.629×10^{14} variants

So that the total possible variations from a single worship is 7.847×10^{35}. If there are three times of worship in a single day, the result will be 2.354×10^{36}. If the schedule is made per month (4 times a week) then the result will be 9.417×10^{36}.

Based on the results of the above calculations, there are so many possible schedule variants so an optimization algorithm is needed, that is the genetic algorithm and the steepest ascent hill climbing algorithm to help find the optimal schedule.

4.2 Design

In a schedule, there are several constraints that must be fulfilled which are used as parameters to evaluate a schedule, such as

- In an activity, there should be no repeated member's name in each division
- In an activity, there should be no member that registered in two divisions at once (serving in two divisions at the same time)
- A member cannot serve in two activities at the same time
- Each member only gets a certain number of serving quota with the formula maximum limit = the total number of slots of a division available in a schedule / total number of members that are registered in the division.

The Steepest Ascent Hill Climbing algorithm is a branch of the Hill Climbing algorithm, this algorithm compares all possible successors, so that the next state is the best or closest to the goal.

The Steepest Ascent Hill Climbing algorithm has the following steps:

1. Evaluation of initial state. If the initial state is the same as the goal state, then return to the initial state and stop the process. If not, proceed to 2nd step.
2. Start with current state = initial state.
3. Get all successors that can be used as the next state in the current state.
4. Evaluate all successors with the evaluation function and record the value of the evaluation results. If one of the successors has a better fitness value than the current state, then make the successor with the best fitness value as the new current state.

   Perform this operation continuously until it reaches the current state = goal state or there is no more changes in the current state.

