Technical university Delft
Faculty Electrical Engineering, Mathematics en Computer Science
Delft Institute of Applied Mathematics

An efficient metaheuristic to solve the project portfolio selection and scheduling problem for industrial projects

Report on behalf of
Delft Institute of Applied Mathematics
as part of obtaining

of the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

Sebastiaan van Schagen

Supervisors

Dr. ir. J.T. van Essen
Ir. T. van der Beek

Other committee member

Dr. Y. van Gennip

July 14, 2020 Delft

Abstract

Industrial companies aim for optimizing profit from delivering project outcomes. Maximizing profit relies on optimization of using resources, production capacity and available time. To reach this goal, companies are typically reliant on planning and production schedules. This problem is known as the project portfolio selection and scheduling problem (PPSSP). The PPSSP can be solved using an integer linear programming (ILP). However, solving an ILP for complex cases with a large number of variables takes a lot of time. Solving the PPSSP using a heuristic method provides a good alternative. Due to the structure, an adapted version of variable neighborhood search (VNS) is chosen as heuristic method. The adapted VNS is combined with tabu search to obtain an alternative for solving the ILP. The solution obtained with the heuristic method is represented as an activity list which is a specified order of planning tasks. The schedule which is represented by the activity list can be obtained using the serial schedule generation scheme (SGS). Serial SGS represents every optimal schedule in the non-preemptive case. When preemption is allowed, schedules might not be represented by an activity list in all cases. The overall profit of the optimal schedule is never smaller than in the non-preemptive case. Because of this, a solution is represented by a selection and an activity list from which the schedule can be obtained through using the preemptive serial SGS. The heuristic is used to obtain some results for less complex instances which are compared to the results obtained by solving the ILP. In some cases, the ILP could not solve the problem in a short time span. It turns out that the performance of the adapted VNS in combination with tabu search provides good estimates close to the real optimum.

Contents

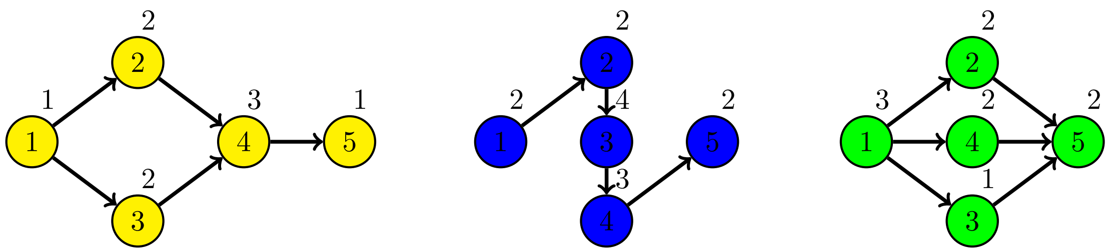
Page

1 Introduction

An industrial company, like a shipbuilder, tries to obtain a profit as high as possible while satisfying customers. The industrial company gets industrial customer projects which have a time limit. Every project consists of different tasks like constructing, assembling, designing, etc. Every task has its own duration, production capacity and requires its own amount of resources. In most industries, these resources are limited and shared between different projects. Some companies form a high level planning instead of a planning in full detail. A company selects a combination of projects to be executed based on profitability and resources required. Not all the possible projects can be completed due to the limiting factors, like resources. From the selected projects, a schedule is made, describing when is worked on which task.

Every project consists of different tasks, the tasks of selected projects are scheduled in such a way the resources and production capacities are never exceeded. During the selection of projects, some non-selected projects might have been turned down due to insufficient resources. However, projects rarely use all resources at the same time, which may result in under utilized capacity while working on the selected projects. This result in leftover resources, which may have been used for other purposes, resulting in higher profits. For example, projects which most differ in resource and capacity use, could be completed concurrently. A schedule should never exceed resource limits and production capacity. Optimizing profit relies on combing available resources, production capacity and time efficiently. This research aims to find an efficient method for maximizing profit using the project portfolio selection and scheduling problem (PPSSP).

In Figure 1, an example with three projects and their profitability is given. This example simplifies the problem by taking an one dimensional instance where only one type of resource is considered. Every project consist of five tasks, denoted by the values in the nodes. Every task has a duration which can be found diagonally above the task. For example, task 3 of project b has a duration of four periods as shown in Figure 1b. As noted, every task requires resources and the number of resources required can be found in Table 1. This number corresponds to the number of resources required every time period there is worked on the task. If we look again at task three of project b, there are five resources needed each period in order to complete this task. The directed connections denote the precedence constraints which is an order of completing the tasks. To finish a project, all tasks need to be completed. Every project execution can contain idle time, which means when a task is completed, the next task can be scheduled for later execution. Likewise, since preemption is allowed, the time periods in which is worked on a task do not have to be consecutive. In this example we use thirteen time increments, during which in each time increment five units of the same resource type are available.



(a) Project a, profit: €90,- (b) Project b, profit: €120,- (c) Project c, profit: €100,-

Figure 1: The three projects of which a selection and schedule is made.

Project	**Tasks**				
	1	**2**	**3**	**4**	**5**
a	1	3	2	2	1
b	2	2	5	3	1
c	3	2	1	3	2

Table 1: Resources needed for every task of the project in Figure 1.

Combining all input, a schedule can be created. First, we try to combine projects b and c, as these are the most profitable projects. To complete task 3 of project b it requires all resources. Therefore, other project cannot be worked on. It turns out, project b and c can never be combined. Meanwhile, project a can be combined with project b as shown by the schedule in Figure 2. Every rectangle has the width of one time increment and the number in the rectangle denotes to which task it corresponds. The resource limit is never exceeded. As displayed in Figure 2, task 3 of project a uses preemption. The precedence relations shown in Figure 1a and Figure 1b are visible in Figure 2. It turns out, this is an optimal selection and schedule for this example. There are other combinations possible, hence the solution is not unique. Finding the optimal solution for three projects is easy, however as the number of projects and/or tasks increases, solving the problem becomes significantly more difficult and time consuming.

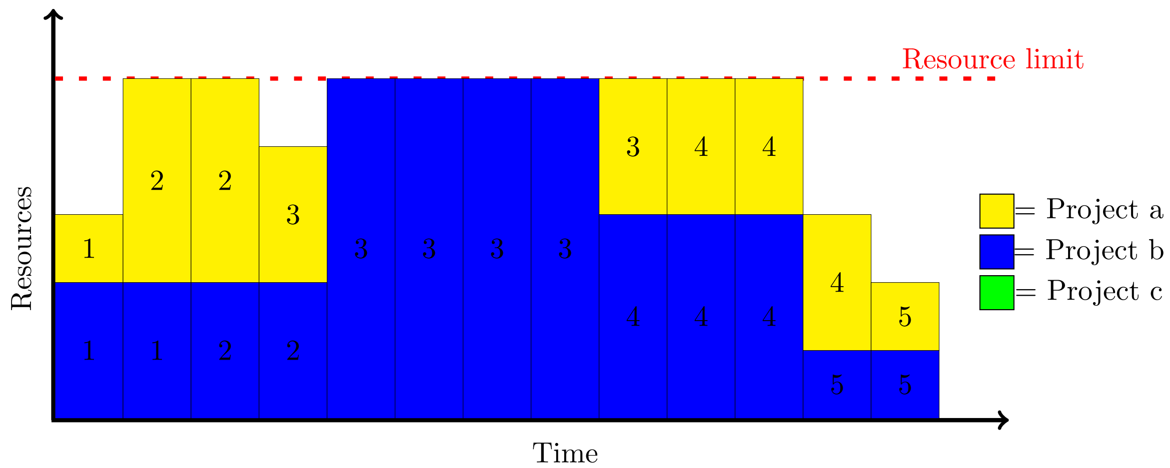


Figure 2: A schedule of the projects in Figure 1 satisfying the resource constraints.

To solve the PPSSP, we start with a literature review in Section 2, where we discuss the PPSSP itself and the resource constraint project scheduling problem (RCPSP) which is a special case of the PPSSP. In addition, we discuss some heuristic methods to solve the PPSSP. Then, in Section 3 the PPSSP is introduced in more detail. For the PPSSP, an integer linear programming (ILP) is formulated. Solving the ILP results in an optimal selection and planning for the given data. This works for a few small projects, however more and larger projects results in a larger computation time. In Section 4 we introduce a heuristic method to obtain a high profit for larger instances. At last, we compare the values obtained with the heuristic method with the solution of the ILP solver. We already know the profit obtained with a heuristic method is always less than or equal to the profit obtained by solving the ILP to optimality. This follows from the heuristic method being a method without a guarantee of optimality, while solving the ILP has a guarantee of optimality. However, solving the ILP to optimality takes a lot more time compared to the heuristic method.

2 Literature review

The project portfolio selection and scheduling problem (PPSSP) (Manish, Mittal, Gunjan, & Dheeraj, 2018) is a widely used approach to generate higher profits for companies. This problem can be solved by formulating it as an integer linear programming (ILP) and using an ILP solver to obtain an optimal solution for this formulation. This way of solving can take a lot of time, depending on the number of variables and constraints. In the case of companies with a small number of projects, the ILP should work fine and an optimal schedule is returned. As the numbers of possible projects and/or tasks grows, more variables are needed which results in a longer computation time. A heuristic method becomes a good alternative. Before the heuristic methods are considered, resource constraint project scheduling problem (RCPSP) is considered which is a special case of the PPSSP.

2.1 Project portfolio selection and scheduling problem

The input of the PPSSP consist of projects, project capacities, resource requirements, resource capacities and a time span. Every project has its own completion time, requirement of resources and precedence relations. The precedence relations define an order in which the tasks need to be completed. Not all the projects have to be scheduled, the PPSSP searches for a selection of the projects and a schedule for which the profit is maximal. All tasks in this schedule must be completed in the given time while never exceeding the resource constraints and project capacities. The schedules can be represented by an activity list (Moumene & Ferland, 2009) which is a vector determining the order of planning. The PPSSP can be solved by solving a integer linear programming (Ghasemzadeh, Archer, & Iyogun, 1999). It can also be solved using a heuristic method like a combinatorial auction algorithm (Shou & Huang, 2010) or other algorithms. As there is much more relevant literature about the RCPSP, we take a look at this problem.

2.2 Resource constraint project scheduling problem

The RCPSP is a special case of the PPSSP. The input of the RCPSP is the same as for the PPSSP. The big difference is that where the PPSSP selects projects and schedules these projects, RCPSP makes a schedule of the given input projects. The aim of the RCPSP is to minimize the make span, so the time needed to complete all the projects is minimized. The schedules can be represented by an activity list. The RCPSP can be changed in such a way that the maximal profit is obtained instead of the minimal make span (Liu & Wang, 2008). The RCPSP can be roughly spread into two categories. In the first category preemption, a break in a task, is allowed (Wall, 1996), while in the second, preemption is prohibited (Talbot, 1982). The same categories can be used for the PPSSP. The RCPSP can be solved using normal optimization methods, such as branch and bound (Brucker, Drexl, Möhring, Neumann, & Pesch, 1999). However, the RCPSP is an NP-hard problem (Demeulemeester & Herroelen, 2002) and since the PPSSP is a generalization of the RCPSP, the PPSSP is NP-hard as well. To give an idea of heuristic methods used to solve the RCPSP, genetic algorithms (Wall, 1996; Sawant, 2016), tabu search (Tsai & Gemmill, 1998; Thomas & Salhi, 1998) and other methods (Wall, 1996; Carazo et al., 2010) can be used to solve the RCPSP. We discuss both the genetic algorithms and tabu search as these methods work for PPSSP as well. Variable neighborhood search is also discussed.

2.3 Metaheuristics

The metaheuristics we consider make use of initial solutions to obtain a new solution for the problem. The choice of initial solutions is important, since a better initial solution has a greater chance to converge faster to the real optimum. The heuristic algorithms aim to improve the initial solutions by making small changes. By doing so, new solutions are obtained, however these might be worse solutions. The overall best solution converges to a local optimum. This full process can be done in different ways. We discuss the genetic algorithms, tabu search and variable neighborhood search and how they can be applied to the RCPSP and PPSSP.

2.3.1 Genetic algorithms

The idea of genetic algorithms, developed by J. Holland (Holland, 1992), is based on evolution theory. A group of organisms, living individuals, evolves by means of three processes: reproduction, natural selection and mutation (Whitley, 1994). Reproduction is the creation of offspring by two different organisms. Survival of the fittest is applicable in the nature, this is also known as natural selection and the best adapted individuals survive. Lastly, mutation is an arbitrary change in an organism, which could happen at any moment. What members of the population can reproduce the most is determined by natural selection. Best individuals get more offspring which is exactly as desired. Reproduction is influenced by two individuals, their genetic material is combined. This does not mean that only individuals with good aspects are born. Due to the reproduction, there will be more variety in genes which forms the genetic material. This more varied gene pool is again influenced by natural selection. This results in improved individuals, which is called evolution.

The evolution process can be implemented to work with the resource constraint scheduling problem (Hartmann, 1998). Every possible solution is used as a chromosome and the initial solutions are defined as a population of the individuals. Before the process can start, a solution representation should be formulated. For example, an activity list or a priority list. This formulation allows both systematic and arbitrary changes in the solution, which are used for reproduction and mutation. The reproduction is defined as taking a part of both parent solutions and combining these, resulting in a new solution. The mutation is an arbitrary change in the solution, still satisfying the precedence constrains. The algorithm takes the initial solutions and generates new solutions, the offspring. This process keeps going until a stop criterion is met. This can be something like, the maximal profit has not changed in several iterations or simply if a number of iterations is reached. In Figure 3, the process of a genetic algorithm is shown in a flowchart.

A genetic algorithm is dependent on some parameters namely rate of mutation, rate of reproduction and the size of the initial population (Shorman & Pitchay, 2015). Rate of mutation and rate of reproduction are used to specify how fast an individual mutates and reproduces, respectively. When these rates are high, there are a lot of changes happening between different generations. For low values, the changes are small. If the rates are too high, too big steps are taken and some good solutions are not considered. On the other hand, if the rates are too low, the process might take more time. The same holds for the size of the initial population. A too high value results in more computations and hence a longer computation time, while a too small value might result in not obtaining a good estimate. Therefore, the parameters should be chosen carefully.

