Methodology

In this chapter, we may explain two main objectives of the project and give some goals of each objective. We also describe the essential details to solve the task allocation problem in the project planning process. There are six main parts as following: the representation of the task allocation problem, the fitness functions for the solutions, the overview of two programs (Genetic Algorithm and (1+1) Evolutionary Algorithm), analysis of the results and testing of works. The description of six main parts are presented below.

3.1 Two main objectives of the project.

The minimisation of the completion times, the goals are shown below:

- Define a sample project plan to represent the task allocation problem.
- Design a simulator to check three constraints and compute the completion times.
- Define the fitness functions for minimisation.
- Implement two programs: Genetic Algorithm and (1+1) Evolutionary Algorithm.
- Compare and analyse two results of Genetic Algorithm and (1+1) Evolutionary Algorithm.

Robustness, the goals are shown below:

- Continue from the first objective.
- Define the fitness functions for consideration in the robustness.
- Compare and analyse the different results of the fitness functions between Genetic Algorithm and (1+1) Evolutionary Algorithm.

3.2 The representation of the task allocation problem.

The representation of the task allocation problem is as a set of possible project plans (the population). The task allocation problem is identified as a sample project plan. The sample project plan includes several resources and contains three constraints. Three constraints of the sample project plan are a team with required skill assigned to a work package, available teams doing work packages and right dependencies. The possible project plans may also base on three constraints of the sample project plan. This part uses a simulator to check three

constraints and compute the fitness values of the possible project plans by using the fitness functions. One possible project plan may be encoded as one individual of the population.

The task allocation problem

In task allocation problem, a task can call a work package. The work package needs specific resources which are limited. The task allocation is the assignment of available resources to each work package. The project plan is a framework of the software project which may show the detail of all work packages such as required skills, dependencies and duration times of work packages. For example, if a team without the required skills is assigned to a work package, the work package may not be completed. Therefore, if the project plan is not generated prudently for a project, many effects will happen in the project. There are the impacts of the task allocation problem that can affect the project as following:

- The software project may not be delivered on time (delay).
- The software project which can be completed with unsuitable resources, it may have low quality and performance.
- The use of resources and the costly investment are made more.
- The project plan may need to be changed to a new plan.

These impacts of task allocation problem also affect reliability and the public image of a company. Therefore, the task allocation problem is one of the crucial problems that should be solved. Search-Based Software Engineering is applied to solve task allocation problem in this project to seek the optimal project plan because the optimal project plan is the important artefact for a project manager to consider suitable resources that are assigned to appropriate tasks of the software project.

Design the sample project plan (Representation)

The sample project plan is designed as a simple problem. A software project may have several plans to complete the works. We are referring the essential instances which are used to design the sample project plan in the following:

- A number of Work Packages = numWP
- A set of teams = {T_1, T_2, …, T_n}
- A set of people = {P_1, P_2, …, P_n}
- A set of specific skills of people in each team = {S_1, S_2, …, S_n}
- A set of durations of Work Packages = {D_1, D_2, …, D_n}
- A set of starting times of Work Packages = {St_1, St_2, …, St_n}
- A set of dependencies of WPs= {(WP_i, WP_j)} ; $0 \leq i \leq n$ and $0 \leq j \leq n$ and $i \neq j$.

A project plan contains various work packages. Work packages are done for finishing a project on time. Each work package requires a specific skill which refer to available resources (people in the teams). Each team is assigned in each work package. Each work package has a starting time and a duration time. For dependencies, some work packages have dependencies. For example, in Figure 2, WP1 needs to complete before WP2 will start and WP2 need to complete before WP3 will start. Moreover, there are no dependencies in WP4 so that WP4 can do in parallel with other work packages.

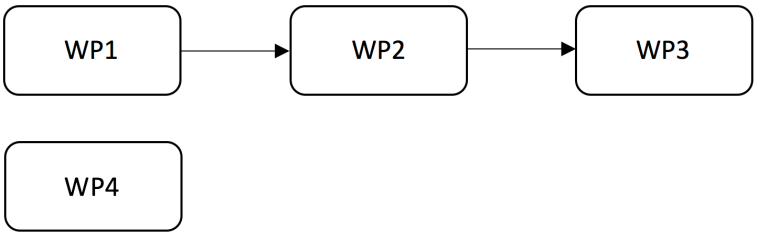


Figure 2. An example of the dependencies.