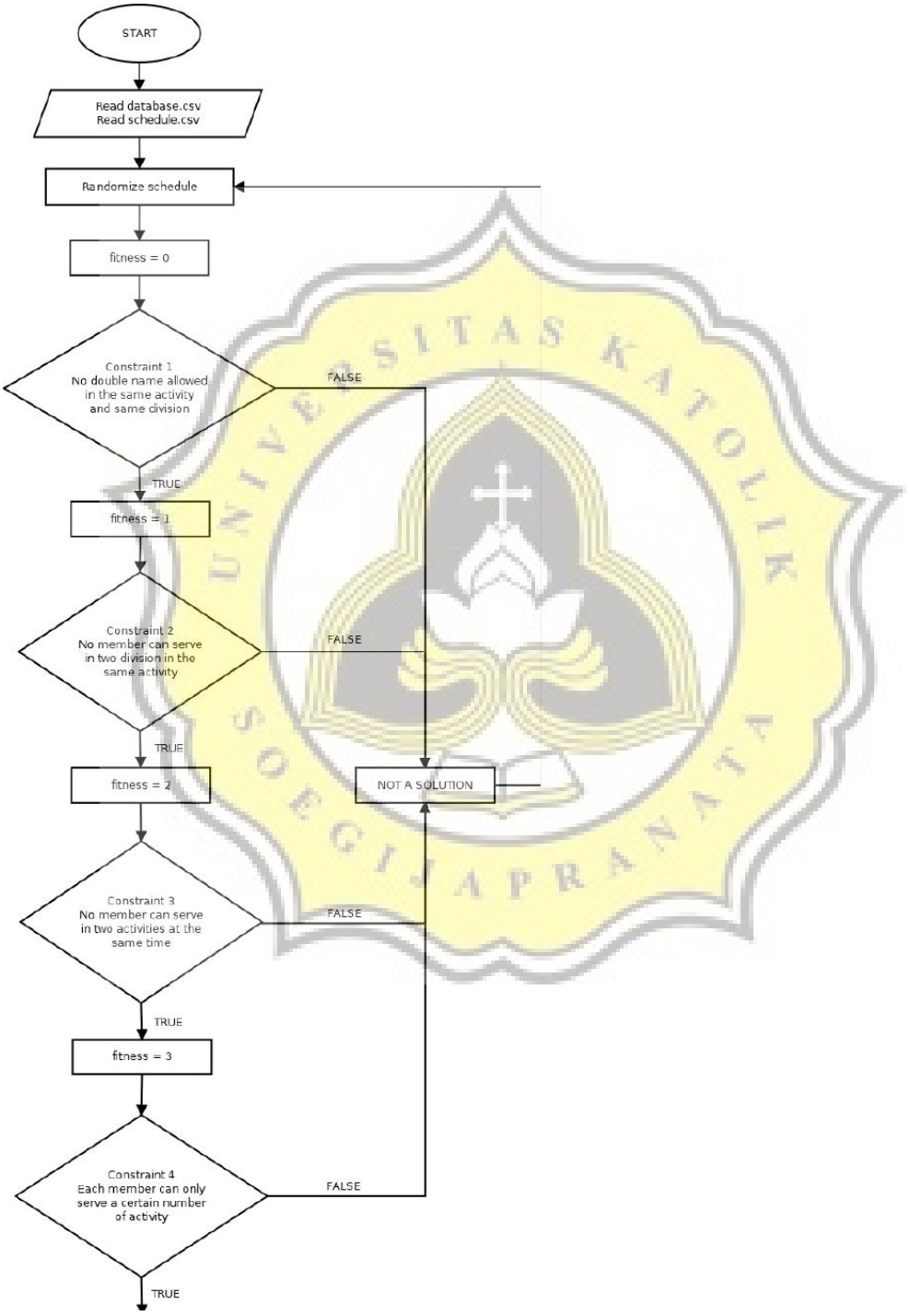


Illustration 4.1: Flowchart of Steepest Ascent Hill Climbing Algorithm

The Genetic Algorithm developed by Goldberg is a computational algorithm inspired by the evolutionary theory of Charles Darwin which states that the survival of a creature is influenced by the rules of "the strong are the victors" and can also be maintained through reproductive processes, crossover, and mutations. This concept can be developed into a computational algorithm to find solutions to a problem in a "natural" way.

A solution that is generated in a Genetic algorithm is called a chromosome and the collection of chromosomes is called population. A chromosome is composed by components that containing data or information in the form of numerical, binary, symbol, character, word, etc. (depending on the problem) which is called a gene. The chromosomes will later evolve continuously which will be called as generation. In each generation the level or value of perfection or success will be evaluated in solving the problem referred to as fitness value.

Genetic Algorithm consists of several steps starting from generating a population that contains random chromosomes and evaluating the fitness value of each chromosome. The next step is to make a selection, which is to choose the chromosome that has the best fitness value to produce the next generation by crossing or commonly called a crossover, then it will produce a new chromosomes that are called as the offsprings. Then the new chromosomes / offsprings will have the possibility to experience mutations, that is changes in the composition of genes / constituents of living things due to natural factors in this algorithm represented by changing one or more values of data / genes in the chromosome with random values. So with this, a new generation will be formed with new fitness values that are different from the previous generation. After several generations have been produced, it will be found the chromosomes that has the maximum fitness value is the best solution to the problem.