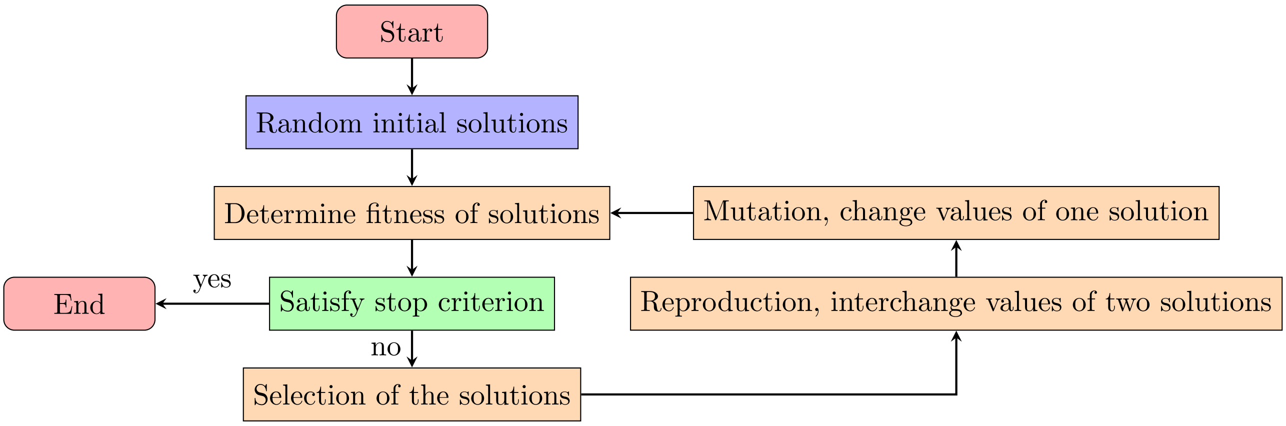


Figure 3: Flowchart of a genetic algorithm (Abdeslam et al., 2014).

2.3.2 Tabu search

Tabu search (Glover, 1990), developed by F. Glover, is another metaheustic which can be used to solve the PPSSP. Tabu search improves the value of an initial solution. A solution representation should be chosen which can be the same formulation as used for the genetic algorithms in Section 2.3.1. The tabu search makes use of a tabu list which is filled with the previous visited solutions. The tabu list is used to prevent the tabu search from going into cycles. Instead of adding the full solutions it is better to only add a characteristic of a solution to save some memory. To get from one solution (S) to another solution (S_n), tabu search uses neighbors of the solutions. The neighborhood of a solution is defined as all the solutions that can be reached by making a predefined change to the original solution. This change is denoted as a move and is dependent of the problem itself. An example of a move is swapping two or more things, adding or removing things. In every iteration, the neighbor with the best value which is not on the tabu list is taken which might be worse than the initial solution. Instead of checking all the neighbors of a solution, it is also possible to check a subset of the neighbors. As the iterations increases, the number of elements on the tabu list increases. The tabu list can have a maximal length and when the tabu list has reached this maximum length, the first element is removed and the new neighbor is added. This predefined maximum length is better for the memory, but also for the tabu search itself. Sometimes, an old solution should be encountered again to obtain a new neighbor of this solution. This could increase the overall optimal value obtained. The tabu search terminates when a stopping criterion is met. For example, this stopping criterion can be, a number of iterations is reached or the maximal profit has not changed in a number of iterations. In Figure 4, the process of tabu search is shown in a flowchart.

Tabu search is dependent on the choices of the maximal length of the tabu list, the number of neighbors checked in each iteration and the information added to the tabu list. The tabu list having a maximal length might result in going into a cycle of minimal length equal to the length of the tabu list. This can be solved by taking the maximum length bigger. When this length is too big, the best solution might not be reached at any time. The time of checking if a neighbor is on the tabu list increases in time if the tabu list is bigger. Tabu search can be applied to RCPSP by defining a neighborhood function (Thomas & Salhi, 1998). The tabu search can also be used in solving PPSSP by changing the neighborhood function, since also the selection needs to be taken care of. For example, we can add, remove or replace a project but also swap or reschedule a task. This is introduced in more detail in Section 4.

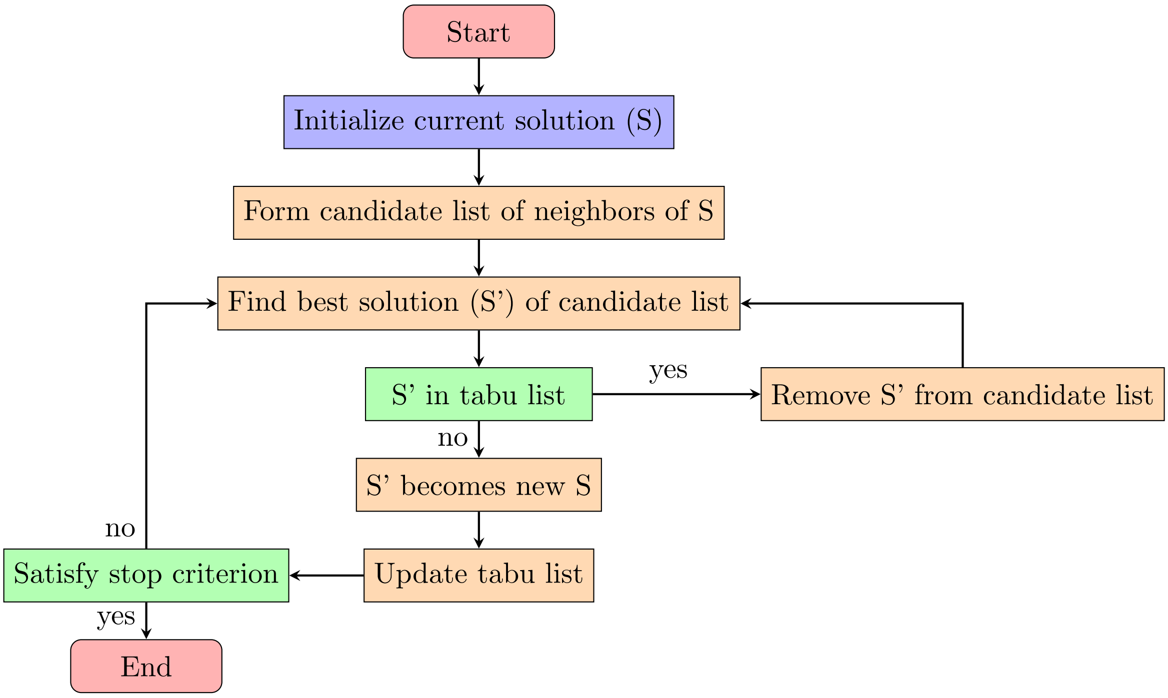


Figure 4: Flowchart of tabu search (Wang et al., 2017).

2.3.3 Variable neighborhood search

Variable neighborhood search (VNS) (Hansen & Mladenović, 2001) is a metaheuristic, like tabu search, based on neighborhoods of solutions. Instead of only one neighborhood of a solution, multiple neighborhoods are considered. The process of finding a neighboring solution in one neighborhood structure is the same as used in tabu search. VNS is a metaheuristic which uses another metaheurstic. There is searched for the best solution in every neighborhood structure which is done with a local search algorithm or metaheuristic. Both genetic algorithm and tabu search could be used for this local search, however there are also other options. Changing neighborhoods can happen at different moments in the process. For example, every second iteration, the neighborhood is changed, but it can also happen every twentieth iteration. For a problem, the set of neighborhood structures (N_k) is defined with $k \in \{1, .., k_{max}\}$ where k_{max} is the number of neighborhood structures. Before the process starts, an initial solution and the neighborhood structures must be given. Of the initial solution (S), a neighbor (S') is created in the first neighborhood. A local search metaheuristic is used to generate new solutions based on S', of which the best is remembered (S''). If the value of S'' is better than the value of S, S is replaced with S'' and the process is started again in the first neighborhood structure. Else, a new neighbor (S') of initial solution S is found in the next neighborhood structure and local search is done on S' and this process keeps repeating itself until a better value than S is found or all neighborhood structures are considered on S. If this happened and the stopping criterion is not met, the neighborhood structure is set back to the first neighborhood structure. This process keeps going until the stopping criterion is met, this can be a number of iterations or a maximal profit that has not changed in a number of consecutive iterations. In Figure 5 the process of VNS is shown as a flowchart.

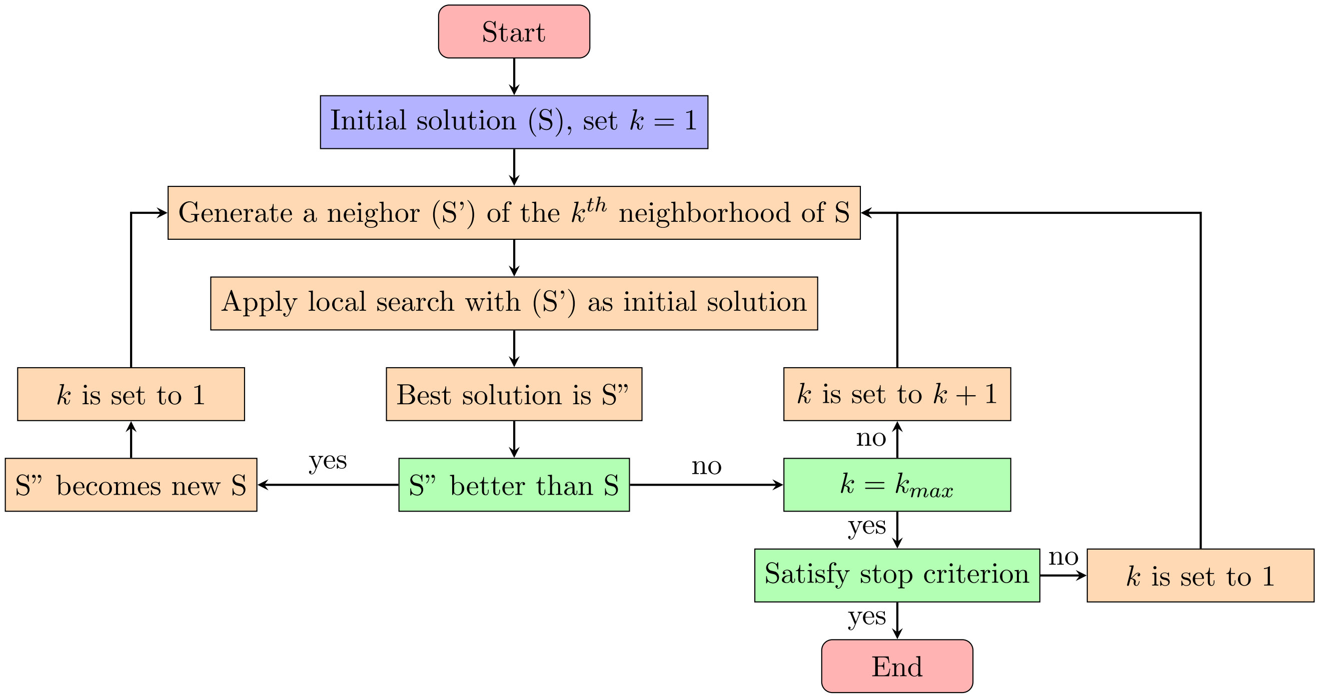


Figure 5: Flowchart of variable neighborhood search (M. A. Adibi et al., 2010).

3 Project portfolio selection and scheduling problem

The main goal of companies is to gain a high profit. The project portfolio selection and scheduling problem (PPSSP) is used to help with this as the aim of the PPSSP is to maximize the total profit by selecting and scheduling a subset of projects from a set of given projects. The PPSSP generates a schedule for the selected projects which maximizes the total profit in such a way the limiting factors permits. In this section, we formulate a mathematical model for the PPSSP. To do so, some assumptions are introduced, followed by the objective function and the constraints.

3.1 Assumptions

Before the mathematical formulation is introduced, some assumptions are discussed.

- The problem works with discrete time.
  Discrete time makes the model better programmable. Using small time increments increases the number of variables and hence the calculation time.

- A project consists of a set of tasks.
  Every project consist of several tasks and every task can have different duration, production capacity and resources required. A project is completed if and only if all individual tasks are completed.

- Precedence restrictions are defined between tasks.
  There is a predefined order on which tasks must be completed. For example, if task 3 of a project is a predecessor of task 4, task 3 must be completed before task 4 can be started.

- Idle time is allowed.
  In between different tasks there is time allowed when nothing on this project is done. It is possible to first complete task 3, wait for several time periods and then start task 4.

- No matter when a project is started, the resources required stay the same.
  Since the project stays the same, no matter when it is built, the required resources stay the same. Hence, the required resources are time independent.

- Preemption is allowed.
  When a task of a project is started it must be completed, however there can be a break in between. A task does not have to be completed in consecutive time periods. While working on a task we can take a break to build another task or maybe even another project.

- A task of a project is scheduled if the whole project can be completed.
  Only when all tasks of a project are completed, a project is fully done. Not completing every task of a project is assumed to have no value. For example, it does not make sense to work on the first tasks and nothing more.

3.2 Problem formulation

Below, the sets, parameters and variables with a short description are given.

Sets	
K	The set of resource types
N	The set of projects
S_i	The set of tasks of project $i \in N$
T	The set of time units
F_{is}	The set of direct predecessors of task $s \in S_i$ of project $i \in N$
G_{is}	The set of direct successors of task $s \in S_i$ of project $i \in N$
Parameters	
d_{is}	Duration of task $s \in S_i$ of project $i \in N$
f_i	Final task of S_i for project $i \in N$
r_{iks}	Amount of resource $k \in K$ required by task $s \in S_i$ for project $i \in N$
w_{it}	Profit when project $i \in N$ ends at period $t \in T$
m_{kt}	Amount of resource $k \in K$ available at time period $t \in T$
Variables	
X_{its}	There is worked on task $s \in S_i$ of project $i \in N$ at time period $t \in T$

Table 2: List of symbols

All possible projects are given by set N, so a selection of projects represents a subset of N. The set of tasks of a project $i \in N$ is given by set S_i. We assume there are precedence constraints and for every task $s \in S_i$ of project $i \in N$, the set of direct predecessors of task s is given by set F_{is}. Likewise, the set of direct successors of task $s \in S_i$ of project $i \in N$ is given by set G_{is}. A good schedule for a selection of projects needs to be completed in the given time span or in the given time increments in set T. The set of resources types is given by set K. Instead of only making a schedule for a given selection, the PPSSP considers the selection as well. We need a selecting and scheduling decision variable, X_{its}. The value of X_{its} is equal to one if task $s \in S_i$ of project $i \in N$ is executed at time increment $t \in T$ and zero otherwise. Every project has its own profit w_{it} which can be dependent on time, since ending a project late may result in extra costs. The opposite may also happen, when a project is finished early, the profit can be higher due to some bonuses. The possibility of finishing a project is limited by two factors. First we consider time which is equally divided in time increments and every task $s \in S_i$ of project $i \in N$ has it own completion time or duration, d_{is}. Secondly, we have resources and every task $s \in S_i$ of project $i \in N$ requires a certain amount of resource $k \in K$ which is denoted by r_{iks}. Not all the resources are available at every time period. The amount of resources of type $k \in K$ available at time increment $t \in T$ is denoted by m_{kt}. From this we can formulate the mathematical model.