For design the sample project plan for this project, there are six work packages which are designed in the sample project plan. Each work package requires a specific skill, a starting time, and a duration. Some work packages have dependencies. People in three teams is the available resources. The sample project plan, available resources and the essential instances are presented in Figure 3 and Figure 4. There is the essential description of the sample project plan for this project, and as shown in the following:

- A number of Work Packages = 6 numbers; {WP1, WP2, WP3, WP4, WP5, WP6}
- A set of teams allocating to six work packages: {1,3,2,1,1,3}
- A set of specific skills allocating to six work packages = {2,1,1,2,2,3}
- A set of durations of six work packages = {3,5,9,6,6,3}
- A set of starting times of six work packages = {12,15,20,0,6,29}
- A set of people in three teams: {(P1,P2), (P3,P4), (P5,P6)}
- A set of skills of people in three teams: {(S1,S2), (S1,S3) , (S1,S3)}
- A set of four dependencies: {(WP1, WP2), (WP2, WP3), (WP3, WP6), (WP4, WP5)}

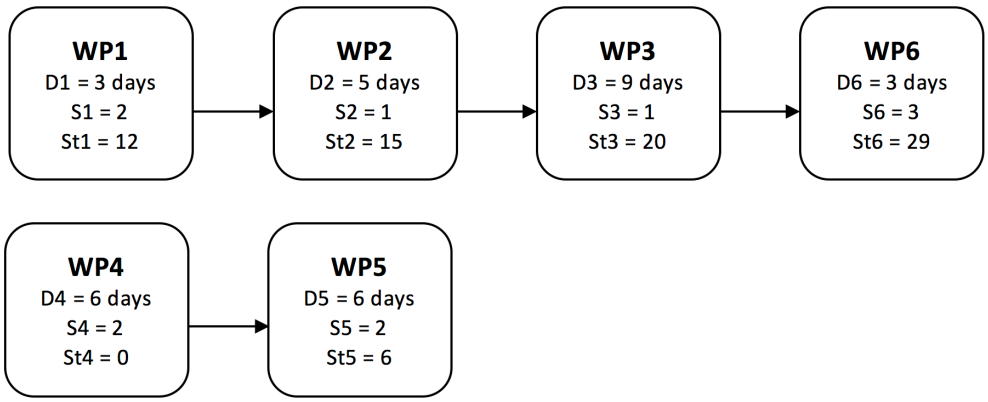


Figure 3. The simple project plan.

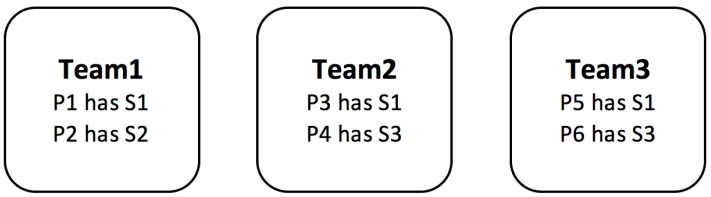


Figure 4. People (P) and different skills (S) of people in three teams.

The Figure 5 displays that three teams can be able to do which work packages. Each work package has a specific skill. There are many possible project plans when we consider between the constraints of right teams and skills of work packages. All work packages can be done by different assigned teams because each team has people that have different specific skills.

WPi Need…	WP1 Skill2	WP2 Skill1	WP3 Skill1	WP4 Skill2	WP5 Skill2	WP6 Skill3
Team1	O	O	O	O	O	X
Team2	X	O	O	X	X	O
Team3	X	O	O	X	X	O

Figure 5 Available teams can do many work packages.

Finding possible project plans with three constraints.

In the basic of software engineering, we need to find the critical path. The critical path is the longest path which has the longest completion time of the project plan. In Figure 6, the critical path of the sample project plan is WP4 → WP5 → WP1 → WP2 → WP3 → WP6 which takes 32 days of completion time. In the sample project plan, there are many possible plans which have different allocations between the available resources and work packages. For example, some possible plans with different allocations are in Figure 7.

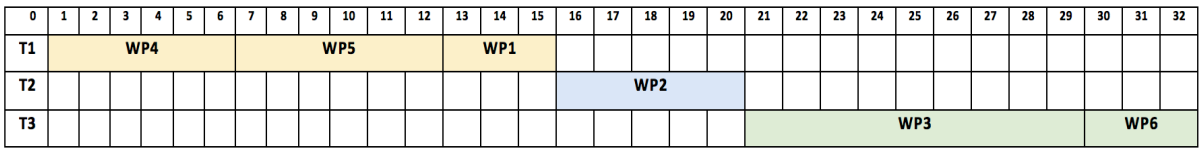
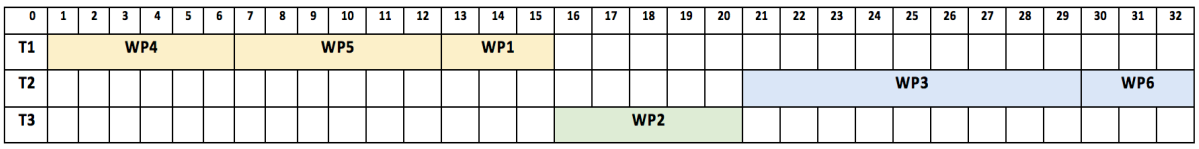
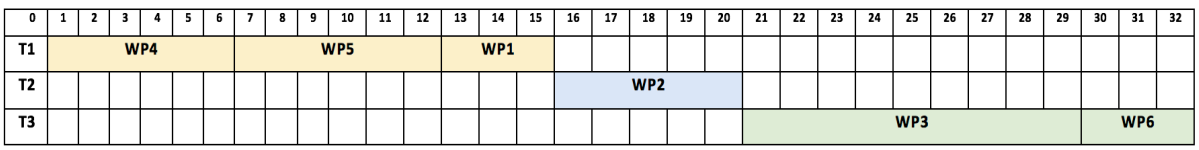


Figure 6. 'WP4 → WP5 → WP1 → WP2 → WP3 → WP6' is the critical path.





