Intelligent Systems - Problem Solving
Minimum Makespan using Genetic Algorithm

Gabriele Sartor
University of Turin

Morgan Gautherot
University of Lille

December 24, 2016

1 Minimum Makespan Problem

In the Minimum Makespan problem we have to assign jobs to many machine working in simultaneously in order to execute them in the smallest possible amount of time. Given a set of n jobs, a set of m machines and p_{ij} the time required by the machine i to execute the job j, we want to find a complete assignment which minimizes the makespan. The **makespan** is the time in which the last job finishes its execution. For this project we had to solve this problem assigning 512 jobs to 16 machines.

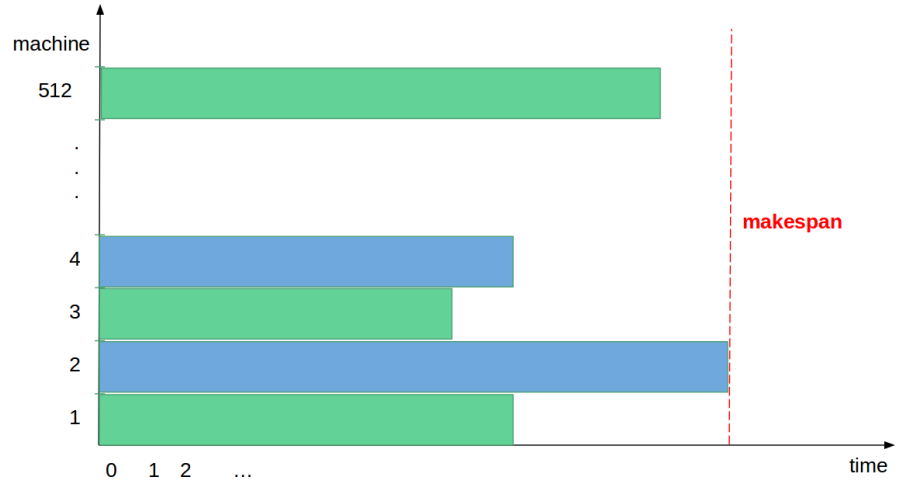

Figure 1: Each machine shows the time needed to execute all its jobs. The time in which all the machines finished their work is the makespan.

2 Implementation

In this section we describe the structure of our project and the main function, giving some motivation about our choices.

2.1 Files

In order to solve this problem we implemented different Java classes:

- **Algorithm.java**, describes different algorithms such as single point crossover, mutation and binary tournament;
- **Chromosome.java**, contains the structure of the chromosome;
- **Exe.java**, creates an instance of the problem and the algorithm, then executes some iterations of the algorithm to solve the problem;

- **Individual.java**, represent an individual having a chromosome and fitness;
- **Population.java**, contains all the individuals and keeps track of the best and worst ones;
- **Problem.java**, is an interface which defines the structure a problem;
- **ProblemMinMakeSpan.java**, calculates the fitness of an individual for the problem we are taking into account;
- **ReadTasks.java**, class useful to read files containing information about execution time of jobs in each machines;
- **testValues.java**, used to run different instances of the problem saving results on csv files.

The problem is solved by executing the main of the class Exe.java specifying of the problem on the terminal the name of the file containing the data, for example `java Exe u_s_hihi_512_16.txt`.

The repository contains also the `README.md` file containing further information about how to run this code.

2.2 Chromosome

In order to solve this problem we use a Genetic Algorithm in which we consider a chromosome as sequence of assignments between machines and jobs. In our representation, the chromosome is the array of integer `alleles` in which `alleles[i]` is the number of the machine executing the job i. Using this implementation we avoid cases in which we have to fix the chromosome, because we will always generate valid assignments.

2.3 Main functions

In order to implement the **single point crossover**, the program takes two individuals and a random index k of the array. A new individual will be created with a chromosome made up of the first k values of the first individual taken into account, and the rest of values from the second one.

The fitness of an individual is calculated taking into account all the time required by machines to finish their work. In `ProblemMinMakeSpan.java` we can see how it is calculated. We have an array `f` in which we keep track of the total amount of time required by each machine to finish its work. The makespan is the highest value contained in `f`. Given that our program will look for the highest fitness, but we have to minimize the makespan, we will save as fitness the makespan making it negative. In this way, we will have only negative fitness and we will try to find the closest one to the value 0. That one will be our best solution at end of the execution of our algorithm.

The **mutation** function consists in calculating a random number between 0 and 1 for each number of the array and, if the random value is smaller than the probability of mutation, we will change the value in that position with a random integer(always between 0 and 15).