3.3 Objective function

As observed, the profit should be maximized. The profit is given by the total profit of the selected projects. The last task of every project is defined to have a duration equal to one time increment and does not require any resources. This last task can only be started when every other task of a project is completed. When a project $i \in N$ is selected, all tasks $s \in S_i$ need to be finished. Without loss of generality, we assume a project is completed if the last task (f_i) of a project is completed. When we multiply w_{it} and X_{itf_i} and sum over the different time increments, we end up with the duration of the last task times the corresponding profit, $d_{if_i} \cdot P = P$ since the duration of the last task is defined to be one. There is summed over all projects to obtain the full profit. The following objective function is as in (Manish et al., 2018)

$$\max \sum_{i \in N} \sum_{t \in T} (w_{it} \cdot X_{itf_i}) \tag{1}$$

3.4 Constraints

For every project $i \in N$, if a task $s \in S_i$ is finished, the number of time increments worked on this task must be equal to the duration of this task. However, when task s is not started, there have been zero periods used. The time we work on a task of a project must be less than or equal to the duration of the task. By our definition of X_{its} we must alter the constraint of (Manish et al., 2018) to hold for preemption case.

$$\sum_{t \in T} X_{its} \leq d_{is} \qquad \forall i \in N, s \in S_i \tag{2}$$

To finish project $i \in N$, all tasks $s \in S_i$ must be completed. By the objective function we want the final task (f_i) to be completed. But then all other tasks $s \in S_i$ must be completed as well. We define this using the final task of a project. The number of time increments worked on a task $s \in S_i$ must be equal to X_{itf_i} times the duration of s.

$$\sum_{t \in T} X_{its} = \sum_{t \in T} X_{itf_i} d_{is} \qquad \forall i \in N, s \in S_i \tag{3}$$

Next, we consider the resources. The resources required by task $s \in S_i$ for project $i \in N$ are independent of time. The amount of resources used at time increment $t \in T$ must be less than or equal to the amount we have in stock (Manish et al., 2018).

$$\sum_{i \in N} \sum_{s \in S_i} r_{iks} X_{its} \leq m_{kt} \qquad \forall t \in T, k \in K \tag{4}$$

As of now, the last task of a project can be completed before the first task is started. The precedence relations are considered. For every task $s \in S_i$ of project $i \in N$, the direct predecessors are given in F_{is}. All tasks $s' \in F_{is}$ must be finished before task s is started, here we changed the constraint of (Shou & Huang, 2010) a little.

$$\sum_{t'=1}^{t-1} X_{it's'} \geq X_{its} \cdot d_{is'} \qquad \forall i \in N, s \in S_i, t \in T, s' \in F_{is} \tag{5}$$

Lastly, we define the decision variables to be binary.

$$X_{its} = \{0, 1\} \qquad \forall i \in N, s \in S_i, t \in T \tag{6}$$

3.5 Mathematical model

Combining the objective function and the constraints, the complete model is obtained.

$$\max \sum_{i \in N} \sum_{t \in T} (w_{it} \cdot x_{itf_i}) \tag{7a}$$

Subject to:

$$\sum_{t \in T} X_{its} \leq d_{is} \qquad \forall i \in N, s \in S_i \tag{7b}$$

$$\sum_{t \in T} X_{its} = \sum_{t \in T} X_{itf_i} d_{is} \qquad \forall i \in N, s \in S_i \tag{7c}$$

$$\sum_{i \in N} \sum_{s \in S_i} r_{iks} X_{its} \leq m_{kt} \qquad \forall t \in T, k \in K \tag{7d}$$

$$\sum_{t'=1}^{t-1} X_{it's'} \geq X_{its} \cdot d_{is'} \qquad \forall i \in N, s \in S_i, t \in T, s' \in F_{is} \tag{7e}$$

$$X_{its} = \{0, 1\} \qquad \forall i \in N, s \in S_i, t \in T \tag{7f}$$

Note, this model does not work for tasks with a duration of zero. If there are tasks with a duration of zero, these do not require any resources and can be neglected.

4 Solving the project portfolio selection and scheduling problem

Optimizing complex instances of the project portfolio selection and scheduling problem (PPSSP) using an integer linear programming (ILP) solver takes a lot of time. A metaheuristic is used to solve the PPSSP in a reasonable amount of time. We solve the PPSSP using an adapted variable neighborhood search (VNS). As metaheuristic within the adapted VNS, tabu search is used. Before the algorithm can start, a solution representation should be chosen.

4.1 Solution representation

The solutions are given as a combination of two lists. The first list consists of the selected projects and the second is an activity list. The activity list represents a schedule by defining the order in which the tasks may be planned. When multiple projects are selected, the activity list is defined as all tasks of all projects in order. An example for an activity list of projects a and b is given by:

$$[(Pa,1),(Pb,1),(Pb,2),(Pb,3),(Pa,3),(Pb,4),(Pb,5),(Pa,2),(Pa,4),(Pa,5)]$$

Every task is linked to their corresponding project. The activity list is made in such a way that the precedence relations are satisfied, the direct successors of a task are placed later in the activity list than the task itself. The first task of a project is defined to be the initialization of the project and requires no resources. As the first task of every projects does not require any resources it can be planned everywhere, not considering the precedence relations. The first task should, due to the precedence constraints, be scheduled before all other tasks. Combining these two observations, the first task of every project is planned before every other task, as can be seen in the example activity list above. Likewise, the last task is the concluding task of a project. It is precedence wise planned after all other tasks of the project are completed. It has a duration of zero and requires no resources, which means, it can be planned directly after all other tasks of a project are completed. To optimize the profit, the last task should be completed as soon as possible, it is planned immediately after all the other tasks are completed. The schedule which is represented by an activity list, can be made by schedule generation schemes (SGS). We discuss the serial and the parallel SGS as described by (Kim & Ellis, 2010) and (Kim, 2009).

4.1.1 Serial schedule generation scheme

The serial schedule generation scheme is described in (Kim & Ellis, 2010) for the non-preemptive case. The first task, according to the activity list, is scheduled at the first time possible, satisfying both the precedence and resource constraints. Followed by the second task being scheduled at the first time increment possible, still satisfying both the precedence and resource constraints. This process keeps repeating until all tasks are scheduled. The algorithm of finding a schedule using the serial SGS is shown in Algorithm 1. The algorithm starts with a given activity list and determines the starting times of all tasks.

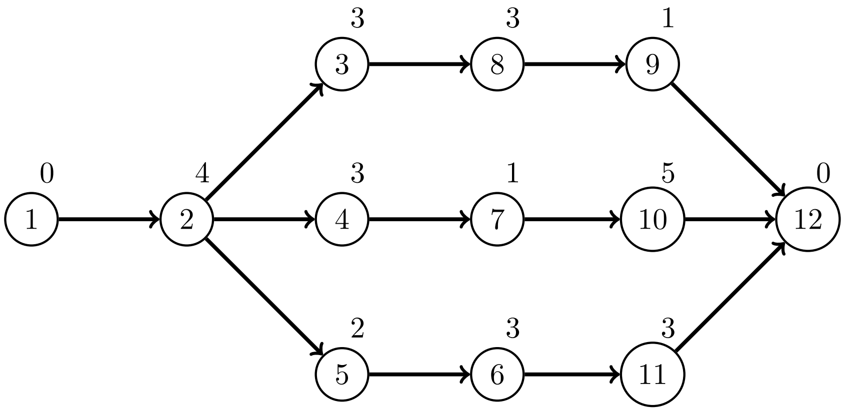
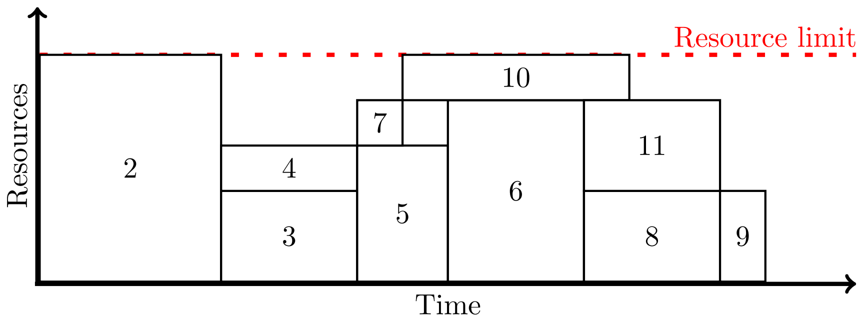


Figure 6: An example project in which every node corresponds to a task with a duration equal to the number diagonally above the node. Every arrow corresponds to a precedence constraint.

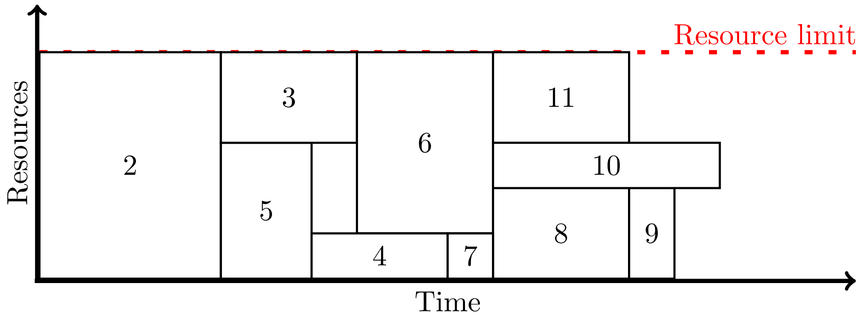
Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Resources	0	25	10	5	15	20	5	10	10	5	10	0

Table 3: Resources needed for every task of the project in Figure 6.

An obtained schedule for the project illustrated in Figure 6 is shown in Figure 7. This project is one dimensional and only limited by one type of resource. The resource requirements of every task is given in Table 3. In this case, the resource limit is taken to be twenty-five. The schedules are obtained with activity lists [1,2,3,4,5,6,7,8,9,10,11,12] (Figure 7a) and [1,2,5,3,6,4,8,7,9,11,10,12] (Figure 7b) with a time span of 16 and 15 periods, respectively. Schedules generated with a serial SGS are active schedules (Kolisch & Hartmann, 1999). An active schedule is a feasible schedule for which it is not possible to form another schedule by changing the order of planning resulting in at least one tasks completing earlier and no task getting delayed. Multiple activity lists can result in the same schedule. We could for example make activity list [1,2,4,3,5,6,7,8,9,10,11,12] and we end up with the schedule shown in Figure 7a.



(a) Activity list=[1,2,3,4,5,6,7,8,9,10,11,12]



(b) Activity list=[1,2,5,3,6,4,8,7,9,11,10,12]

Figure 7: Two activity lists and the schedule they describe using the serial SGS where every block corresponds to a task.

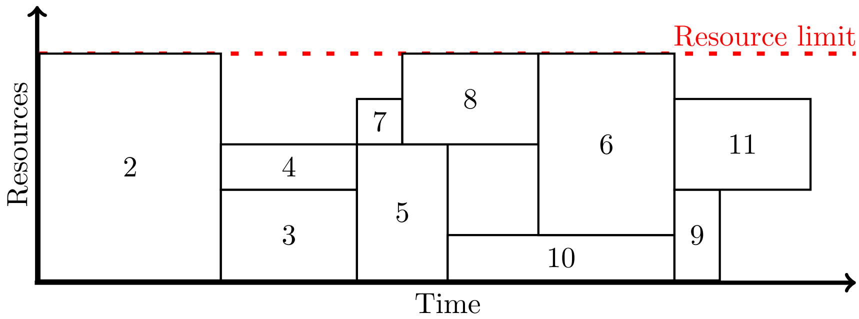
Algorithm 1: Serial schedule generation scheme

Data: A selection of projects with an activity list and resource constraints
$Activitylist \leftarrow$ given
$Tasksnotplanned \leftarrow$Activitylist
$r_{iks} \leftarrow$ Amount of resource type $k \in K$ required for task $s \in S_i$ of project $i \in N$
$m_{kt} \leftarrow$ Amount of resource type $k \in K$ available at time increment $t \in T$
$d_{is} \leftarrow$ Duration of task $s \in S_i$ of project $i \in N$
To determine:
$b_{is} \leftarrow$ Starting time of task $s \in S_i$ for project $i \in N$
while $Tasksnotplanned \neq \emptyset$ **do**
    Pick the first task (s) of $Tasksnotplanned$
    $i \leftarrow$ The project to which task s belongs
    $t' \leftarrow$ Latest time period in which the direct predecessors of s are scheduled
    while $s \in Tasksnotplanned$ **do**
        if $r_{iks} \leq m_{kt}\ \forall k \in K, t \in [t'+1, t'+d_{is}]$ **then**
            Remove s from $Tasksnotplanned$
            for $k \in K$ **do**
                for $t \in [t'+1, t'+d_{is}]$ **do**
                    $m_{kt} \leftarrow m_{kt} - r_{iks}$
                end
            end
            $b_{is} \leftarrow t'+1$
        else
            $t' \leftarrow t'+1$
        end
    end
end
Result: Starting time of every task (b_{is})

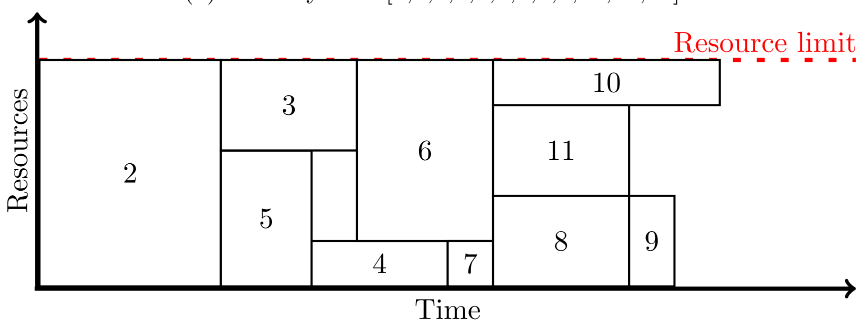
4.1.2 Parallel schedule generation scheme

The parallel schedule generation scheme is described for the non-preemptive case (Kim, 2009). The activity list represents a priority list. Instead of checking the tasks individually, the parallel SGS checks the time increments individually. At the first time increment, the tasks of which all direct predecessors are already planned are considered. The task that comes first in the activity list and satisfies both the resource constraints and precedence constrains on the domain which consists of this time increment until the time increment plus the duration, is planned. Then the second task, with respect to the activity list, is considered and planned if resources are not exceeded. If this process is done for all precedence allowed tasks, when there a no more allowed tasks or no task can be planned due to lack of resources, the next time increment is considered. Again the tasks of which all predecessor are already completed are considered and the process keeps repeating until all tasks are planned. The algorithm of finding a schedule using the parallel SGS is shown in Algorithm 2.