…

Figure 7. Some possible plans with different assigned teams to work packages.

Moreover, there are three constraints which are identified for the sample project plan. Three constraints are an available team doing work packages, a right team doing work packages and right dependencies. These constraints involve each other and they will be used for checking the possible project plans in the simulator. The description of three constraints which refers to the sample project plan and available resources is shown below.

The available team doing work package is the first constraint in the sample project plan. This constraint means one team can do only one work package. For instance, Team1 has to do WP4, WP5, and WP1 respectively. If Team1 is doing WP4, Team1 may be unavailable until WP4 completed. Also, if WP4 completed by Team1, Team1 is available again and then Team1 can be able to do the next work package (WP5).

For the second constraint, the right team doing work packages means a team with the required skill assigned to a work package. For example, in the available resources and the sample project plan, Team1 contains skill1 and skill2 so that Team1 can do every work packages without WP6 because Team1 do not have Skill3 which requires by WP6. In addition, Team2 has Skill1 and Skill3, Team2 can do only WP2, WP3, and WP6 because Team2 do not have Skill2 that requires by WP1, WP4 and WP5.

Thirdly, the right dependencies of the sample project plan relate to the order of some work packages in the sample project plan. For example, if WP1 goes to WP2 (WP2 has one dependency), WP2 may start when WP1 completed. If WP1 and WP2 have no dependency, they can be done in parallel, but they need to consider the other constraints namely the available team doing work package and a right team doing work package. For right dependencies, they focus on the order of some work packages which may be done by the different teams.

The allocations and the project plans encoding

There are two allocations in each possible project plan. Two allocations are the allocation of starting times to work packages and the allocation of teams to work packages. Both allocations will be encoded to one individual. The individual contains a set of the twelve numbers of the allocations. There are two parts of the project plans encoding below

First part: Encoding of the allocation of starting time of each work package.

WP1 ← 12 = 12

WP2 ← 15 = 15

WP3 ← 20 = 20

WP4 ← 0 = 0

WP5 ← 6 = 6

WP6 ← 29 = 29

12, 15, 20, 0, 6, 29 ; encoded

Second part: Encoding of the allocation of team to work packages.

WP1 ← Team1 = 1

WP2 ← Team3 = 2

WP3 ← Team2 = 3

WP4 ← Team1 = 1

WP5 ← Team1 = 1

WP6 ← Team3 = 3

1, 3, 2, 1, 1, 3 ; encoded

Two parts of encoding are combined (in Figure 8). The encoding numbers of the sample project plan is "12, 15, 20, 0, 6, 29, 1, 2, 3, 1, 1, 3" that is an "Individual". Each number in the individual can call as "Gene".

The allocation of starting times to work packages						The allocation of teams to work packages					
WP1	WP2	WP3	WP4	WP5	WP6	WP1	WP2	WP3	WP4	WP5	WP6
12	15	20	0	6	29	1	2	3	1	1	3
12, 15, 20, 0, 6, 29						1, 2, 3, 1, 1, 3					
[12, 15, 20, 0, 6, 29, 1, 2, 3, 1, 1, 3]											

Figure 8. An example of two allocation and the project plan encoding.

Considerations in three constraints of the project plan by the simulator.

The simulator is generated in part of the programs by Java. The simulator may check the possible project plans in three constraints: the available team doing work packages, the right team doing work packages and the right dependencies. Therefore, two main functions in the simulator are to check three constraints and compute the completion times of the possible project plans.

For checking three constraints, the simulator may check each possible project plan by considering in each constraint. When the simulator finds wrong constraints, these wrong constraints are the mistakes of work packages in the possible project plan. The wrong constraints are "Penalty". There are three penalties for three constraints. The simulator will compute their wrong constraints to gain the value of penalties. The description of the wrong constraints as following:

In the first wrong constraint, an unavailable team doing work packages. This wrong constraint means that a team is doing more than one work package at the same time. This constraint may consider between a team and any work packages assigned in each team. For example, in Team1, if WP4 is assigned to Team1 at time 0 and WP5 is assigned at time 4, there is an overlapping between two work packages (Figure 9). The overlapping is appeared because WP4 will complete at time 6 while WP5 starts at time 4 so that the overlapping is the wrong constraint. The penalty of this wrong constraint is the overlapping which is at 2.