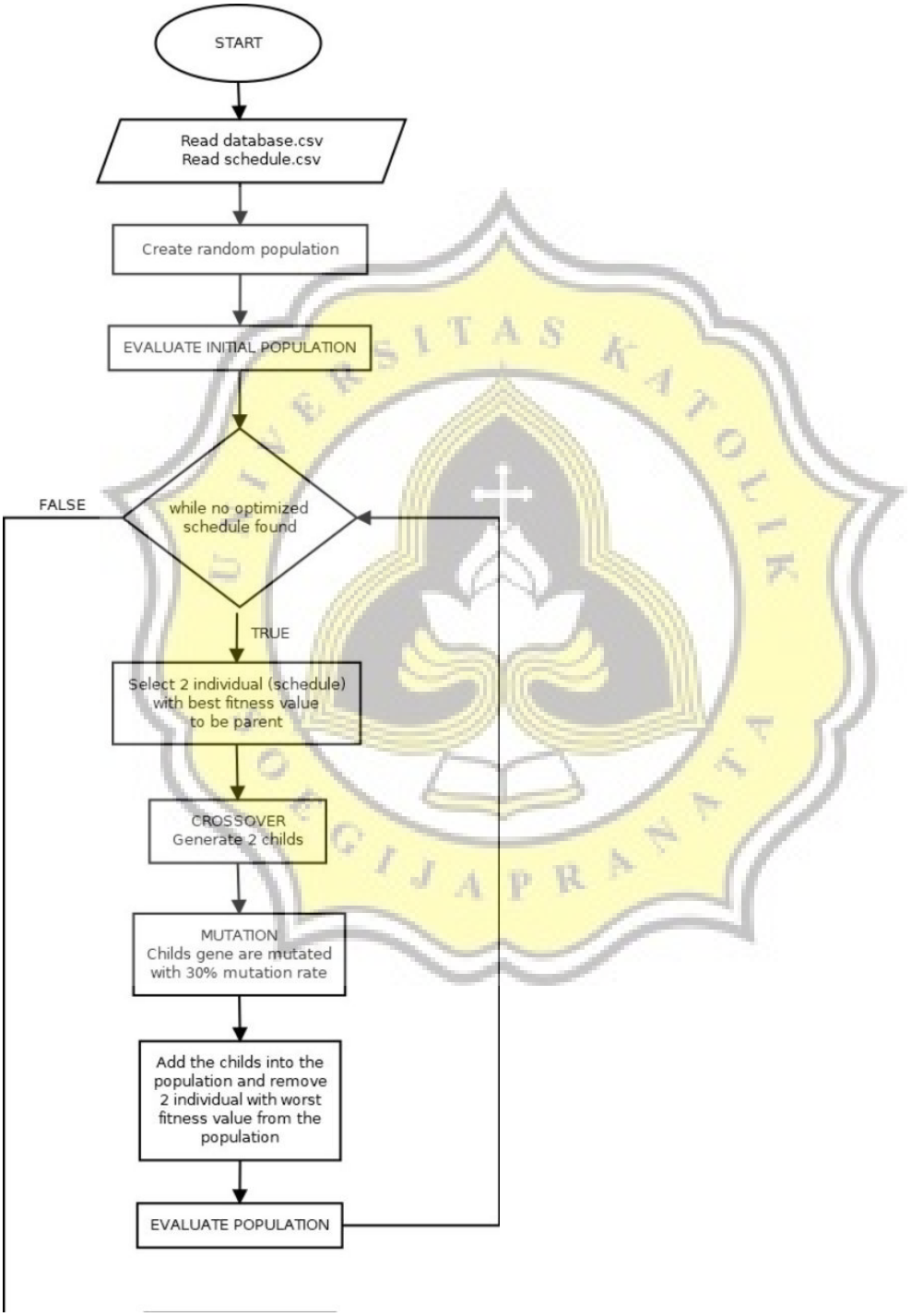


Illustration 4.2: Flowchart of Genetic Algorithm

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.