Geometrically, when we apply the crossover we find new solutions close to the point used in this functions. Applying the mutation, we could move in a completely different place of the solution space.

3 Description of the algorithm

First of all, our program will initialize the instances of the objects we need (problem, population, algorithm, etc...). Consequently, we will have a population containing chromosome created using random numbers between 0 and 15(representing our machines). In this way, it is not possible to create conflicts inside the chromosome, therefore we cannot have chromosomes to fix.

After that we carry out one step of the algorithm on our problem until we find the best known solution or reaching to maximum of the steps defined by the program(`MAX_ISTEPS`).

Each step, the algorithm will choose two individuals using the binary tournament algorithm, then perform a single point crossover on them creating a new individual, then calculates its fitness. If the fitness of the new individual is better than the worst fitness of the population, the new entity will replace the worst one.

At the end of the algorithm, the best solution found is displayed on the terminal(chromosome and fitness).

4 Statistics Analysis for u_s_hihi_542_16

4.1 introduction

Now we have finish the change of the code we have to make a statistics analysis to find the best parameters for two data sets. We will change three parameters, the probability of mutation, the probability of crossover and the size of the population and we will see if when we change this parameter we will see a change of the result.

4.2 visualization

We will start to see the change of the probability of crossover on our result. I change the probability from 0.7 to 0.9 by 0.05.

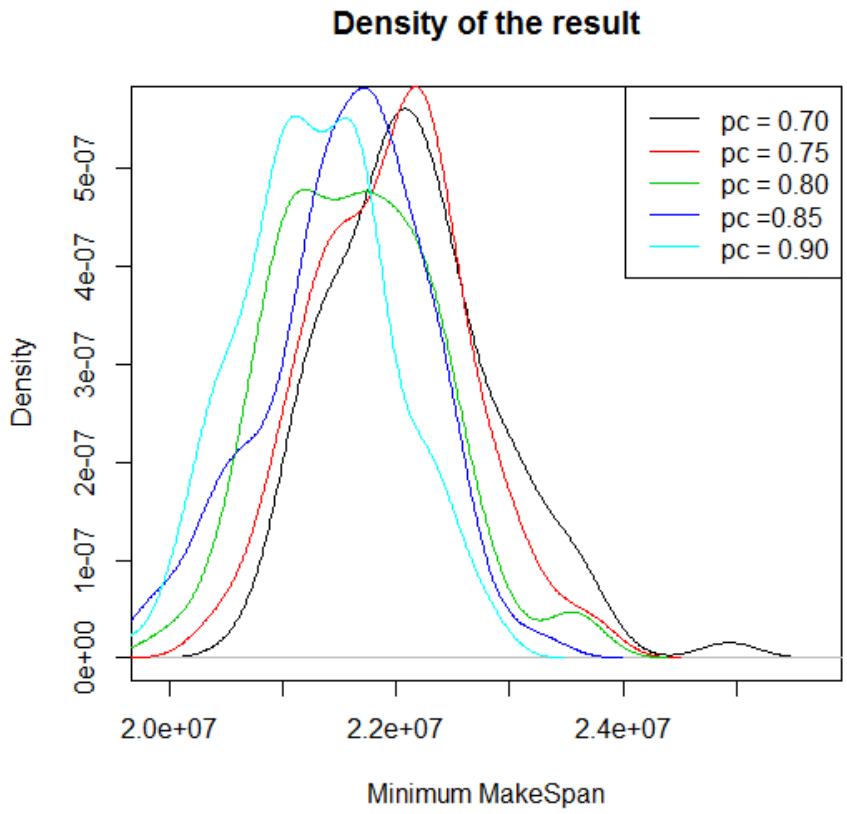

Figure 2: Density for different value of the crossover probability

We can see that the result seems better when the probability of the crossover increase.

We will now see the change of the probability of mutation on our result. I change the probability from 0.002 to 0.18.