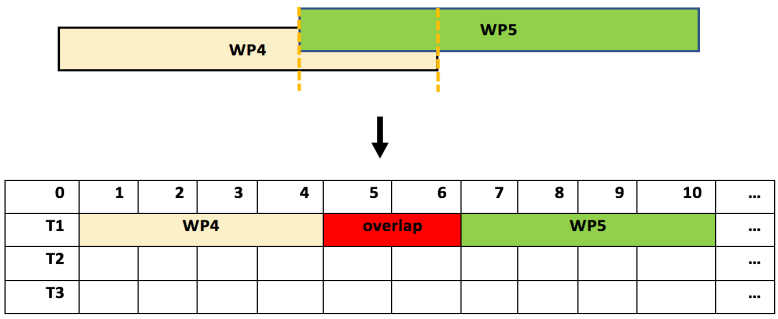


Figure 9. The unavailable team doing work packages (first wrong constraint).

The second wrong constraint is the team without the required skill is assigned to a work package. The allocation of teams to work packages in the possible project plan will be considered. This check refers to some limited skills in three teams and required skills in work packages. Each team contains different skills while each work package requires only one specific skill. For example, in Figure 10, Team2 and Team3 cannot do in WP1 because Team2 and Team3 have no skill2. For example, in the wrong constraint, if Team2 is assigned to do WP1, it means the team without the required skill doing WP1 and then get the penalty at 1. This constraint may check the assigned team in each work package.

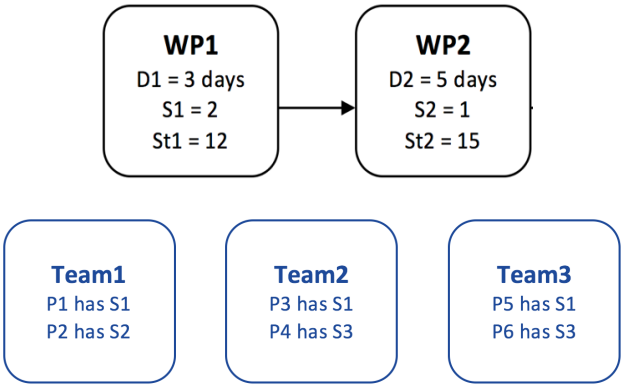


Figure 10. The sample project plan and Teams (second wrong constraint).

The third wrong constraint is wrong dependencies of work packages in the possible project plan. For example, if the current work package (WP1) goes to WP2, WP2 has one dependency which is WP1. WP2 can start when WP1 completed. If the current work package (WP4) has a dependency to WP5, WP5 need to wait until WP4 completed before WP5 will start. For example, in wrong dependencies, if WP2 starts before WP1 will be completed, there is the overlapping. Similarly, if WP5 starts and finishes before WP4, this is the wrong constraint because WP5 should work after WP4 completed. The overlapping of dependencies or the penalty will be computed and as shown in Figure 11. This value is the penalty. These examples illustrate the right and wrong dependencies.

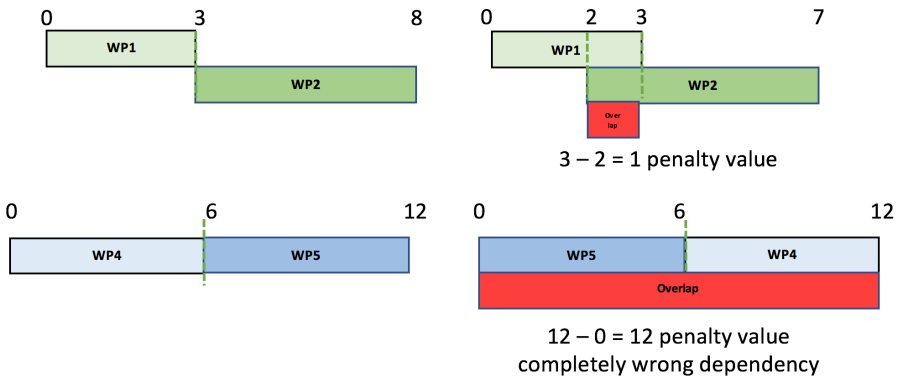


Figure 11. The dependencies of work packages (Third wrong constraint).

The computation of individuals.

Various possible project plans that can be found when two parts of the allocations are running at random by the programs. The allocation of starting times to work packages is random as an amount of the starting times between 0 and 30. The allocation of teams to work packages is random as a number of teams from 1 to 3. The completion times of the possible project plans sometimes contain with no penalty or some penalties.