In Figure 8, the schedules for the same activity lists are given but now with use of the parallel schedule generation scheme. The time span for activity list [1,2,3,4,5,6,7,8,9,10,11,12] (Figure 8a) is equal to 17 periods, for [1,2,5,3,6,4,8,7,9,11,10,12] (Figure 8b), the time span is 15 periods. As shown by (Kolisch & Hartmann, 1999), all schedules generated with a parallel SGS are non-delay. A non-delay schedule is a schedule in which no resources are kept idle while a task is ready to be scheduled. Note that the schedules in Figure 7b and Figure 8b are the same schedules.



(a) Activity list=[1,2,3,4,5,6,7,8,9,10,11,12]



(b) Activity list=[1,2,5,3,6,4,8,7,9,11,10,12]

Figure 8: Two activity lists and the schedule they describe using the parallel SGS where very block corresponds to a task.

Algorithm 2: Parallel schedule generation scheme

Data: A selection of projects with an activity list and resource constraints

$Activitylist \leftarrow$ given
$Tasksplanned \leftarrow \{\}$
$Tasksnotplanned \leftarrow Activitylist$
$Taskscompleted \leftarrow \{\}$
$Tasksfeasible \leftarrow$ {Tasks of which all direct predecessor are in $Taskscompleted$}
$G_{is} \leftarrow$ {Direct successors of task $s \in S_i$ of project $i \in N$}
$F_{is} \leftarrow$ {Direct predecessors of task $s \in S_i$ of project $i \in N$}
$r_{iks} \leftarrow$ Amount of resource $k \in K$ required by task $s \in S_i$ of project $i \in N$
$m_{kt} \leftarrow$ Amount of resource $k \in K$ available at time period $t \in T$
$d_{is} \leftarrow$ Duration of task $s \in S_i$ of project $i \in N$
$t \leftarrow 0$

To determine:

$b_{is} \leftarrow$ Starting time of task $s \in S_i$ for project $i \in N$

while $Tasksnotplanned \neq \emptyset$ **do**
  Sort $Tasksfeasible$ according to $Activitylist$
  $t \leftarrow t + 1$
  for $s \in Tasksfeasible$ **do**
    $i \leftarrow$ The project to which task s belongs
    $t' \leftarrow$ Latest time period in which the direct predecessors of s are scheduled
    if $r_{iks} \leq m_{kt}\ \forall k \in K, t \in [t'+1, t'+d_{is}]$ **then**
      Remove s from $Tasksfeasible$
      Add s to $Tasksplanned$
      Remove s from $Tasksnotplanned$
      for $k \in K$ **do**
        for $t_2 \in [t'+1, t'+d_{is}]$ **do**
          $m_{kt_2} \leftarrow m_{kt_2} - r_{iks}$
        end
      end
      $b_{is} = t' + 1$
    end
  end
  for $s \in Tasksplanned$ **do**
    $i \leftarrow$ The project to which task s belongs
    if $b_{is} + d_{is} = t$ **then**
      Add s to $Taskscompleted$
      Remove s from $Tasksplanned$
      for $ds \in G_{is}$ **do**
        if $F_{ids} \cup Taskscompleted = Taskscompleted$ **then**
          Add ds to $Tasksfeasible$
        end
      end
    end
  end
end

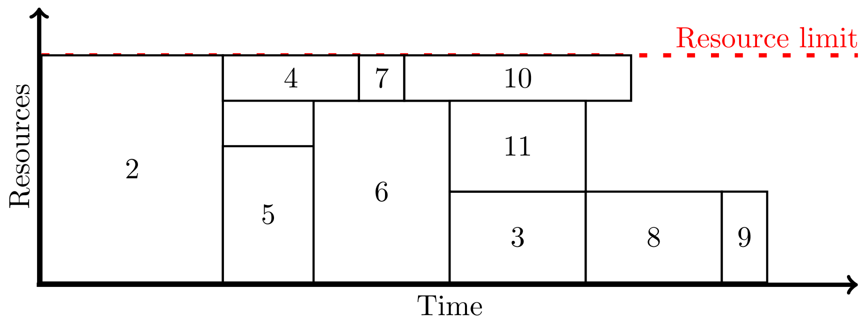
Result: Starting time of every task (b_{is})

4.1.3 Serial or parallel schedule generation scheme

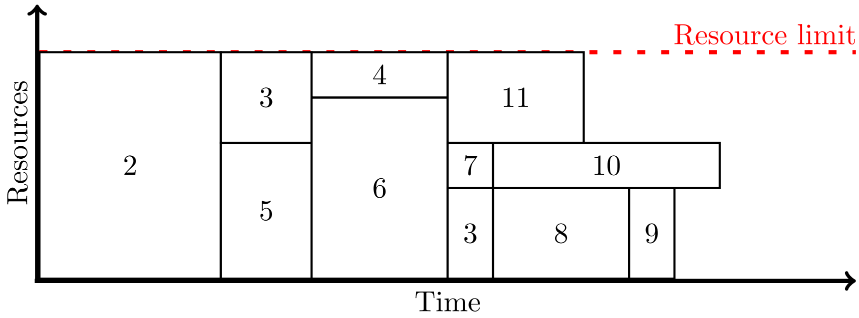
Two schedule generation schemes for the non-preemptive case are considered. A schedule made using the serial SGS is less computational-intensive than using the parallel SGS (Kim & Ellis, 2010). A schedule obtained using the serial SGS is always an active schedule. This means a feasible schedules in which it is not possible to form other schedules having at least one task completed earlier without another task getting delayed. The parallel SGS results in non-delay schedules, which is a subset of active schedules. The serial SGS can form more schedules (Kolisch & Hartmann, 1999). More specific, every active schedule can be made using the SGS, which does not hold for the parallel SGS. The optimal schedule for projects with a time dependent payment is always an active schedule. Therefore, the serial SGS can always represent the optimal schedule (Ballestín, Valls, & Quintanilla, 2008). Considering this, the serial SGS is preferred over the parallel SGS.

4.2 Preemptive serial schedule generation scheme

It is known that the serial SGS outperforms the parallel SGS in the non-preemptive case. The preemptive case is only considered for the SGS. Preemption in a serial SGS is defined as noted in (Behrouz, 2014). In this preemptive serial SGS, every task is subdivided into d_{is} disjoint parts with $s \in S_i$ for $i \in N$. Every part of a task is considered as a standalone task with the same precedence constraints as the task it originates of. In every time increment, only one of the parts is allowed to be scheduled. The direct successors of a task are only allowed to be scheduled if all part of the task have been completed. The planning procedure is the same as described in Algorithm 1. The complete algorithm of the preemptive serial SGS can be found in Algorithm 3. The schedules obtained this way may use preemption, however this does not mean all schedules obtained use preemption. Since we now allow preemption, the time span of the optimal schedule might decreases. This is shown in Figure 9 where both the schedules are made for activity list [1,2,5,6,3,4,8,9,7,10,11,12]. In Figure 9a, the serial SGS is used resulting in a time span of 16 time periods. The schedule shown in Figure 9b is made with use of the preemptive serial SGS, resulting in a time span of 15 time periods. By allowing preemption, the time span of this activity list is decreased with 1 time period. Be aware, by introducing preemption, not every active schedule can be represented by an activity list. The profit obtained using the preemptive serial SGS can never be smaller than the profit obtained using the serials SGS. Considering this, the preemptive SGS is used for solving the PPSSP.



(a) No preemption allowed



(b) Preemption allowed

Figure 9: The schedule obtained with normal and preemptive serial SGS for activity list=[1,2,5,6,3,4,8,9,7,10,11,12] where every block corresponds to either a part of task or a complete task.

Algorithm 3: Preemptive serial schedule generation scheme

Data: A selection of projects with an activity list and resource constraints
$Activitylist \leftarrow$ given
$Tasksnotplanned \leftarrow Activitylists$
$r_{iks} \leftarrow$ Amount of resource $k \in K$ needed for task $s \in S_i$ of project $i \in N$
$m_{kt} \leftarrow$ Amount of resource $k \in K$ available at time period $t \in T$
$d_{is} \leftarrow$ Duration of task $s \in S_i$ of project $i \in N$
To determine:
$b_{isl} \leftarrow$ Starting time of part $l \in [1, d_{is}]$ of task $s \in S_i$ of project $i \in N$
while $Tasksnotplanned \neq \emptyset$ **do**
    Pick the first task (s) of $Tasksnotplanned$
    $i \leftarrow$ The project to which s belongs
    $t \leftarrow$ Latest time period the direct predecessors of s are scheduled
    $l = 1$
    while $l \leq d_{is}$ **do**
        $t \leftarrow t + 1$
        if $r_{iks} \leq m_{kt}$ $\forall k \in K$ **then**
            $b_{isl} \leftarrow t$
            $l \leftarrow l + 1$
            $m_{kt} \leftarrow m_{kt} - r_{iks}$
        end
    end
    Remove s from $Tasksnotplanned$
end
Result: The time period there is worked on part l of task $s \in S_i$ of project $i \in N$ (b$_{isl}$)

4.3 Variable neighborhood search project portfolio selection and scheduling

The PPSSP can be separated into two different sub problems, one for the search of the best possible selection and one for the search of the best possible schedule for a given selection. The last sub problem can be considered as the RCPSP with the aim to maximize profit. We alter VNS in the following way. For a given selection (Sel) a random activity list (Sch) is generated. A local search algorithm is applied to Sel and the most profitable activity list is remembered. When the local search heuristic has ended, we switch to the most profitable allowed neighboring selection and again the local search heuristic is applied. The most profitable activity list is returned, the process is shown in Figure 10. Before the results can be obtained, the two neighborhood structures for the PPSSP are introduced. When the structures are defined, the initial solution is introduced, followed by the complete algorithm.

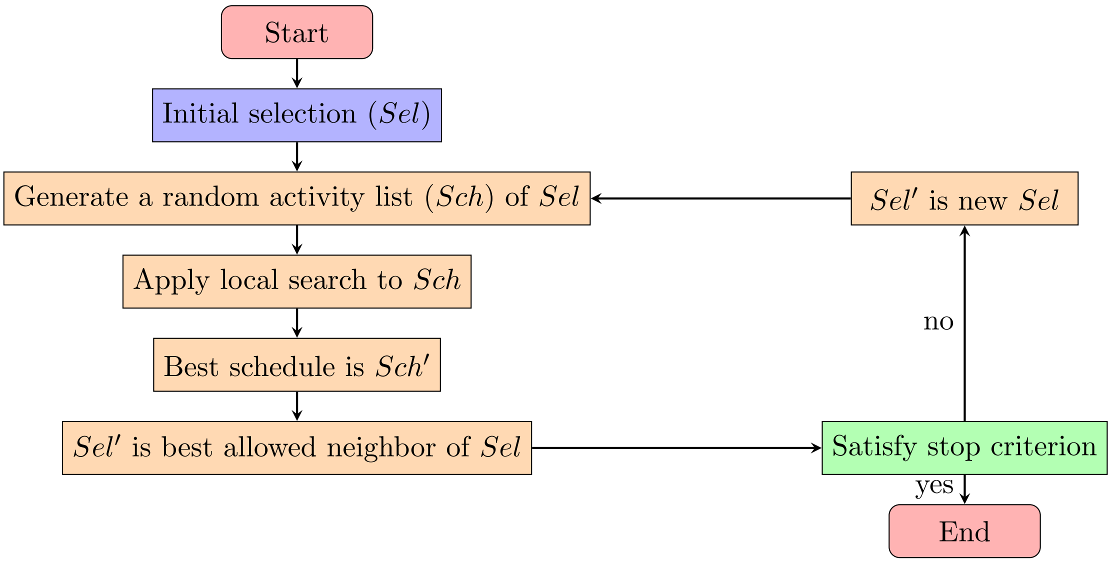


Figure 10: Flowchart of the adapted variable neighborhood search.

4.3.1 Neighborhood structure selection

The first neighborhood structure is used to get from a selection of projects to another selection. The order of the selected projects does not influence the obtained solution. Some operations are defined on the neighborhood structure:

- Adding a project
- Removing a project
- Replacing a project with another project

If we add a project, a not selected project is added to the selection, resulting in a greater possible profit. This does not necessarily mean that inserting a project results in higher profits. Some combinations of projects are not feasible in the given time span and adding a project can make a selection infeasible. On the other hand, removing a project can make the problem feasible. Replacing an already selected project by a not yet selected project, can make the problem feasible as well. Replacing a project of a feasible selection can also result in both a lower and higher profit, depending on the projects.

4.3.2 Neighborhood structure schedule

The second neighborhood structure is used to get from one activity list to another activity list. This can be interpreted as going from one schedule to another schedule. In an activity list the order does influence the schedule, therefore we should be more careful. Some operations on the neighborhood structure are defined (M. Adibi, Zandieh, & Amiri, 2010), where we have changed the name of an insert move to a reschedule move.

- Swapping two tasks
- Rescheduling a task

Both of operations start with an activity list and we discuss them in more detail.

4.3.2.1 Swapping a task

From a given activity lists (al_1), one task (s_1) is chosen. The restriction on s_1 is that it can be every task except the first and last task of every project. For the selected task s_1, the first successor and last predecessor in al_1 are determined. All the tasks in between this first successor and last predecessor are candidate swaps. For every of those candidates, it is checked whether swapping this task with s_1 results in an activity list satisfying the precedence constraints. A random valid candidate task (s_2) is chosen and swapped with s_1. An example of a valid and invalid swap of activity list [1,2,3,4,5,6,7,8,9,10,11,12] of the project in Figure 6 is shown in Figure 11. The original activity list, al_1 is given in Figure 11a. If we swap tasks 3 and 5, the precedence constraints are still satisfied, resulting in a valid swap and so the obtained activity list is a possible neighbor. On the other hand, if tasks 3 and 6 are swapped, the precedence constraints are not satisfied anymore and the obtained activity list rejected as neighbor. The algorithm shown in Algorithm 4 swaps a chosen task with a random feasible task. This can be changed in such way that the best profitable swap is taken.