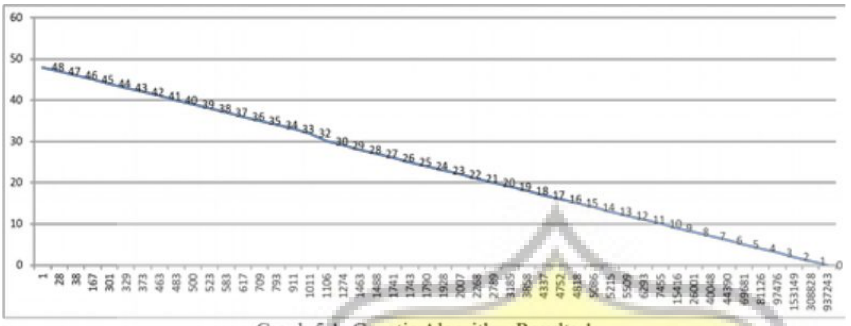
5.2 Testing

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

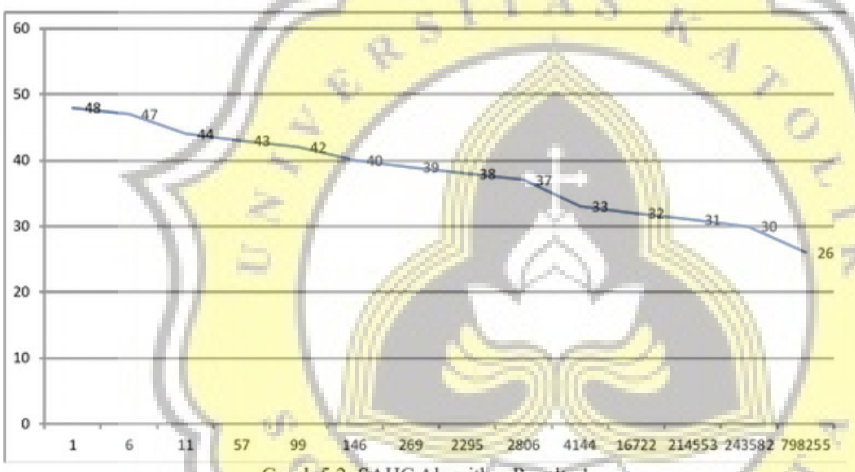
- Morning Public Worship 1 (December 2, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 2, 2018, 09.00 – 11.00) • Evening Public Worship (December 2, 2018, 17.00 – 19.00)
- Morning Public Worship 1 (December 9, 2018, 06.30 – 08.30)
- Morning Public Worship 2 (December 9, 2018, 09.00 – 11.00)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

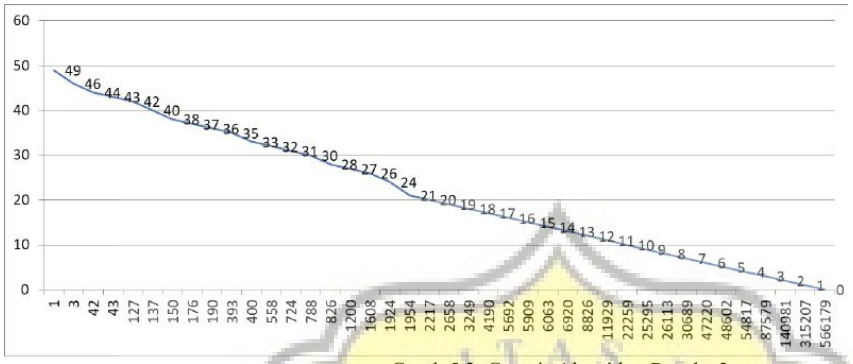


Graph 5.1: Genetic Algorithm Results 1

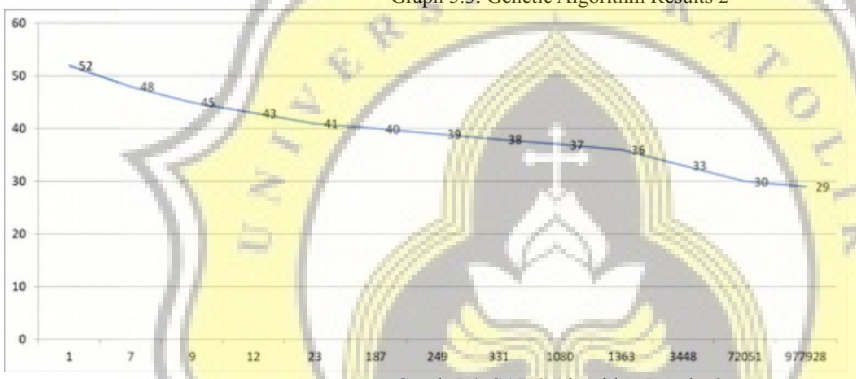


Graph 5.2: SAHC Algorithm Results 1

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.