When one possible project plan is chosen at random, the possible project plan is checked three constraints and computed the completion time of the project plan by using some functions. Figure 12 illustrates the example of the completion times of some project plans. For example, if the possible project plan has some penalties, the penalties will be added into the completion time of the project plan. The functions for checking and computation the completion time are the fitness functions that will describe in the next phase.

starting times to work packages	teams to work packages	Status	Fitness values
…	…	…	…
13, 16, 21, 0, 7, 30	1, 2, 3, 1, 1, 3	Accepted, Penalty: 0	32
…	…	…	…
13, 16, 21, 0, 7, 30	1, 3, 2, 1, 3, 1	Accepted, Penalty: 2	32 + 2 = 34
…	…	…	…

Figure 12. Examples of the completion times of some project plans.

3.3 The Definition of the fitness functions

The fitness functions are used in the simulator of each program to check and calculate the fitness values of the individual (each project plan). The fitness values will be used to analyse in both objectives. There are two fitness functions for two objectives namely the fitness function for the minimisation of the completion times of the project plans and another fitness function for the consideration of the robustness of the project plan. The fitness values of each individual may be different because the values depend on starting times, duration times and the penalty of three constraints.

Fitness1: Minimizing completion time.

The first fitness function for the task allocation problem is the minimisation of the completion time of the project plan. The program computes the duration times of each work packages and keeps the longest duration time of the work package as the completion time of the project plan. The program checks and computes the penalties in three constraints. Therefore, the fitness value is the completion time plus with all penalties. This objective compares the results of two algorithms between Genetic Algorithm and (1+1) Evolutionary Algorithm. The full functions are shown below.

$$\boldsymbol{d_i = St_i + W_i}$$

d_i is duration of the work package i

St_i is the starting time of the work package i

w_i is time of doing the work package i

$$\textbf{fitness1} = \mathbf{d_{max}} + \boldsymbol{p1 + p2 + p3}$$

fitness1 is the fitness value of the project plan.

d_{max} is a maximum duration of work package as the completion time the plan.

m is the number of work packages.

p1 is the penalty value of wrong teams assigned to work packages.

p2 is the penalty value of teams doing work packages at the same time.

P3 is the penalty value of wrong dependencies of work packages.

$$\mathbf{A_{fitness}} = \frac{\sum_{i=1}^{m} fitness1}{m}$$

$A_{fitness}$ is an average of all fitness values.

m is the number of iterations.

fitness1 is the fitness value of the project plan.

For the fitness function, starting times, duration times and three penalties are very important factors for computation the completion time of one possible project plan. The simulator will compute the completion time of the plan that based on these factors. When the program working, the program may consider three factors (penalty values) and calculates the completion time to get the fitness value.

Fitness2: Robustness

The second fitness function is taking into account the robustness of the project plan. When we consider the robustness, we may focus on unexpected resources which may occur in the project plans. The robustness of the project plan may compute the completion times of the possible project plans which based on uncertainties or unexpected resources. When the unexpected resources occur in the project plan, the completion times of the project plan may also change. This fitness function also need to compare between the results of Genetic Algorithm and results of (1+1) Evolutionary Algorithm. The fitness function of fitness2 is shown below.

fitness2= fitness1 + Noise

fitness2 is the completion times with noise.

fitness1 is the fitness value of the project plan.

Noise is the random number which refers unexpected resources in the plan.

The programs may generate the noise as unexpected resources and then the program records all results of both algorithms. There is a comparison between the fitness values of the completion times without the noise and the fitness values of the completion times with the noise. If the noise cannot affect the completion times of the project plan too much, it means that the project plan is robust.

3.4 Pseudo-codes and Programming

There are two programs for this project. Two programs are Genetic Algorithm and (1+1) Evolutionary Algorithm. These programs are applied to seek the optimal or near optimal project plan which based on two fitness functions. The fitness values in each program may be considered the minimisation of the completion times of the project plans and takes into account the robustness of the project plan. These programs work many iterations and then print out the completion time, the individual of the last iteration and the average of the completion times in all iterations. This part may illustrate the pseudo-codes of the fitness functions (fitness1 and fitness2), Genetic Algorithm and (1+1) Evolutionary Algorithm.

Pseudo-codes for fitness1: the minimisation of the completion times.

Individual = {a set of starting times to WPs + a set of teams to WPs}

First constraint: available team doing work packages.

```
CheckSameTime (durations of WPs, individual)
…
For (each WPi; i=1; i<7) {
        Team = which team doing WPi.
        for (starting time of WPi) {
                finish time = starting time + duration of WPi
                If (Team is now available) {then…
                        update status of Team to unavailable
                        from starting time to finish time
                        by add a name of work package.
                        }
                else // it means Team is unavailable. {
                        Update status of Team as well
                        count the penalty to penalty1.
                        }
        }
} return penalty1.
```

Second constraint: a right team doing work package.

```
CheckSkillandTeam (skill of WPs, individual)
…
for (each team of WPi in the individual; i=1; i<7) {
        Team = each team of WPi.
        Skill = each skill of WPi.
                for (each skill in Team) {
                        if (Team contain Skill) {
                                it is correct.
                                update status to correct.
                        }
                }
        if (status is not correct) {
                add penalty to penalty2.
        }
        do in the next round.
} return penalty2.
```