1	2	3	4	5	6	7	8	9	10	11	12

(a) Original activity list

1	2	**5**	4	**3**	6	7	8	9	10	11	12

(b) Valid swap, swapping tasks 3 and 5

1	2	**6**	4	5	**3**	7	8	9	10	11	12

(c) Invalid swap, swapping tasks 3 and 6

Invalid swap

Figure 11: Swapping two tasks, one valid and one invalid swap where the differences are shown in red.

Algorithm 4: Swapping two tasks of which one given and one random

Data: Activity list and projects
$Activitylist \leftarrow$ given
$Candidateswaps \leftarrow \{\}$
$G_{is} \leftarrow \{$Direct successors of task $s \in S_i$ of project $i \in N\}$
$F_{is} \leftarrow \{$Direct predecessors of task $s \in S_i$ of project $i \in N\}$
Start
Pick task (s) out of $Activitylist$ (Not the first or last tasks of any project)
$t_1 \leftarrow$ Index of last predecessor of task s in $Actvitylist$
$t_2 \leftarrow$ Index of first successor of task s in $Actvitylist$
$swap_1 \leftarrow$Index of task s in $Actvitylist$
for $t \in [t_1 + 1, t_2 - 1] \setminus swap_1$ **do**
    Add $Activitylist[t]$ to $Candidateswaps$
end
for $ps \in Candidateswaps$ **do**
    $t_{1ps} \leftarrow$ Index of last predecessor of task ps
    $t_{2ps} \leftarrow$ Index of last successor of task ps
    if $t_{1ps} \geq swap_1$ or $t_{2ps} \leq swap_1$ **then**
        Remove ps from $Candidateswaps$
    end
end
Pick random task n from $Candidateswap$, preferably not s
$Al_2 \leftarrow Activitylist$
$swap_2 \leftarrow Index(Activitylist, n)$
$Al_2[swap1] \leftarrow Activitylist[swap2]$
$Al_2[swap2] \leftarrow Activitylist[swap1]$
Result: A neighboring activity list, Al_2

4.3.2.2 Rescheduling a task

Next the rescheduling of a task is considered. From a given activity list (al_1), one task s is chosen. This chosen task cannot be the first or last task of a project. The first successor and last predecessor are determined and task s is placed in between this interval, however not on the place it originates of. All the task between the original place and the new place have shifted one place, this are at least two tasks. In the case of two tasks, it can be defined as a swap. In Figure 12, two neighbors resulting from rescheduling are shown for activity list [1,2,3,4,5,6,7,8,9,10,11,12] and the project given in Figure 6.

1	2	3	4	5	6	7	8	9	10	11	12

(a) Original activity list

1	2	**4**	**3**	5	6	7	8	9	10	11	12

(b) Valid rescheduling, task 4 to place 3

1	2	3	4	**7**	**5**	**6**	8	9	10	11	12

(c) Valid rescheduling, task 7 to place 5

Figure 12: Two valid rescheduling moves where the differences are shown in red.

4.3.3 Initial solution

The algorithm of the adapted VNS starts with an initial solution. In case of the PPSSP, this is a selection and an activity list. A good initial solution results in a faster convergence and might even give better results than a bad initial solution. To obtain an initial solution, we first solve the ILP relaxation of Equation (7) by changing Equation (7f) to

$$X_{its} \in [0, 1]$$

By rounding the selection variables to the nearest integer (0 or 1), we obtain an good initial selection of projects.

For the given initial selection, it is possible to make an activity list satisfying the precedence constraints. Making the initial activity list is done completely arbitrary. The first step of making an arbitrary activity list is by adding all the tasks which does not have direct predecessors in a list (*Availabletasks*). One of the tasks s of *Availabletasks* is taken at random and added to the activity list. *Availabletasks* is updated by adding the direct successors of task s of which all direct predecessor are in *Availabletasks* and removing s. A new random task of *Availabletasks* is taken and the same procedure is used. This process keeps repeating until there are no more tasks to be added in the activity list, see Algorithm 5.

Algorithm 5: Get a random activity list for a given selection

```
Data: Selected projects
Activitylist ← {First task of every project}
Availabletasks ← {The tasks having no predecessors}
G_is ← {Direct successors of task s ∈ S_i of project i ∈ N}
F_is ← {Direct predecessors of task s ∈ S_i of project i ∈ N}
while Availabletasks ≠ ∅ do
    Pick random task (s) out of Availabletasks
    Add s to Activitylist
    Remove s from Availabletasks
    i ← The project to which s belongs
    for s_s ∈ G_is do
        if F_{is_s} ∪ Activitylist = Activitylist then
            Add s_s to Availabletasks
        end
    end
end
Result: An activity list satisfying the precedence constraints
```

4.3.4 Variable neighborhood search algorithm

Both the two neighborhood structures for the adapted VNS applied to PPSSP are determined. The first structure is to get from one selection to another selection. The second structure is used to maximize the profit for a given selection, in which all selected projects are scheduled, given as an activity list. For some selections, not all tasks can be planned in the original time span due to the resource limits. In our definition of the preemptive serial SGS, the process keeps going until all tasks are planned. This means there is a chance, the schedule exceeds the given time span. Since the resource limits are not defined after the original time span, the limits after the time span are taken as a constant continuation of the last defined resource limits. To compensate this, a penalty is given to every project for every time increment the original time span is exceeded. This penalty value (pen) is a subtraction from the profit. The schedules obtained this way are in the infeasible space since it does not meet all constraints. Hence, in some cases, an activity list results in an infeasible schedule. The final solution always has to be a feasible schedule. This can be done by only considering feasible schedules as real solution, in this way a feasible schedule is obtained. Be aware, it might happen that the empty selection results in the best real solution. In this case another procedure is used. This procedure is as follows, the best infeasible solution is checked more intensively by considering more neighboring activity lists. If still no feasible schedule is obtained, all combinations of selected projects are listed and their profit according to the schedule is considered. The combinations of projects can consist of only one project up till all projects of the selection and is ordered in decreasing profit. The least profitable combination is removed and the new obtained selection is checked more intensively. If this still results in an infeasible schedule, the second least profitable combination is removed of the original selection and checked by taking more neighbors. This process keeps repeating until the removal of a combination of projects results in a feasible schedule and this schedule with corresponding selection is the final solution.

The starting selection of the adapted VNS is obtained from the ILP relaxation and a randomly made activity list for this selection. The profit of this activity list is determined and the activity list is added to a tabu list ($Tabu_{al}$). Then the neighborhood structure for the schedule is entered and there is searched for r random neighbors of the given activity list. To do so, every possible swap and rescheduling is determined for the activity list and this is shuffled. In this way, the first changes which result in r different neighbors can be taken as the r neighbors. Obtaining r valid neighbors is not always possible, since in some cases an activity list has less neighbors and only these neighbors are generated. The neighbor with the highest profit which is not in $Tabu_{al}$ is taken and added to $Tabu_{al}$. If the new obtained profit is higher than the so far highest profit, both the selection and schedule are remembered. r neighbors of the best neighboring activity list are generated. This process repeats itself until in five consecutive iterations the profit has not increased. So, we stay in this neighborhood structure until there are $5 \cdot r$ activity lists where the profit did not increase. The $tabu_{al}$ has a maximum length of $maxlength_{tabu}$. The pseudo-code of this complete procedure can be found in Algorithm 6. Some of the activity lists obtained in this way do not satisfy the time span constraint as there are more time increments used. This algorithm does exactly this, but also checks whether feasible solutions have been found on the way and remembers the most profitable one and this one is returned. In this way, if there is at any iteration a feasible solution obtained, this is chosen to be better than a delayed solution. In some cases, no feasible solutions are reached.

Algorithm 6: Search best activity list for a given selection

Data: N projects with their tasks and requirements

$Activitylist \leftarrow$ Random activity list, made with Algorithm 5

$Profitsame = \leftarrow 0$

$Tabulist \leftarrow \{Activitylist\}$

$Prof \leftarrow \{\}$

$Realprofit \leftarrow$ Profit of $Activitylist$ obtained with Algorithm 3.

$Realal \leftarrow Activitylist$

$Penalty_{Realal}$

$\leftarrow pen\cdot$ Number of time increments used after given time of activity list $Realal$

if $Penalty_{Realal} = 0$ **then**

  $Feasal \leftarrow Realal$

  $Feasprofit \leftarrow Realprofit$

else

  $Feasal \leftarrow \{\}$

  $Feasprofit \leftarrow 0$

end

while $Profitsame < 5$ **do**

  $Candidate\leftarrow$ All possible swaps and reschedules with respect to $Activitylist$

  Random shuffle $Candidate$

  $Neighbors \leftarrow$ {First r different neighbors out of $Candidate$ (if possible, else less)}

  for $al \in Neighbors$ **do**

    $Plan_{al} \leftarrow$ Planning obtained with the preemptive serial SGS, algorithm 3.

    $Prof_{al} \leftarrow$ The profit of activity list al

    Add $Prof_{al}$ to $Prof$

  end

  Sort $Prof$ from large to small

  for $Prof_{al} \in Prof$ **do**

    if $al \notin Tabulist$ **then**

      $Tempprof = Prof_{al}$

      $Tempal = al$

      $Break$

    end

  end

  if $Tempprof > Realprofit$ **then**

    $Realprofit = Tempprof$

    $Realal = Tempal$

    $Profitsame = 0$

    if $Penalty_{Tempal} = 0$ **then**

      $Feasal = Tempal$

      $Feasprofit = Tempprof$

    end

  else

    $Profitsame = Profitsame + 1$

  end

  $Activitylist = Tempal$

  Add $Tempal$ to $Tabulist$ and remove the first value if $maxlength_{tabu}$ is exceeded

end

Result: The most profitable activity list ($Realal$) obtained for a given selection and the most profitable feasible activity list ($Feasal$) for the selection

If the profit did not change for the $5 \cdot r$ neighbors, we change to the selection neighborhood. For this structure, we define one tabu list which consists of the complete solutions. The neighboring selection with the highest theoretical profit allowed by the tabu list is taken as neighbor and added to the tabu list. The highest theoretical profit is the profit of a selection, when there is an unlimited amount of resources, hence every project is completed as soon as possible. The process of finding the most profitable neighboring selection allowed by the tabu list is shown in Algorithm 7. In this case there is no upper bound on the number of neighbors and all possible neighbors are considered. If a selection consist of m projects and there is a total of n projects to choose from, there can be one project removed in m different ways, one project added in $n - m$ different ways and there are $m \cdot (n - m)$ possible swaps. So there is a total of $n + (n - m) \cdot m$ possible neighbors for a selection with m projects.

As can be seen in Algorithm 7 the complete selection is added in the tabu list, instead of adding a characteristic of the solution. In first instance, the move to get from one solution to the best neighbor seems a good characteristic to add to the tabu list. In this way the inverse move cannot be done for a couple of iterations. With the inverse move is meant the opposite move, so if project a is added, the inverse move is the removal of project a. By doing so, we still end up in a cycle, which is not as wanted by the tabu search. Considering this, we chose to add the complete solution to the tabu list, since it does not impact the program significantly.

Algorithm 7: Search best neighboring selection

Data: A selection and tabu list

$Intitialselection \leftarrow$ Given

$Selection \leftarrow$ Given

$Tabu_{sel} \leftarrow$ Given

$Neighbors \leftarrow \{\}$

$Prof_{unlimited} \leftarrow \{\}$

$Prof_{max} \leftarrow 0$

$Selection_{max} \leftarrow \{\}$

$maxlength_{tabu2} \leftarrow Given$**for** $i \notin Selection$ **do**

  $Neighbor = Selection \cup \{i\}$

  Add $Neighbor$ to $Neighbors$

end

for $i \in Selection$ **do**

  $Neighbor = Selection \setminus \{i\}$

  Add $Neighbor$ to $Neighbors$

end

for $i \in Selection$ **do**

  for $j \notin Selection$ **do**

    $Neighbor = Selection \cup \{j\} \setminus \{i\}$

    Add $Neighbor$ to $Neighbors$

  end

end

for $Neigh \in Neighbors$ **do**

  if $Neigh \notin Tabu_{sel}$ **then**

    Add profit of the schedule with unlimited resources to $Prof_{unlimited}$

    $Prof_{max} \leftarrow Prof_{unlimited}$

    $Neigh_{max} \leftarrow Neigh$

    Break

  end

end

$Selection_{max}$= Selection corresponding to $Prof_{max}$

Add $Neigh_{max}$ to $Tabu_{sel}$

If $maxlength_{tabu2}$ is exceeded, the first value is removed

Result: The most theoretically profitable neighboring selection ($Selection_{max}$)

Now, a random activity list is generated for the new selection and the process of searching the best schedule with these projects is again started and the structure of scheduling is considered. If the new obtained profit is higher than the so far highest profit, both the selection and schedule are remembered. The process of selecting a neighbor for selection is terminated when the profit did not change for 50 consecutive selections or when no more neighboring selections are available. The complete adapted VNS process can be found in Algorithm 8.

Algorithm 8: Variable neighborhood search

Data: All projects, tasks and requirements

$Selection \leftarrow$ A randomly selected combination of projects

$Tabu_{sel} \leftarrow \{\}$

$Activitylist_{best} \leftarrow \{\}$

$Profit_{best} \leftarrow 0$

$Nochangeprofit \leftarrow 0$

while $Nochangeprofit < 50$ **do**

    if $ILP_{relax}(Selection) \leftarrow feasible$ **then**

        $Nochangeprofit \leftarrow Nochangeprofit + 1$

        Start Algorithm 6 and obtain $Realal$ for $Selection$

        $ProfReal$ is fitness of schedule represented with $Realal$, made with Algorithm 3.

        if $ProfReal > Profit_{best}$ **then**

            $Nochangeprofit \leftarrow 0$

            $Profit_{best} \leftarrow ProfReal$

            $Activitylist_{best} \leftarrow Realal$

        end

    end

    Start Algorithm 7 and obtain a new selection, $Selection$ and the updated $Tabu_{sel}$

    if $Selection = \emptyset$ **then**

        $Break$

    end

end

Result: The most profitable activity list ($Activitylist_{best}$) obtained for the problem

Before finding the best activity list is started, the solution for the ILP relaxation is determined. The ILP relaxation is almost the same model as given in Equation (7) in Section 3. The model is changed in such a way that it gives a schedule for a given selection, hence the project selection is already fixed. Only for the projects in this selection, variables are made. Since only a planning should be made, Equation (7b) is changed to an equality. As of now, the model still describes an ILP, but can be transformed to an ILP relaxation if every variable is allowed to take every value in between 0 and 1 instead of being either 0 or 1 (Equation (7f)). The problem has become significantly easier and if the ILP relaxation cannot form a feasible schedule, the problem with variables of value 0 or 1 can neither form a feasible schedule. This is used in such a way that selections, in which the ILP relaxation results in infeasibility, are not planned and a new neighbor is searched.