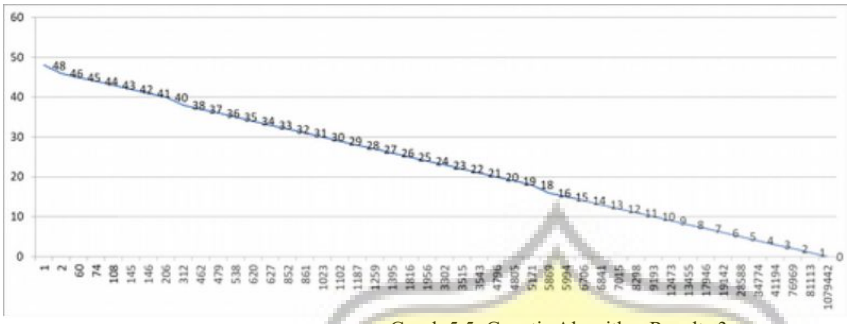


Graph 5.3: Genetic Algorithm Results 2

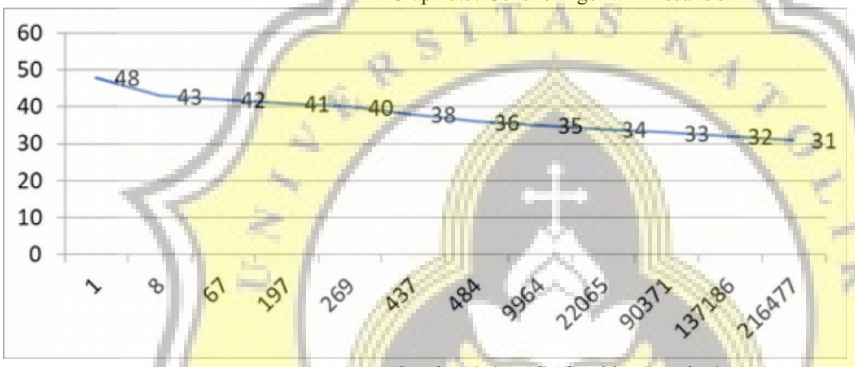


Graph 5.4: SAHC Algorithm Results 2

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.

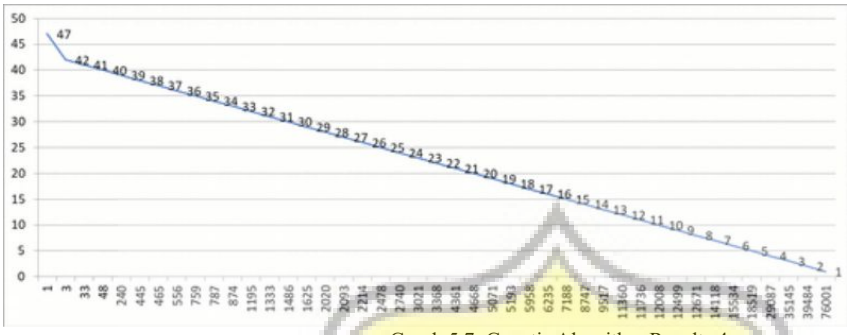


Graph 5.5: Genetic Algorithm Results 3

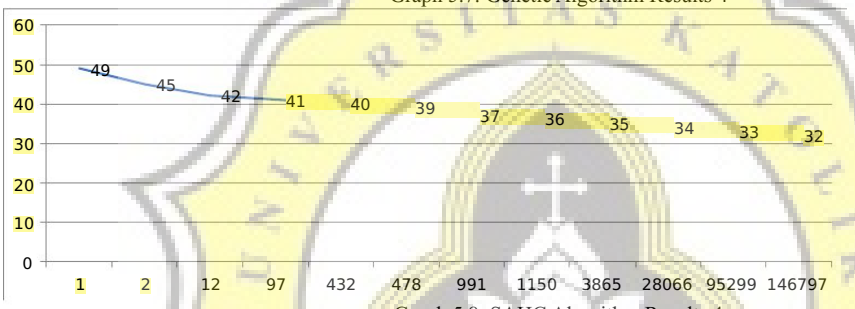


Graph 5.6: SAHC Algorithm Results 3

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.