Third constraint: dependencies.

```
CheckDepend (duration of WPs, dependencies of WPs, individual)
for (each WPi; i=1; i<7) {        //each work packge
    for (each WPj; j=1; j<7)     //check each WPi has go to all WPj.
        if (have dependency between WPi to Wpj) {
                if (starting time + duration of WPi > starting time WPj) {
                 get wrong dependency.
                Penalty3= (starting time+duration of WPi)-starting time WPj.
                } else it is right dependency.
        }
    } do next round for check WP(i+1) has dependency to other WPj
} return total of penalty3.
```

Computation: Duration times.

ComputeDuration (duration of WPs, individual)

For (each WPi) {

Compute duration time of each WPi by starting $time_{WPi}$ *+* $duration_{WPi}$

If (which duration time of WPi is maximum) {

Keep WPi and the maximum duration time.

}

} return maximum duration time.

Computation: fitness1.

…

//in each individual.

check first constraints (individual) to get penalty1.

check second constraints (individual) to get penalty2.

check three constraints (individual) to get penalty3.

Compute the duration (individual) and keep maximum duration time

// the values of all parameters are computed to fitness1.

fitness1 = maximum duration time+ penalty1+ penalty2+ penalty3.

…

Pseudo-codes for the fitness function: Robustness.

Individual = {a set of starting times to WPs + a set of teams to WPs}

Computation: fitness2.

…

// the values of all parameters are computed to fitness1.

fitness1 = maximum duration time+ penalty1+ penalty2+ penalty3.

Then…

Add the noise to fitness1 to get the values of fitness2 at random.

fitness2 = fitness1+ Noise.

// The noise is a random number (0-9).

// The noise depend on the longest duration of one work package in the plan.

…

Genetic Algorithm

The program of Genetic Algorithm may run many iterations. The results also compute and recode from two fitness functions for analysis robustness of the optimal project plan. The pseudo-code of Genetic Algorithm is as following.

```
Genetic Algorithm ():
    Initialization (population)
    Evaluation (the fitness evaluation)
    for (any iterations) do
        {
        for (each offspring; i=1 to λ) do  // λ is the number of the individuals
            {
            Selection (two parents x and y by tournament selection)
            Let z = crossover (x, y).
            Let next offspring be y = mutate(z).
            evaluation (offspring y).
            // evaluate the fitness for next generation
            }
            // go to the next generation.
        }
    Out of the loop when stopping condition appeared.
```

Pseudo-code of Genetic Algorithm above explains the main programming of Genetic Algorithm. There is a process that is the specific process in the simulator for this project such as the calculation the completion times, checking the constraints. In Genetic Algorithm, selection is that two parents are chosen by tournament selection. Crossover process is changing some parts of the individual. Mutation is the swapping any numbers of the individual. The Genetic Algorithm program may run various iterations for finding the optimal or near optimal project plan.

In crossover, we specify one crossover point and take 50% (0.5%) of crossover the individuals in the program. We decide 0.5% of crossover because the individuals include two different

allocations which have different length of the random numbers so that crossover phase should depend on the two allocations (½). Two individuals (two parents) may be crossover at 50%, then six genes of one individual are changed with six genes of another individual to get a new individual.

Furthermore, in mutation, we identify 100% (1%) of the mutation, it means all genes of the new individual are completely changed. We decide 1% of the mutation for managing the individuals to reproduce in the next generation because a half of the individual which is the allocation of teams to work packages has very small length of the random number of teams (1,2,3). Therefore, to avoid the poor reproduction, we decide to set 1% mutation.

(1+1) Evolutionary Algorithms Computing

This algorithm is a basic technique in Search-Based Software Engineering. The work of (1+1) Evolutionary Algorithm is simple than Genetic Algorithm. (1+1) Evolutionary Algorithm uses only one individual while Genetic Algorithm uses large individual. The program of (1+1) Evolutionary Algorithm may run many iterations as well. The results also compute and recode from two fitness functions for analysis robustness of the optimal project plan. Pseudo-code of (1+1) Evolutionary Algorithms can be seen below.

```
(1+1) Evolutionary Algorithm ():
    Initialize (individual)
    for (any iterations) do
        {
            Selection (individual X)        // one selected population (X)
            Mutation (individual X)         // mutate X → X'
            if (X' is better than X) then   //Get the better individual
            {
                repeat (X ← X')             // next individual is X'.
            } else keep X                   // next individual is X.
            (next generation)
        }
    Out of the loop when stopping condition appeared.
```

Pseudo-code of (1+1) Evolutionary Algorithm presents the main works of (1+1) Evolutionary Algorithm. The specific process is the simulator which may be the calculation of the completion time and checking three constraints. In (1+1) Evolutionary Algorithm, mutation process is set at 100% of mutation. (1+1) Evolutionary Algorithm program may run various iterations for finding the optimal or near optimal project plan. Figure 13 illustrates the overview of two programs for this project.