As of now, all projects need to be completed within the given time span. This is mostly done by making the penalty value arbitrarily large. By doing this, it is more profitable to not selected the delayed project. Depending on the value of the penalty, *pen*, the problem could be altered in such a way that a fine is given for every time increment a project is delayed. This could give some interesting results, in some cases it might be better to know in advance the company gets a fine, but the overall profit is bigger.

5 Results

The project portfolio selection and scheduling problem (PPSSP) is introduced in Section 3 and in Section 4 the way of solving PPSSP is explained. For given data, the PPSSP is solved using Python. The results are obtained on a notebook with 8gb ram, Windows 10 Home, x64-processor, Intel Core i5-7200 CPU 2.50GHz and Python 3.7 (64-bit). The test data consists of a hundred randomly generated projects shown in Appendix A. Every project consists of twelve tasks in which task one and twelve have a duration or zero and require no resources. Every project is made out of maximal four different types of resources. The profit is taken to be time independent, hence no matter when a project is finished, the profit stays the same. Two different sizes on the set of projects are considered, a smaller case with a set of five projects and a bigger case with a set of ten projects. The instances and profits are given in Appendix B and C for the small and big instances, respectively. In this section, the solution is shown as only a selection and profit, however, the solution consists of an activity list as well.

5.1 ILP

The integer linear programming (ILP) given in Equation (7) is solved in Python with help of the ILP solver, Gurobi. For every problem, there are either five or ten possible projects given. In the smaller case, we let Gurobi run for maximal thirty minutes and the best solution obtained until this moment is saved. For the smaller case, the resource limit, time span and profits can be found in Appendix B. The results obtained are shown in Table 4. The first column lists the projects that can be selected. The second and third column combined form the best solution at the moment the ILP was stopped, the fourth column gives the upper bound, the obtained objective function value should always be lower or equal to this value. The fifth column is the gap between the obtained profit and the upper bound with respect to the profit. The last column shows the time when the ILP solver was stopped. This can either be the time at which the optimum value is obtained or when thirty minutes have expired. In some cases, the ILP is solved to optimality in a short period of time.

Projects	**Selection**	**Obj. val.**	**Up bound**	**Gap**	**Time**
9,19,21,26,68	9,19,68	14523	15081	3.84%	1800
21,28,32,52,82	21,28,32	13895	18691	34.52%	1800
17,23,32,80,90	32,80,90	14248	14248	0.00%	41
29,31,37,49,92	29,37,49	11525	14860	28.94%	1800
2,38,71,74,86	38,74,86	9492	9492	0.00%	0
39,58,60,79,80	58,60,79,80	15484	15484	0.00%	76
36,52,59,89,94	36,89,94	15200	15660	3.03%	1800
21,38,48,49,57	21,48,49	12210	15134	23.95%	1800
5,12,71,81,93	5,81,93	13476	17800	32.09%	1800
24,40,41,73,91	41,73,91	15474	15474	0.00%	23

Table 4: The results of the ILP for the small instances while running for maximal thirty minutes.

The same table can be made for the bigger instances. The maximal calculation time of Gurobi should be greater and is taken to be one hour. The results are shown in Table 5 with the same structure as the previous table. The column with time is removed since for each instance the time limit is reached before optimality is obtained.

Projects	Selection	Obj. val.	Up bound	Gap
2,3,33,46,53,63,70,72,86,96	2,3,46,53,63,70,72,96	32180	35917	11.61%
10,15,52,53,58,63,68,76,93,98	10,15,52,53,58,68,76,93	33022	37064	12.24%
7,12,20,23,35,50,52,58,73,87	20,35,50,52,58,73,87	31504	39290	24.71%
5,16,20,37,40,44,61,79,85,99	5,16,20,40,61,79,99	28482	33569	17.86%
4,19,27,42,45,46,59,63,72,75	4,27,42,45,46,63,72	32427	37345	15.17%
2,7,22,23,42,51,58,68,79,100	2,7,22,42,51,58,79,100	31878	37053	16.23%
4,15,17,18,53,61,66,74,79,80	4,15,18,61,66,74,80	30190	34104	12.94%
6,22,24,53,66,72,73,82,83,96	6,22,24,66,72,82,83,96	31127	34783	11.75%
0,14,25,31,36,57,61,72,83,98	14,25,36,57,61,83,98	31660	37095	17.17%
2,7,26,30,35,57,75,78,86,97	2,26,35,57,78,86,97	28909	31483	8.90%

Table 5: The results of the ILP for the big instances while running for one hour.

5.2 Variable neighborhood search

The ILP is solved for both the small and big instances. The obtained solutions of the small instances can be used to test the performance of the variable neighborhood search on the smaller instances. The obtained information about the adapted VNS performance is used to apply the adapted VNS to the big instances. First, the results of adapted VNS for the small instances are obtained, followed by the results for the big instances.

5.2.1 Small instances

Solving the small instances is done in two different ways, based on the number of neighbors encountered. In the first method, there is searched for fifty different neighbors of an activity list, while in the second method is searched for a hundred different neighbors. Using a hundred neighbors gives more possibilities of obtaining a higher profit, however it takes more time to consider them all. Therefore, the second method should return better results and take more time. The neighbors considered are randomly generated. To obtain a good estimate of the performance of the adapted VNS, both methods are completed five times and the average is taken. The obtained profits and calculation time for both the methods are shown in Table 6, both rounded to integers. The second method gives better results in 80% of the cases and outperforms the first method. As noted, the time elapsed to compute the solution has increased. Taking more neighbors does not always result in better results. However, in general we can conclude, taking more neighbors results in better results and an increase of computation time. The big cases take more time and only the second method with a hundred neighbors is considered.

The profit obtained with adapted VNS with a hundred neighbors results in 40% of the cases in a equal or higher profit compared to the ILP. However, the time elapse solving the ILP to optimality takes more time than the adapted VNS. In some cases, the ILP is solved to optimality in a short period of time and the adapted VNS results in worse results. These results are good cases to evaluate the performance of the adapted VNS. It turns out, the adapted VNS performs similar to the solved ILP while taking less time.

	50 neighbors		100 neighbors		ILP		
Projects	**Obj. val.**	**Time**	**Obj. val.**	**Time**	**Obj. val**	**Up bound**	**Time**
9,19,21,26,68	14003	70	14331	104	14523	15081	1800
21,28,32,52,82	14559	141	14559	240	13895	18691	1800
17,23,32,80,90	13688	54	13786	83	14248	14248	41
29,31,37,49,92	11517	99	10894	202	11525	14860	1800
2,38,71,74,86	9492	33	9492	46	9492	9492	0
39,58,60,79,80	12492	57	12349	98	15484	15181	76
36,52,59,89,94	13092	140	13602	223	15200	15660	1800
21,38,48,49,57	12242	99	12242	172	12210	15134	1800
5,12,71,81,93	13434	78	13476	135	13476	17800	1800
24,40,41,73,91	13616	100	13399	199	15474	15474	23

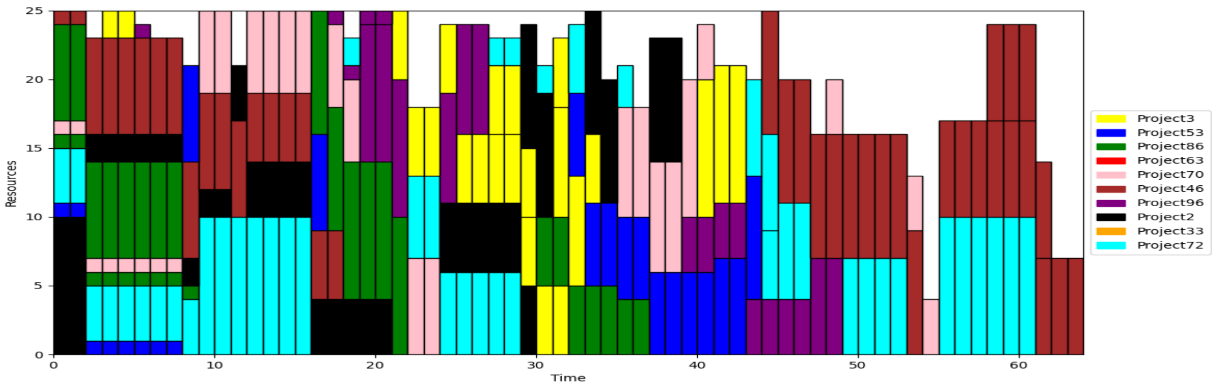
Table 6: The average results over five computations of the adapted VNS for the smaller instances compared with the ILP which has a maximal time of thirty minutes.

5.2.2 Big instances

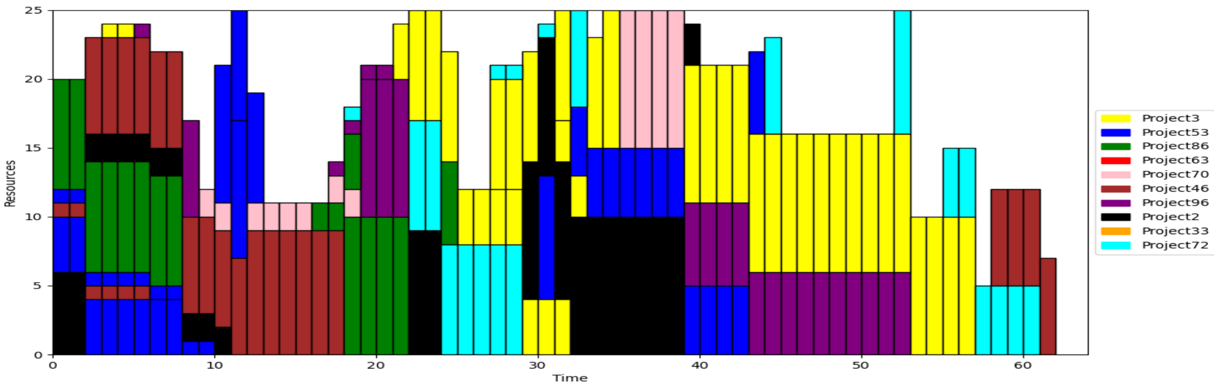
The big instances take significantly more time. Only the method where is searched for a hundred different neighbors is done, as this method performs better. The results obtained for one use of the adapted VNS is shown in Table 7. In this table, the upper bound of every instance is shown as well. The upper bound is obtained using the ILP and Gurobi as shown in Table 5. The calculation time has increased a lot due to the complexity. The adapted VNS results in 80% of the cases in a higher profit. However, the computation time of the adapted VNS is in several cases twice as long as the maximal computation of solving the ILP. Solving the ILP to to optimality could take even more time. As a test, the first big instances has been running for more than three hours and even then, no optimal solutions was obtained. We can conclude that the performance of the adapted VNS is good. An example of the resource usage is shown in Figure 13, this usage is for the first big instance.

	100 neighbors			ILP	
Projects	**Selection**	**Obj. val.**	**Time**	**Obj. val.**	**Up bound**
2,3,33,46,53,63,70,72,86,96	2,3,46,53,70,72,86,96	32348	2627	32180	35917
10,15,52,53,58,63,68,76,93,98	10,58,63,68,76,93,98	31331	9045	33022	37064
7,12,20,23,35,50,52,58,73,87	7,20,23,35,50,52,58,73	34142	7947	32504	39290
5,16,20,37,40,44,61,79,85,99	20,37,40,61,79,85,99	29219	11877	28482	33569
4,19,27,42,45,46,59,63,72,75	19,27,4,45,46,63,72	32922	8613	32427	37345
2,7,22,23,42,51,58,68,79,100	2,7,23,42,51,58,68,79	32807	4613	31878	37053
4,15,17,18,53,61,66,74,79,80	4,15,53,61,66,74,79,80	31289	5251	30190	34104
6,22,24,53,66,72,73,82,83,96	6,22,24,53,66,72,73,83	30383	7099	31127	34783
0,14,25,31,36,57,61,72,83,98	0,14,25,36,57,61,72	32045	7852	31660	37095
2,7,26,30,35,57,75,78,86,97	2,7,35,57,78,86,97	28431	9367	28909	31483

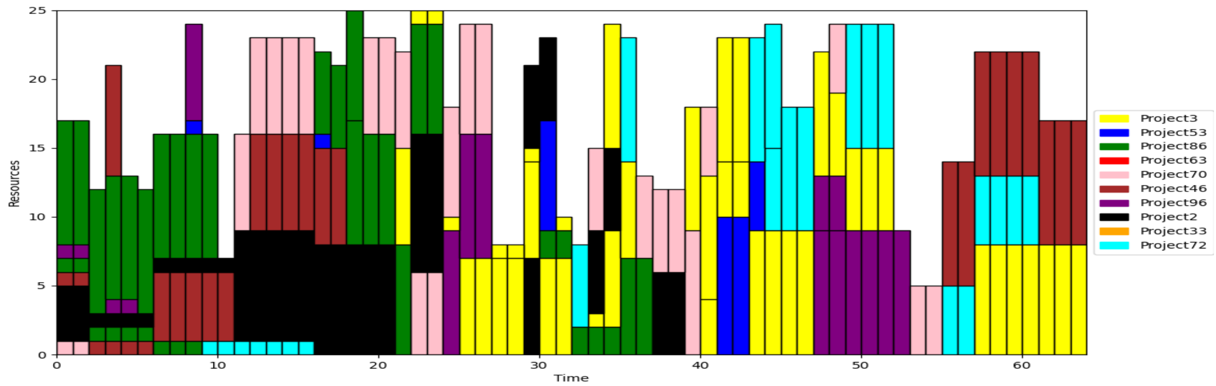
Table 7: The results of the adapted VNS for the bigger instances compared with the ILP which has a maximal time of one hour.