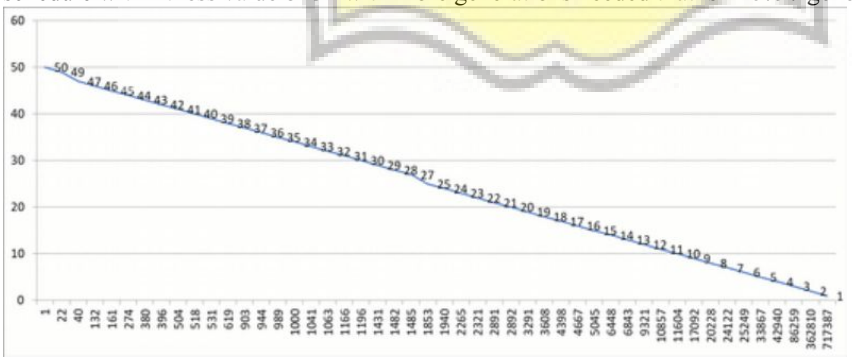


Graph 5.7: Genetic Algorithm Results 4

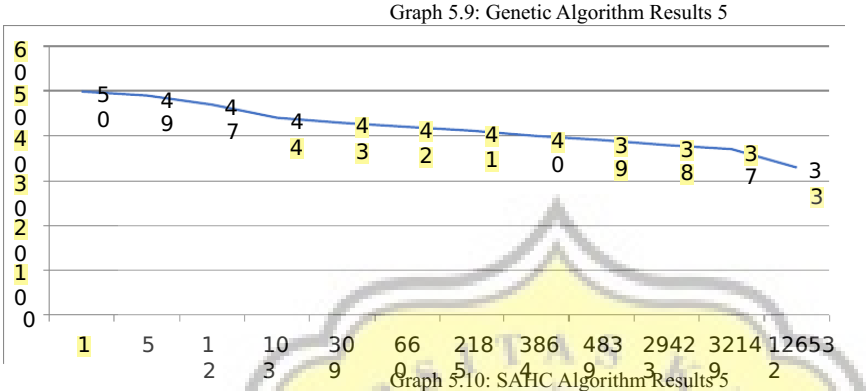


Graph 5.8: SAHC Algorithm Results 4

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.

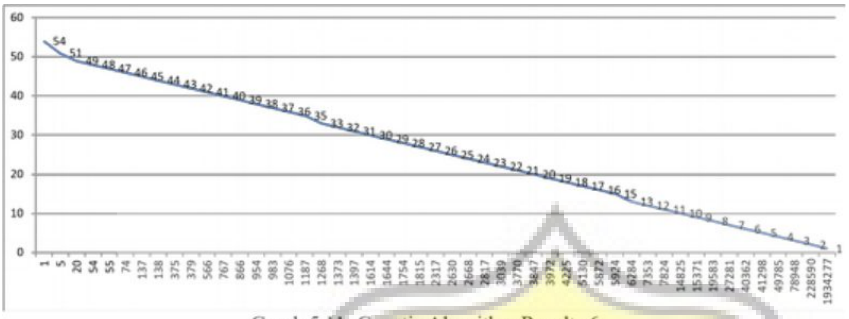


Graph 5.9: Genetic Algorithm Results 5

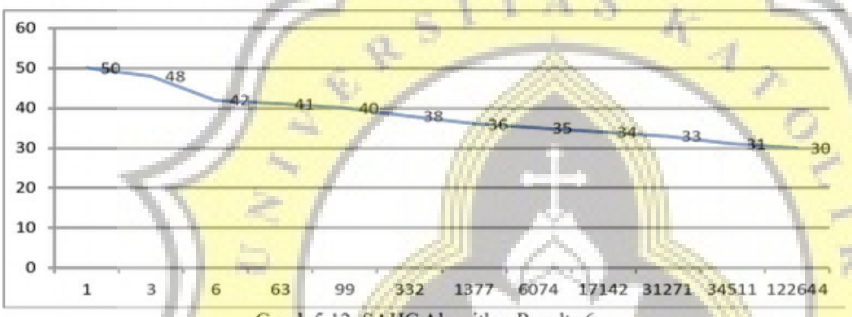


Graph 5.10: SAHC Algorithm Results 5

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.