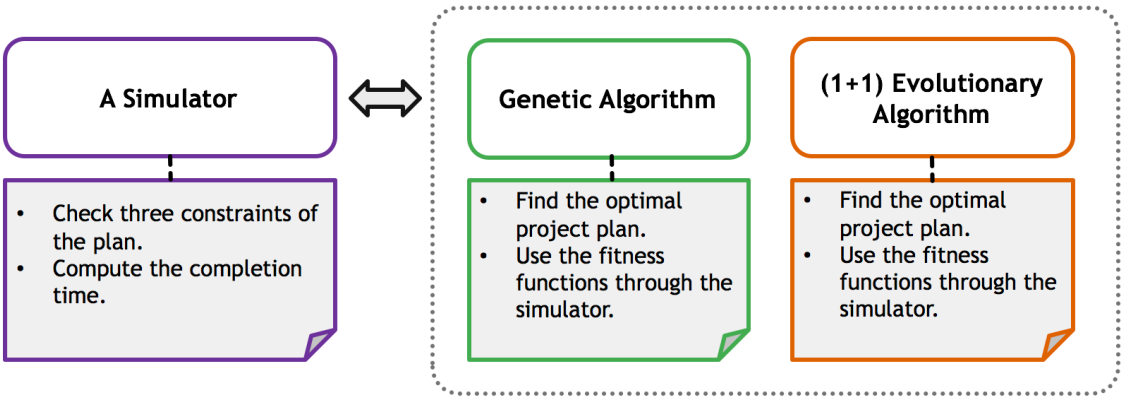


Figure 13. The overview of the programming.

3.5 The explanation of analysis of the results.

The results of both programs may be compared and be analysed for two objectives. The results of the completion times in each algorithm may be varied and the results of the fitness1 and fitness2 should present the minimisation of the completion times of the project plans. The results of fitness1 in both programs may be compared and analysed for the minimisation the completion times (objective1). The results of fitness2 in both programs are the computation times of the project plans with unexpected resources. These results may be compared and analysed in part of the robustness of the project plans.

Moreover, the averages of the completion times in many running of each program are also used to analyse the performance in both objectives. This analysis uses the Wilcoxon Rank Sum Test for the consideration. The Wilcoxon Rank Sum Test is used for analysis two objectives. by using two sample data. This test may consider which programs is effective and which program is robust. For the robustness of the project plan, it means that the project plan

should have a good performance and also should be strong enough when unexpected resources or problem uncertainties occur in the project plan.

For the Wilcoxon Rank Sum Test, it is used to analyse the average results in both algorithms to observe which algorithm is more effective than another one. In basic work of the Wilcoxon Rank Sum Test, there are two data which have the number of the averages of the fitness values from 0 to 600 iterations of Genetic Algorithm and (1+1) Evolutionary Algorithm. The program may calculate the p-value to analyse the performance of both algorithms. For example, if the p-value is more than 0.5, it means that it is strong to give its effective enough. On the other hand, if the p-value is lower than 0.5, it means that it is not strong enough to say its effective.

3.6 Testing

In the simulation, Genetic Algorithm and (1+1) Evolutionary Algorithm are described the important descriptions. Even though their processes are generated automatically by the programs, the results may be wrong. Therefore, we need to check all processes that each process can work correctly, and the result of each process is outputted right. Testing is an essential procedure for checking their works. We may use software testing to check the accuracy of the works in the programs.

We use a unit testing and system testing for validation the programs. The unit test is the smallest testing that can checks for validity in each unit or component of the works of both programs. The system testing is used to validate all works of Genetic Algorithm and (1+1) Evolutionary Algorithm. We use hand checking to check the completion times of the possible project plans by drawing the paths of the project plans. There are three parts which need to check validity. Three parts are simulation, Genetic Algorithm and (1+1) Evolutionary Algorithm.

The simulation testing

- The simulator: hand checking (Figure 14)
- The simulator: unit testing (Figure 15)

In the program, this part may check three constraints and duration to calculate the completion time of the project plan. For hand checking, we may draw the path of the project plan and assigned tasks follow the allocation of starting times to work packages and the allocation of teams to work packages. We may check the constraints and calculate the completion time of the project plan. For example, the individual "12, 15, 20, 0, 6, 29, 1, 2, 3, 1, 1, 3" is calculated by the programs and by hand. Figure 14 presents the hand checking of the same individual which displays the output of the programs. Figure 15 also shows the outputs of one project plan which includes the individual, penalty1, penalty2, penalty3, and the fitness value of the plan. If the results of the unit testing and hand checking are same, it means that the work of the programs is right.