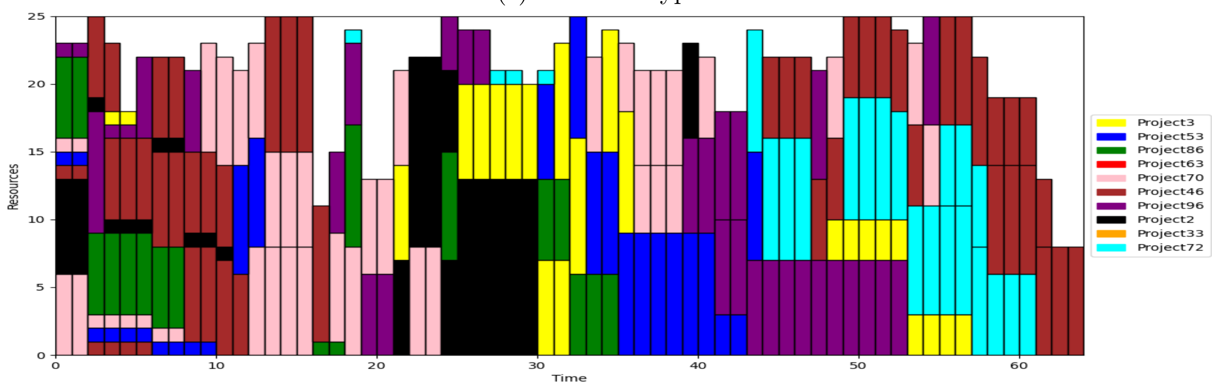
(a) Resource type 1



(b) Resource type 2



(c) Resource type 3



(d) Resource type 4

Figure 13: The resources usage in the given time span according to the adapted VNS procedure for the first big instance.

6 Conclusion and recommendations

The performance of the adapted variable neighborhood search (VNS) on the project portfolio selection and scheduling problem (PPSSP) is determined by comparing the adapted VNS to solving the integer linear programming (ILP). Some small instances of the PPSSP can be solved using an ILP solver within a reasonable amount time. As the problem becomes more complicated, solving the ILP takes more time. Solving the PPSSP using a heuristic takes less time. First, the adapted VNS is tested on small instances. Solving the PPSSP with the adapted VNS is done with two different numbers of neighbors. The method with more neighbors results in 80% of the cases in a higher profit, however the computation time increases. Nevertheless, the method where a hundred neighbors are considered is chosen to be the better method. The obtained results are compared with the solutions of solving the ILP. In 40% of the cases, the adapted VNS results in profits higher or equal than solving the ILP for thirty minutes. The computation time of the adapted VNS is smaller than the time needed to solve the ILP. In some cases, the ILP can be solved to optimality in a short time period and in all cases the adapted VNS returns a good estimate.

The adapted VNS is an efficient method for maximising profit for small instances of the PPSSP. Now, the bigger instances are considered. As is turned out that taking more neighbor results in better solutions, big instances are only considered with a hundred neighbors. In big cases, the ILP cannot be solved in a reasonable amount of time. The upper bounds are compared with the results of the adapted VNS. The values obtained with the adapted VNS are good estimates for the real maximum. The profits are in 80% of the cases higher than the solutions of solving the ILP for one hour. The adapted VNS is no time limit given, while solving the ILP has a time limit of one hour. In several cases, the adapted VNS took more than twice the time limit of solving the ILP. Nevertheless, the adapted VNS with a hundred neighbors is a good metaheuristic used for solving complex instances of the PPSSP. Solving the ILP suffices for small instances.

As is turns out, the adatped VNS is a efficient method for maximizing profit for both small and big instances of the PPSSP. However, in some cases, the ILP can be solved to optimality in a reasonable computation time and the ILP is preferred.

The performance of the adapted VNS could be improved by considering more neighboring solutions. As there are more neighbors considered, the chance of obtaining a better estimate increases. This does however, increase the calculation time. In a case with ten project this would not be a problem and more neighbors can be taken. When the instance become more complex, the calculation time could increase significantly. However, in normal instances, the calculation time of the adapted VNS turns out to be smaller than the calculation time of solving the ILP to optimally. Using a way faster computer, the number of neighbors considered might be increased resulting in much better results while the computation time does not increase significantly.

The profit could also be increased by changing the stop criterion. We used two different stop criteria. The stop criterion for the schedule is a non changing profit in five consecutive iterations. By changing this to a value greater than five, more neighbors are considered and the profit could increase. The second stop criterion is used for the selection. Increasing this stop criterion could also increase the profit. Both methods result in a longer computation time.

Instead of only using tabu search as local search algorithm of the adapted VNS, a combination of multiple metaheuristics could be used. This could result in higher profits, but this is not certain. Doing this could also result in lower profits. However, it should be considered.

Bibliography

Abdeslam, A., El Bouanani, F., & Ben-azza, H. (2014, 06). Four parallel decoding schemas of product block codes. *Transactions on Networks and Communications*, *2*, 49-69. doi: 10.14738/tnc.23.229

Adibi, M., Zandieh, M., & Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications*, *37*(1), 282 - 287. Retrieved from http://www.sciencedirect.com/science/article/pii/S0957417409004199 doi: https://doi.org/10.1016/j.eswa.2009.05.001

Adibi, M. A., Zandieh, M., & Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications*, *37*(1), 282 - 287. Retrieved from http://www.sciencedirect.com/science/article/pii/S0957417409004199 doi: https://doi.org/10.1016/j.eswa.2009.05.001

Ballestín, F., Valls, V., & Quintanilla, S. (2008). Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, *189*(3), 1136 - 1152. Retrieved from http://www.sciencedirect.com/science/article/pii/S0377221707005905 doi: https://doi.org/10.1016/j.ejor.2006.07.052

Behrouz, A. (2014). Resource constrained project scheduling subject to due dates: Preemption permitted with penalty. *Advances in Operations Research*. Retrieved from https://doi.org/10.1155/2014/505716 doi: 10.1155/2014/505716

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999, jan). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, *112*(1), 3–41. Retrieved from https://doi.org/10.1016%2Fs0377-2217%2898%2900204-5 doi: 10.1016/s0377-2217(98)00204-5

Carazo, A., Gómez, T., Molina, J., Hernández-Díaz, A., M.Guerrero, F., & Caballero, R. (2010, apr). Solving acomprehensivemodelformultiobjectiveprojectportfolioselection. *Computers OperationsResearch*, *37*(4), 630–639. Retrieved from https://doi.org/10.1016/j.cor.2009.06.012

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling, a research handbook*. Dordrecht: Kluwer Academic Publishers.

Ghasemzadeh, F., Archer, N., & Iyogun, P. (1999, jul). A zero-one model for project portfolio selection and scheduling. *Journal of the Operational Research Society*, *50*(7), 745–755. Retrieved from https://doi.org/10.1057/palgrave.jors.2600767 doi: 10.1057/palgrave.jors.2600767

Glover, F. (1990, aug). Tabu search: A tutorial. *Interfaces*, *20*(4), 74–94. Retrieved from https://doi.org/10.1287%2Finte.20.4.74 doi: 10.1287/inte.20.4.74

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, C., & Denk, M. (2009, jan). Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research*. Retrieved from https://doi.org/10.1007%2Fs10100-008-0081-z doi: 10.1007/s10100-008-0081-z

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, *130*(3), 449 - 467. Retrieved from http://www.sciencedirect.com/science/article/pii/S0377221700001004 doi: https://doi.org/10.1016/S0377-2217(00)00100-4

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, *45*(7), 733-750. doi: 10.1002/(SICI)1520-6750(199810)45:7⟨733::AID-NAV5⟩3.0.CO;2-C

Holland, J. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–73. Retrieved from http://www.jstor.org/stable/24939139

Kim, J.-L. (2009, dec). Proposed methodology for comparing schedule generation schemes in construction resource scheduling. In *Proceedings of the 2009 winter simulation conference (WSC).* IEEE. Retrieved from https://doi.org/10.1109%2Fwsc.2009.5429252 doi: 10.1109/wsc.2009.5429252

Kim, J.-L., & Ellis, D., R. (2010). Comparing schedule generation schemes in resource-constrained project scheduling using elitist genetic algorithm. *Journal of Construction Engineering and Management*, *136*(2), 160-169. Retrieved from https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-9364%282010%29136%3A2%28160%29 doi: 10.1061/(ASCE)0733-9364(2010)136:2(160)

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. *J.Weglarz (ed.), Project scheduling: Recent models, algorithms and applications*, 147-178. doi: 10.1007/978-1-4615-5533-97

Liu, S.-S., & Wang, C.-J. (2008, nov). Resource-constrained construction project scheduling model for profit maximization considering cash flow. *Automation in Construction*, *17*(8), 966–974. Retrieved from https://doi.org/10.1016%2Fj.autcon.2008.04.006 doi: 10.1016/j.autcon.2008.04.006

Lova, A., & Tormos, P. (2001). Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. *Annals of Operations Research*, *102*, 263-286. doi: https://doi.org/10.1023/A:1010966401888

Manish, K., Mittal, M., Gunjan, S., & Dheeraj, J. (2018). A hybrid tlbo-ts algorithm for integrated selection and scheduling of projects. *Computers Industrial Engineering*, *119*, 121 - 130. Retrieved from http://www.sciencedirect.com/science/article/pii/S0360835218301141 doi: https://doi.org/10.1016/j.cie.2018.03.029

Moumene, K., & Ferland, A., J. (2009). Activity list representation for a generalization of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *199*(1), 46 - 54. Retrieved from http://www.sciencedirect.com/science/article/pii/S0377221708009673 doi: https://doi.org/10.1016/j.ejor.2008.10.030

Sawant, V. C. (2016, jun). Genetic algorithm for resource constrained project scheduling. *International Journal of Science and Research (IJSR)*, *5*(6), 139–146. Retrieved from https://doi.org/10.21275%2Fv5i6.nov164087 doi: 10.21275/v5i6.nov164087

Shorman, M., S, & Pitchay, A., S. (2015, feb). Significance of parameters in genetic algorithm, the strengths,its limitations and challenges in image recovery. *ARPN Journal of Engineering and Applied Sciences*, *10*(2).

Shou, Y.-y., & Huang, Y.-l. (2010). Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value. *Journal of Zhejiang University SCIENCE C*, *11*, 562–574. Retrieved from https://doi.org/10.1631/jzus.C0910479 doi: 10.1631/jzus.C0910479

Talbot, F. B. (1982, oct). Resource-constrained project scheduling with time-resource trade-offs: The nonpreemptive case. *Management Science*, *28*(10), 1197–1210. Retrieved from https://doi.org/10.1287%2Fmnsc.28.10.1197 doi: 10.1287/mnsc.28.10.1197

Thomas, P., & Salhi, S. (1998). A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics*, *4*, 123–139. Retrieved from https://doi.org/10.1023/A:1009673512884

Tsai, Y., & Gemmill, D. D. (1998, nov). Using tabu search to schedule activities of stochastic resource-constrained projects. *European Journal of Operational Research*, *111*(1), 129–141. Retrieved from https://doi.org/10.1016%2Fs0377-2217%2897%2900311-1 doi: 10.1016/s0377-2217(97)00311-1

Wall, M. B. (1996). *A genetic algorithm for resource-constrained scheduling.* Massachesettts Institute of Technology.

Wang, Z., Wu, G., Boriboonsomsin, K., & Barth, M. (2017, 10). Intra-platoon vehicle sequence optimization for eco-cooperative adaptive cruise control.. doi: 10.1109/ITSC.2017.8317879

Whitley, D. (1994, jun). A genetic algorithm tutorial. *Statistics and Computing*, *4*(2). Retrieved from https://doi.org/10.1007%2Fbf00175354 doi: 10.1007/bf00175354

Appendices

A Projects

The project are given as an adapted Patterson format. For a task, given in the first column, the duration is denoted in the second column. The resources needed per resource type are given in the third until sixth columns followed by the direct successors in the last four columns. A zero as successors means that there are no more direct successors.