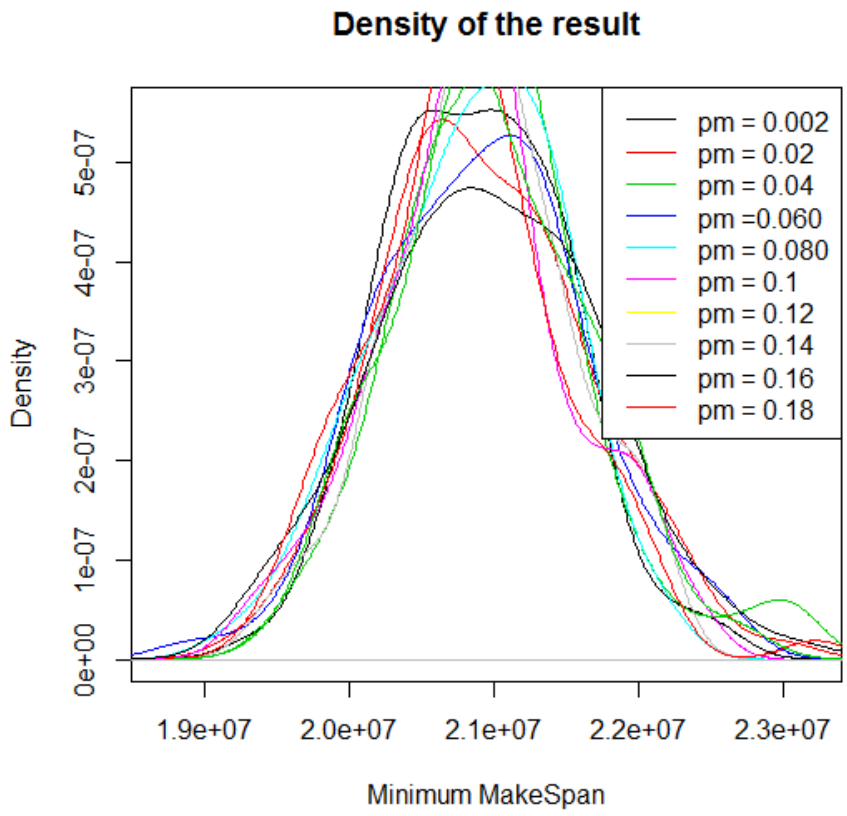

Figure 3: Density for different value of the mutation probability

We can see that the result seems the same when the probability of the mutation increase.

We will now see the change of the size of the population on our result. I change the probability from 256 to 1024.

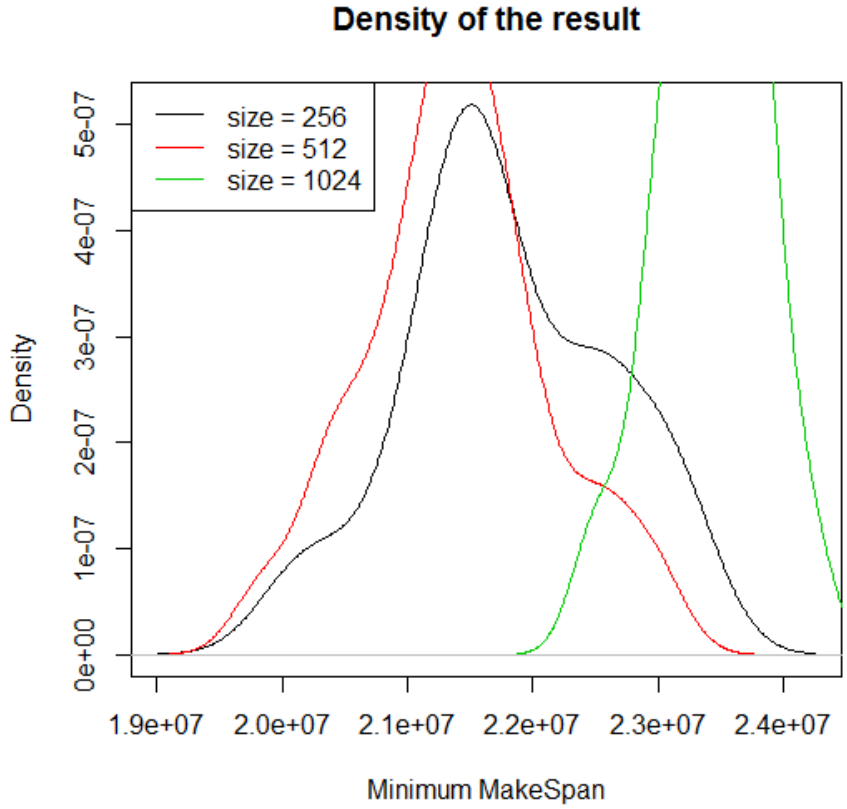

Figure 4: Density for different value of the size of the population

We can see that the result seems the same when the probability of the mutation increase.

4.3 Which value use ?

We create a class call testValues, which create a csv.file with 50 obvservations of every change of parameters.