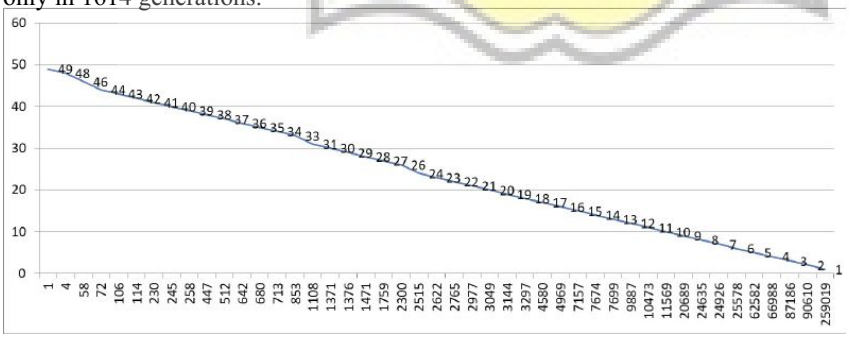


Graph 5.11: Genetic Algorithm Results 6

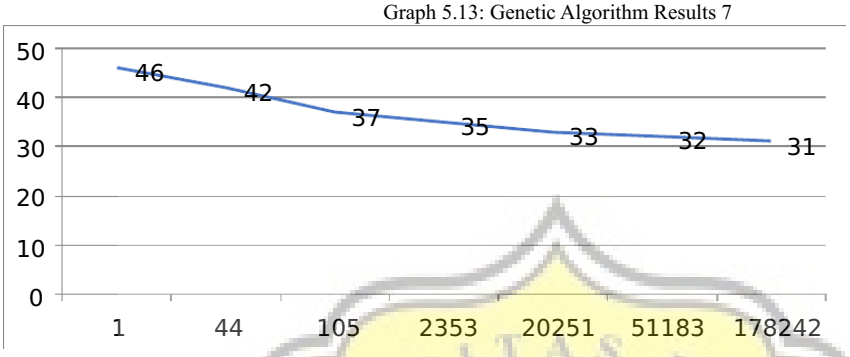


Graph 5.12: SAHC Algorithm Results 6

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.

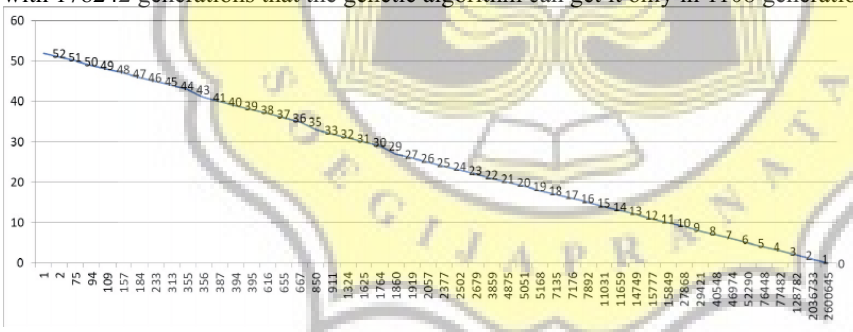


Graph 5.13: Genetic Algorithm Results 7

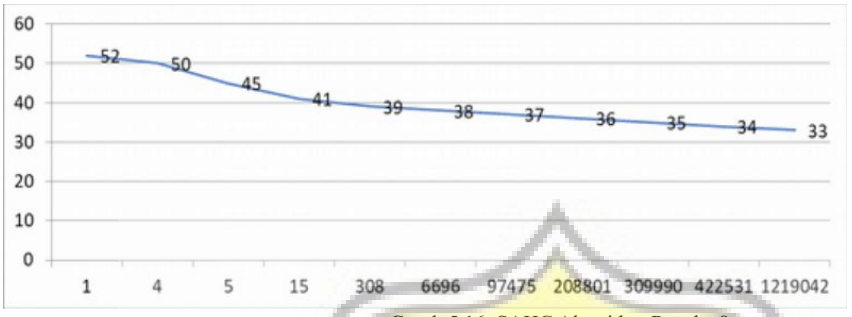


Graph 5.14: SAHC Algorithm Results 7

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.