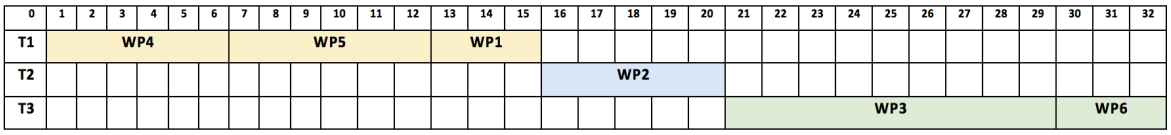


Figure 14. The completion time of the individual by hand checking.

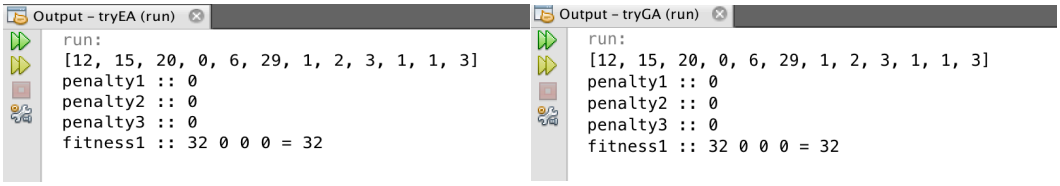


Figure 15. The completion time of the same individual of two programs

Genetic Algorithm testing

- Programming: unit testing (like in the simulation testing)
- Programming: system testing

  This part may check each process of Genetic Algorithm. The program of Genetic Algorithm may be separated as a unit for checking validity, then the program may be checked in all processes together as the system testing. The results of each unit may be printed out for check the results is right. The program can run the completion times of project plans in each iteration and also output the optimal completion time at the last iteration (Figure 16).

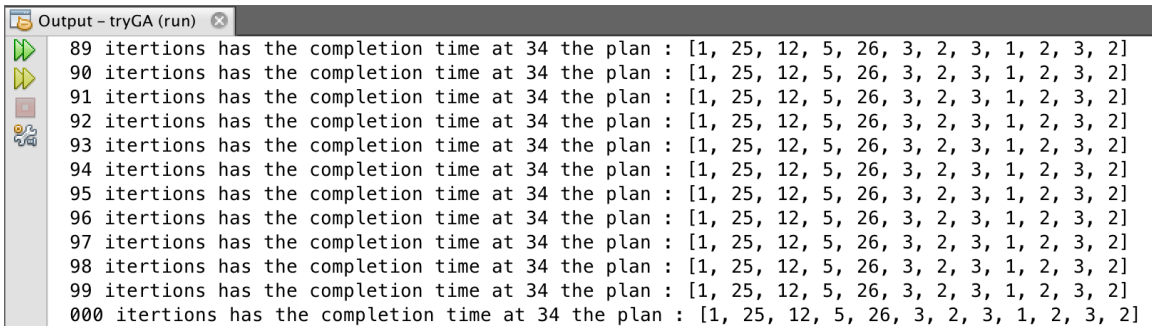


Figure 16. The output of the GA program.

(1+1) Evolutionary Algorithm testing

- Programming: unit testing (like in the simulation testing)
- Programming: system testing

  This part may check each process of (1+1) Evolutionary Algorithm. The program of (1+1) Evolutionary Algorithm may be checked in each component for checking validity. The results of each unit may be printed out for check the results is right. The program may be checked in all processes together as the system testing. The program can run the completion times of project plans in each iteration and also output the optimal completion time at the last iteration (Figure 17).

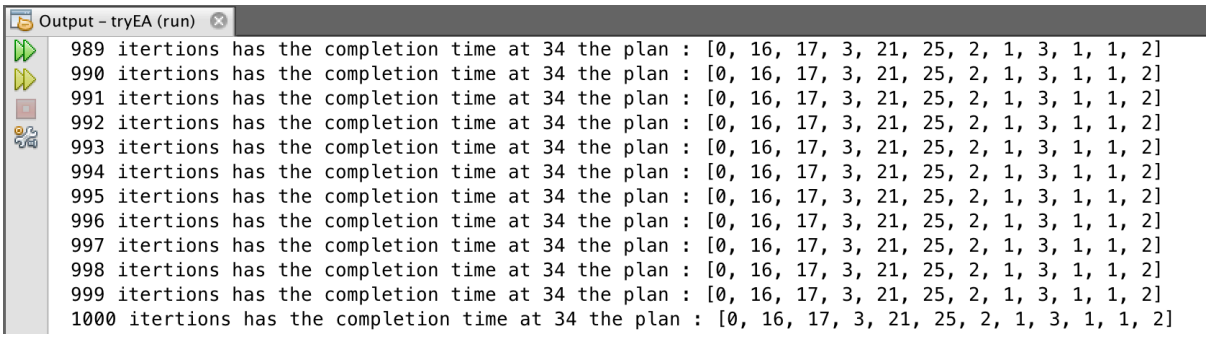


Figure 17. The output of the (1+1) EA program.