Table 1: Change of the parameter

Parameters	Start value	End value	Step
Mutation probability	0.7	0.9	0.1
Crossover probability	2/512	10/512	4/512
Population size	256	512	x^2

To continue our study we will create label to our distribution we will name our distribution like that sizepop_probabilityCrossOver_ProbabilityMutation. For example if we have distribution with 256 population 0.8 in p-crossover and 2/512 in mutation it will be called 256_0.8/2.

4.4 Normality of our data

I will use the shapiro test on R to see if our data follow a normal distribution.

Table 2: Change of the parameter

Label	P-value	Normality
256_0.8_2	0.79	True
256_0.8_6	0.11	True
256_0.8_10	0.57	True
256_0.9_2	0.2	True
256_0.9_6	0.74	True
256_0.9_10	0.84	True
512_0.8_2	0.06	True
512_0.8_6	0.84	True
512_0.8_10	0.42	True
512_0.9_2	0.19	True
512_0.9_6	0.78	True
512_0.9_10	0.84	True

All our distribution follow a normal distribution because all their p-value avec above 0.05.

4.5 Homoscedasticity of our data

We will use the fisher test on R to check the homogeneity of variance.

256_0.8_10 have a p-value below 0.05 so we will continue without this set of data.For all the other they have p-value above 0.05.

4.6 ANOVA

We check previously the normal distribution and the homoscedasticity of our data.Therefore we can apply anova on that set. This test give us a p-value equal to $9.9 * 10^{-}9$ it is below 0.05 so we can say that their distribution were different.

4.7 TukeyHSD

We use TulkeyHSD on R to see the difference with our distributions. We can see that when the probability of crossover is equal to 0.9 the distribution give better result than this one equal 0.8. And in the most of the case when we have 512 population we have better result than with 256. But for the mutation probability there is not significant difference, we plot the different result.

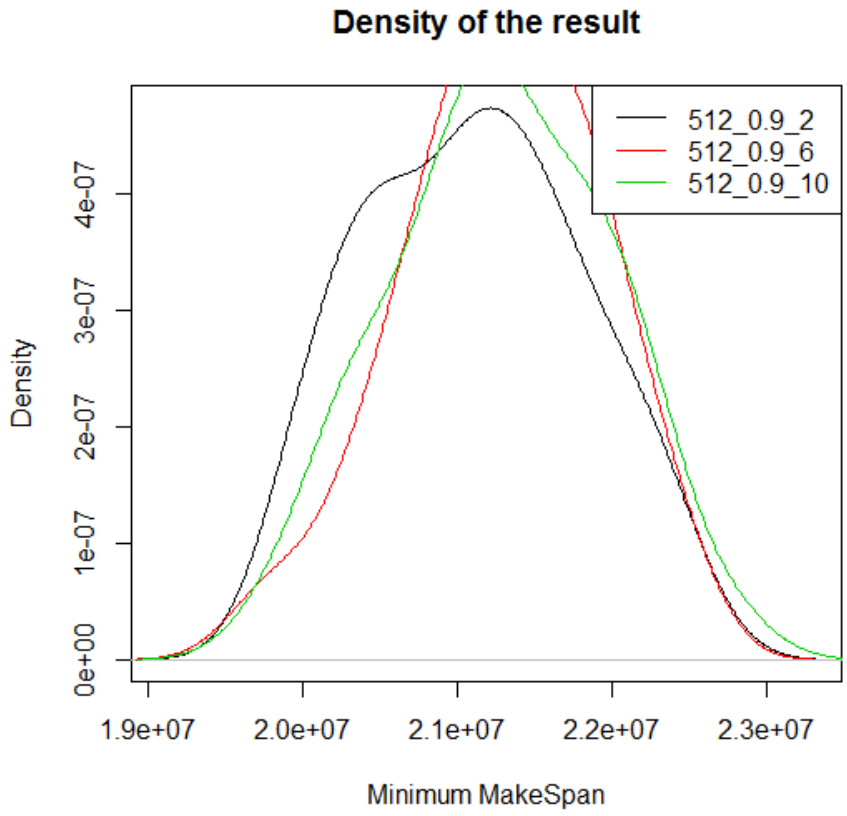

Figure 5: Density for different value of the mutation probability

This plot show that with the value 2/512 the distribution looks like better. Like the distribution are significantly different by the anova we can take 2/512 as parameter for the probability of mutation.

4.8 Conclusion

For the file u_s_hihi_542_16 best parameters are for the probability of crossover 0.9, for the probability of mutation 2/512 and for the size it is 512.