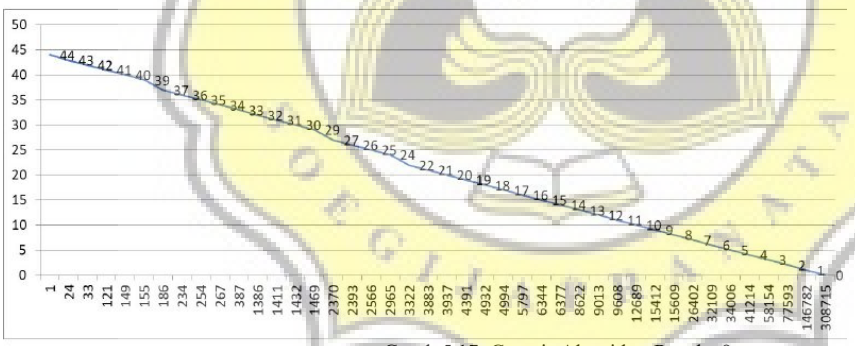


Graph 5.15: Genetic Algorithm Results 8

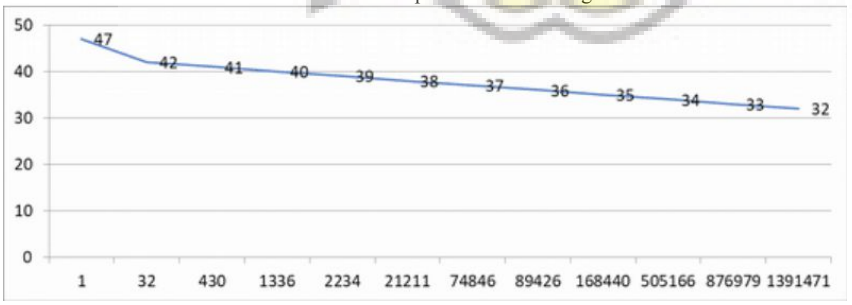


Graph 5.16: SAHC Algorithm Results 8

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.

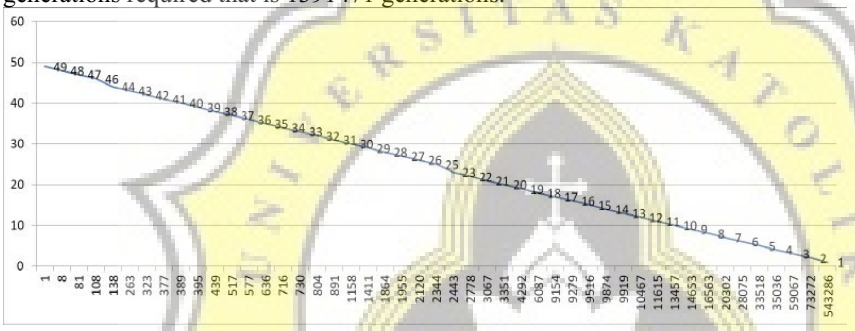


Graph 5.17: Genetic Algorithm Results 9

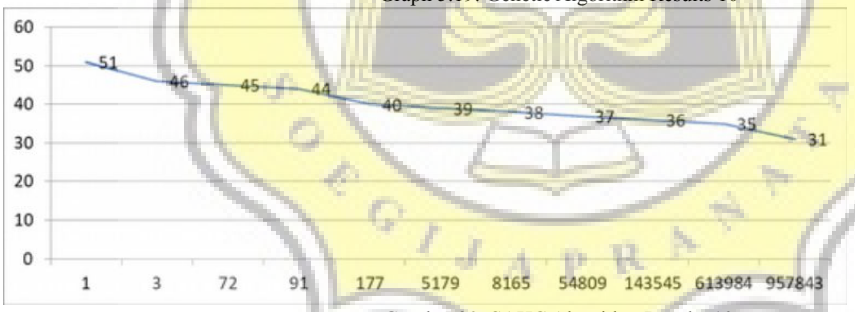


Graph 5.18: SAHC Algorithm Results 9

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.

CHAPTER 6 CONCLUSION

Based on this project results, in conclusion, the genetic algorithm has been consistently outperforms the SAHC algorithm and the genetic algorithm can be used to optimize the scheduling system for Hermon Community Church, this can be proven by the running results and analysis, it can find the optimal or almost optimal schedule (with the fitness value of 0 or 1) whereas the SAHC algorithm can only find schedule with the fitness value of 30 approximately and it cannot be used to optimize the scheduling system for Hermon Community Church.

In terms of effectiveness, although both algorithms have same complexity, the genetic algorithm has better processing speed (240 generations / sec) for finding the optimal schedule than the SAHC algorithm (207 generations / sec).

More research is needed about this project with different variations, for example by using different data structures or even with different programming languages to get more information or different data variations about this study.