5 Statistics Analysis for u_s_lohi_542_16

5.1 introduction

We will study the same way that for u_s_hihi_542_16

5.2 Normality of our data

I will use the shapiro test on R to see if our data follow a normal distribution.

Table 3: Change of the parameter

Label	P-value	Normality
256_0.8_2	0.57	True
256_0.8_6	0.87	True
256_0.8_10	0.61	True
256_0.9_2	0.35	True
256_0.9_6	0.48	True
256_0.9_10	0.10	True
512_0.8_2	0.7	True
512_0.8_6	0.75	True
512_0.8_10	0.42	True
512_0.9_2	0.20	True
512_0.9_6	0.78	True
512_0.9_10	0.84	True

All our distribution follow a normal distribution because all their p-value avec above 0.05.

5.3 Homoscedasticity of our data

We will use the fisher test on R to check the homogeneity of variance. And for all the p-value is above 0.05.

5.4 ANOVA

We check previously the normal distribution and the homoscedasticity of our data.Therefore we can apply anova on that set. This test give us a p-value equal to $2 * 10^{-}16$ it is below 0.05 so we can say that their distribution were different.

5.5 TukeyHSD

We use TulkeyHSD on R to see the difference with our distributions. We can see that when the probability of crossover is equal to 0.9 the distribution give better result than this one equal 0.8. And in the most of the case when we have 512 population we have better result than with 256. But for the mutation probability there is not significant difference, we plot the different result.

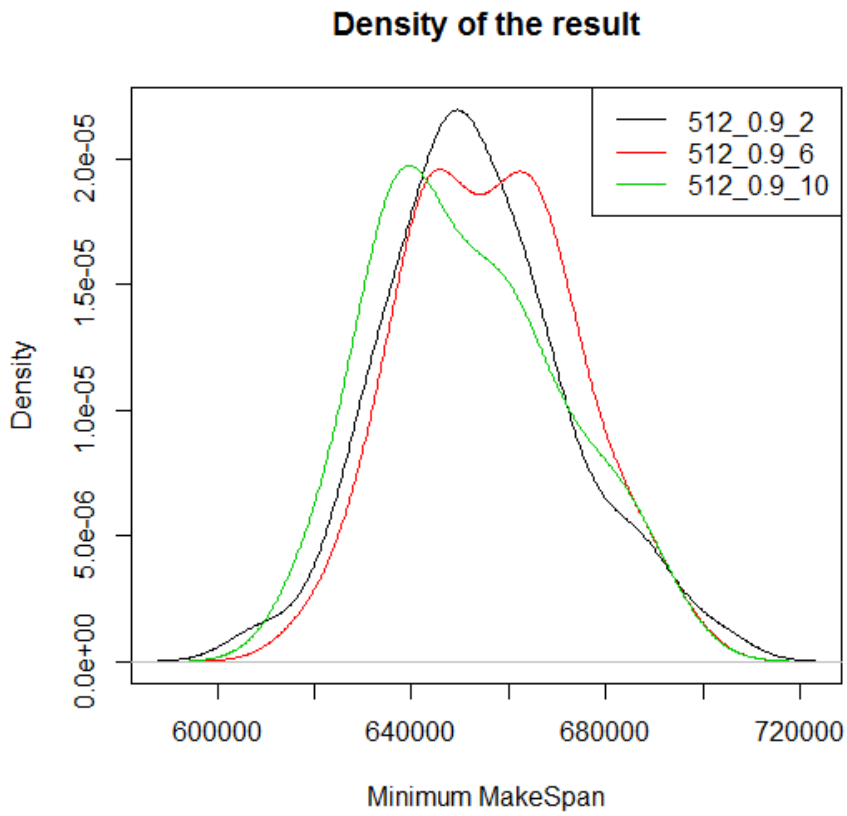

Figure 6: Density for different value of the mutation probability

This plot show that with the value 10/512 the distribution looks like better. Like the distribution are significantly different by the anova we can take 10/512 as parameter for the probability of mutation.

5.6 Conclusion

For the file u_s_hihi_542_16 best parameters are for the probability of crossover 0.9, for the probability of mutation 10/512 and for the size it is 512.

6 Future work

6.1 CrossOver

If we have more time we could make a better implementation of the crossover. For the moment it takes only his result from 2 parents. Maybe with more parents the result will be better.

6.2 Parameter

Maybe with the correct package we can make a program which switch parameters by itself to find the best parameter for specific set.