Project 1		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	6
2	5	2	0	0	2	8	4	0	0
3	8	6	0	1	10	4	0	0	0
4	8	0	3	9	6	9	7	0	0
5	8	0	6	0	0	9	8	0	0
6	10	0	0	9	10	11	9	0	0
7	7	10	10	4	6	11	10	0	0
8	7	0	5	10	5	11	10	0	0
9	5	0	6	4	0	10	0	0	0
10	1	0	0	5	4	12	0	0	0
11	5	0	0	0	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 2		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	9	2	2	1	1	9	4	0	0
3	2	0	9	10	7	11	9	8	7
4	10	4	0	8	0	8	6	0	0
5	2	10	6	4	7	11	8	7	0
6	8	0	0	0	7	11	7	0	0
7	1	0	0	7	7	10	0	0	0
8	6	9	0	6	0	10	0	0	0
9	6	5	0	0	6	10	0	0	0
10	1	0	3	0	7	12	0	0	0
11	10	0	10	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 3		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	2	2	1	0	1	8	4	3	0
3	8	5	8	1	0	9	7	6	5
4	2	8	3	0	10	5	0	0	0
5	2	0	0	7	9	11	10	0	0
6	3	10	0	4	0	11	10	0	0
7	10	0	10	9	0	11	10	0	0
8	8	5	4	7	7	11	10	0	0
9	9	0	10	0	3	10	0	0	0
10	7	0	0	8	0	12	0	0	0
11	4	0	0	6	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 4		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	2	1	1	1	0	5	3	0	0
3	8	0	7	6	1	11	8	7	6
4	9	4	0	6	0	8	6	5	0
5	2	6	7	0	10	10	7	0	0
6	6	9	7	8	7	10	9	0	0
7	7	0	8	0	7	9	0	0	0
8	10	8	7	0	0	9	0	0	0
9	6	0	0	9	5	12	0	0	0
10	1	8	0	0	0	12	0	0	0
11	9	0	5	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 5		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	5	0	0
2	9	0	0	1	0	6	4	3	0
3	4	3	0	6	0	9	8	7	0
4	8	6	1	0	1	8	7	0	0
5	3	4	0	8	5	11	8	0	0
6	9	0	9	0	6	7	0	0	0
7	1	10	0	3	3	11	10	0	0
8	6	0	8	10	0	10	0	0	0
9	8	0	0	8	9	10	0	0	0
10	10	5	0	0	10	12	0	0	0
11	1	8	6	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 6		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	9	0	1	3	0	8	3	0	0
3	6	1	0	4	1	11	7	4	0
4	2	0	0	5	5	5	0	0	0
5	3	0	4	0	4	6	0	0	0
6	1	0	0	10	9	10	9	0	0
7	1	0	8	0	0	9	0	0	0
8	6	4	10	7	5	9	0	0	0
9	4	10	0	8	9	12	0	0	0
10	1	9	7	5	0	12	0	0	0
11	2	0	0	0	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 7		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	5	0	0
2	6	0	1	0	0	9	4	3	0
3	9	1	10	1	0	6	0	0	0
4	4	7	0	6	1	6	0	0	0
5	5	4	0	0	0	7	0	0	0
6	4	10	7	5	10	11	8	0	0
7	10	0	5	0	0	11	8	0	0
8	3	4	7	10	0	10	0	0	0
9	5	0	8	9	5	10	0	0	0
10	7	0	0	5	0	12	0	0	0
11	8	10	4	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 8		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	5	0	1	0	1	4	3	0	0
3	2	0	2	0	0	11	7	6	5
4	10	0	6	1	6	10	7	0	0
5	9	0	0	0	0	10	9	0	0
6	10	0	9	8	5	10	9	0	0
7	2	0	7	10	6	8	0	0	0
8	2	0	10	6	8	9	0	0	0
9	5	1	6	0	10	12	0	0	0
10	7	10	7	5	0	12	0	0	0
11	2	7	6	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 9		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	6
2	2	3	1	1	1	4	0	0	0
3	10	5	5	7	7	4	0	0	0
4	1	0	0	8	0	11	8	7	0
5	7	0	7	8	4	11	10	8	0
6	10	0	6	0	0	10	8	0	0
7	9	0	0	0	7	10	9	0	0
8	6	10	9	0	10	9	0	0	0
9	10	0	8	0	8	12	0	0	0
10	9	0	8	0	5	12	0	0	0
11	10	6	4	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 10		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	6	0
2	8	1	1	1	1	9	5	4	0
3	8	0	0	9	8	9	8	4	0
4	8	0	5	0	0	7	0	0	0
5	8	10	0	4	0	8	0	0	0
6	4	0	0	9	0	7	0	0	0
7	9	9	7	7	0	11	10	0	0
8	6	0	5	0	5	11	10	0	0
9	2	7	0	0	9	10	0	0	0
10	5	6	9	0	7	12	0	0	0
11	5	3	9	0	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 11		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	5
2	9	0	1	1	0	11	6	0	0
3	2	0	2	0	1	11	6	0	0
4	8	0	9	3	7	11	10	7	0
5	8	0	10	0	5	11	10	9	8
6	7	1	4	10	0	10	7	0	0
7	9	0	0	4	9	9	8	0	0
8	3	0	6	8	0	12	0	0	0
9	5	9	0	0	6	12	0	0	0
10	6	0	6	10	0	12	0	0	0
11	3	8	10	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 12		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	8
2	1	0	1	0	0	4	0	0	0
3	2	1	6	0	2	4	0	0	0
4	4	8	0	0	3	11	6	0	0
5	3	0	0	1	8	11	6	0	0
6	10	7	8	0	10	9	7	0	0
7	2	0	8	6	0	10	0	0	0
8	4	6	7	0	0	9	0	0	0
9	9	9	6	8	7	12	0	0	0
10	6	5	0	8	0	12	0	0	0
11	9	6	0	7	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 13		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	2	0	2	0	0	9	6	5	4
3	10	1	7	1	1	9	5	4	0
4	9	0	8	3	10	10	8	0	0
5	5	7	0	7	0	11	7	0	0
6	8	0	0	0	4	11	7	0	0
7	4	0	0	10	0	10	0	0	0
8	5	0	0	5	8	11	0	0	0
9	1	10	10	8	5	10	0	0	0
10	2	0	3	7	6	12	0	0	0
11	9	0	0	7	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 14		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	8	1	2	0	3	5	3	0	0
3	6	10	0	1	0	11	10	7	6
4	10	10	6	2	3	11	10	7	6
5	2	0	7	8	8	10	6	0	0
6	10	8	6	10	0	9	8	0	0
7	3	4	0	0	0	8	0	0	0
8	7	3	10	6	0	12	0	0	0
9	6	0	5	10	10	12	0	0	0
10	3	0	0	0	0	12	0	0	0
11	7	0	0	5	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 15		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	6
2	2	1	1	0	1	4	0	0	0
3	3	0	7	1	0	4	0	0	0
4	9	10	0	8	8	11	10	7	0
5	7	6	0	7	8	11	10	7	0
6	7	8	6	4	0	7	0	0	0
7	10	0	0	0	0	9	8	0	0
8	5	5	7	0	0	12	0	0	0
9	1	4	9	10	8	12	0	0	0
10	8	8	0	0	5	12	0	0	0
11	10	0	0	0	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 16		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	9	0
2	8	2	1	0	0	6	4	0	0
3	8	0	5	0	0	5	4	0	0
4	2	0	5	0	1	11	7	0	0
5	2	0	5	1	0	11	7	0	0
6	8	0	10	9	8	7	0	0	0
7	9	8	0	8	9	8	0	0	0
8	3	6	7	0	0	10	0	0	0
9	7	4	8	4	4	11	0	0	0
10	4	0	0	8	0	12	0	0	0
11	8	10	7	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 17		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	2	0	0	0	4	9	6	4	3
3	8	1	1	1	3	8	5	0	0
4	10	5	6	0	0	11	8	7	0
5	1	8	0	0	10	11	7	0	0
6	6	7	9	0	0	8	7	0	0
7	1	9	0	10	0	10	0	0	0
8	3	7	0	0	10	10	0	0	0
9	8	6	7	0	0	11	0	0	0
10	7	5	7	6	6	12	0	0	0
11	2	0	0	7	3	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 18		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	9	1	1	5	1	6	4	0	0
3	3	0	9	0	0	10	9	8	7
4	9	9	6	3	5	10	9	7	0
5	9	0	0	0	8	9	8	7	0
6	10	0	7	6	6	8	7	0	0
7	5	9	0	0	8	11	0	0	0
8	4	3	0	10	8	11	0	0	0
9	7	8	5	0	6	11	0	0	0
10	7	0	0	0	0	11	0	0	0
11	1	6	8	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 19		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	11	0
2	7	0	1	1	2	3	0	0	0
3	2	0	0	0	3	6	5	0	0
4	10	1	5	4	0	10	6	0	0
5	7	8	0	0	4	10	7	0	0
6	4	0	6	0	10	7	0	0	0
7	1	6	0	8	10	9	8	0	0
8	10	0	10	8	0	12	0	0	0
9	8	0	6	0	0	12	0	0	0
10	1	7	8	7	7	12	0	0	0
11	4	8	0	8	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 20		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	10	0	0	0	2	6	5	0	0
3	8	1	1	7	6	6	5	0	0
4	5	10	0	0	0	10	5	0	0
5	2	9	4	3	6	11	8	7	0
6	8	0	4	0	7	10	8	0	0
7	10	3	10	10	0	9	0	0	0
8	2	0	7	0	6	12	0	0	0
9	5	0	0	3	5	12	0	0	0
10	10	7	0	0	0	12	0	0	0
11	6	0	10	7	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 21		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	0
2	6	1	1	1	0	3	0	0	0
3	5	7	0	7	0	11	8	7	0
4	8	4	10	0	1	7	6	0	0
5	4	10	9	0	0	11	7	0	0
6	4	9	0	10	0	11	10	9	0
7	2	0	7	0	9	10	9	0	0
8	3	7	0	0	0	10	9	0	0
9	2	4	3	0	0	12	0	0	0
10	6	5	6	6	8	12	0	0	0
11	9	7	6	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 22		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	5
2	10	0	1	1	0	11	8	6	0
3	5	2	0	7	1	11	8	6	0
4	9	0	0	8	0	11	8	6	0
5	6	0	10	0	2	11	9	7	0
6	2	0	0	8	9	9	7	0	0
7	6	0	7	5	8	10	0	0	0
8	6	0	7	0	7	9	0	0	0
9	7	0	0	7	7	12	0	0	0
10	7	10	6	6	8	12	0	0	0
11	3	0	5	6	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 23		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	6	1	1	0	1	9	5	3	0
3	10	0	0	1	0	8	4	0	0
4	7	0	0	7	0	11	6	0	0
5	3	10	0	9	0	11	7	0	0
6	10	5	0	7	9	7	0	0	0
7	2	6	9	8	9	10	0	0	0
8	8	0	0	0	6	10	0	0	0
9	2	0	0	3	6	10	0	0	0
10	5	7	8	7	0	12	0	0	0
11	4	7	6	0	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 24		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	9	0
2	7	1	0	0	0	3	0	0	0
3	5	0	0	1	3	8	5	0	0
4	4	8	1	0	7	5	0	0	0
5	4	0	7	0	10	10	6	0	0
6	3	0	9	6	5	7	0	0	0
7	7	7	3	8	0	11	0	0	0
8	1	6	0	7	5	10	0	0	0
9	2	0	10	5	6	10	0	0	0
10	9	8	0	9	0	12	0	0	0
11	7	0	6	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 25		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	4	1	1	0	1	4	3	0	0
3	8	0	5	1	0	11	8	6	5
4	5	7	9	0	10	11	10	6	0
5	3	9	0	0	0	10	9	0	0
6	7	4	6	5	8	7	0	0	0
7	10	6	7	8	7	9	0	0	0
8	4	9	8	9	6	9	0	0	0
9	8	0	0	7	0	12	0	0	0
10	9	0	0	0	0	12	0	0	0
11	7	0	0	0	4	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 26		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	7	0
2	10	1	2	0	1	11	4	0	0
3	4	4	0	0	3	11	4	0	0
4	4	0	0	1	0	8	5	0	0
5	6	0	4	4	0	6	0	0	0
6	7	9	10	7	0	10	9	0	0
7	1	0	6	9	9	10	9	0	0
8	6	7	8	8	9	9	0	0	0
9	8	9	4	0	0	12	0	0	0
10	10	0	0	0	0	12	0	0	0
11	6	0	8	7	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 27		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	6
2	10	0	3	3	2	8	4	0	0
3	1	0	7	0	0	8	4	0	0
4	8	1	10	3	7	11	9	7	0
5	9	7	5	10	4	11	8	7	0
6	7	0	6	9	8	11	8	7	0
7	2	9	0	0	5	10	0	0	0
8	10	0	0	0	10	10	0	0	0
9	7	0	5	7	0	10	0	0	0
10	2	7	0	4	0	12	0	0	0
11	8	6	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 28		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	5	1	1	0	1	6	4	0	0
3	6	6	0	0	6	11	8	7	6
4	3	0	0	1	9	5	0	0	0
5	7	6	0	5	6	11	10	8	0
6	1	0	0	0	0	10	9	0	0
7	9	6	8	0	0	10	9	0	0
8	4	9	0	9	4	9	0	0	0
9	8	0	0	0	5	12	0	0	0
10	4	7	0	9	8	12	0	0	0
11	2	7	9	6	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 29		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	9	0
2	9	0	1	0	1	3	0	0	0
3	1	0	0	1	10	6	5	0	0
4	9	0	9	10	7	11	5	0	0
5	2	1	6	7	5	7	0	0	0
6	3	0	4	0	0	11	10	0	0
7	8	9	7	0	10	8	0	0	0
8	6	0	8	0	5	10	0	0	0
9	2	5	0	0	3	10	0	0	0
10	4	9	0	0	9	12	0	0	0
11	10	0	7	0	4	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 30		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	2	0	2	1	1	6	4	0	0
3	5	0	3	0	0	6	4	0	0
4	1	1	0	10	8	11	10	8	0
5	5	0	0	7	9	6	0	0	0
6	2	8	0	7	0	10	7	0	0
7	7	6	8	8	5	9	0	0	0
8	8	4	0	0	0	9	0	0	0
9	8	8	0	5	0	12	0	0	0
10	4	9	10	0	7	12	0	0	0
11	5	0	7	4	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 31		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	11
2	3	2	1	0	1	4	0	0	0
3	1	0	7	1	8	4	0	0	0
4	4	0	10	5	6	7	6	0	0
5	3	7	7	0	0	7	6	0	0
6	6	0	8	0	7	10	9	8	0
7	7	10	1	0	7	9	8	0	0
8	6	6	0	0	0	12	0	0	0
9	8	0	0	0	0	12	0	0	0
10	4	4	8	8	7	12	0	0	0
11	7	7	0	10	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 32		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	4	0	5	1	0	6	4	0	0
3	8	0	3	0	0	4	0	0	0
4	7	1	10	0	1	11	8	7	0
5	4	9	5	7	9	8	7	0	0
6	10	9	9	0	9	9	7	0	0
7	1	0	7	7	5	10	0	0	0
8	7	5	0	7	7	9	0	0	0
9	4	0	0	8	6	12	0	0	0
10	4	0	0	0	0	12	0	0	0
11	10	0	3	0	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 33		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	10	0	3	0	6	3	0	0	0
3	8	1	0	5	2	8	7	6	5
4	4	5	4	0	5	10	8	7	0
5	8	8	0	4	6	11	10	9	0
6	9	0	10	0	7	10	9	0	0
7	4	0	0	0	6	11	0	0	0
8	3	7	0	0	4	9	0	0	0
9	9	0	7	5	0	12	0	0	0
10	10	5	0	10	8	12	0	0	0
11	10	10	0	0	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 34		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	7	0
2	6	0	1	0	0	11	4	0	0
3	2	0	7	1	0	11	4	0	0
4	2	1	6	7	0	6	5	0	0
5	9	6	4	5	0	10	8	0	0
6	7	7	0	9	0	8	0	0	0
7	6	0	8	4	2	8	0	0	0
8	2	8	10	0	10	9	0	0	0
9	2	6	6	0	0	12	0	0	0
10	2	0	0	8	0	12	0	0	0
11	5	8	6	8	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 35		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	1	2	1	1	0	11	6	3	0
3	3	3	6	5	0	5	0	0	0
4	7	0	8	7	1	5	0	0	0
5	1	10	0	0	0	10	9	7	0
6	5	0	0	7	0	10	9	7	0
7	8	5	0	9	8	8	0	0	0
8	4	0	5	7	0	12	0	0	0
9	1	10	10	0	9	12	0	0	0
10	4	0	8	7	0	12	0	0	0
11	3	0	4	5	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 36		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	1	2	1	1	0	11	6	3	0
3	3	3	6	5	0	5	0	0	0
4	7	0	8	7	1	5	0	0	0
5	1	10	0	0	0	10	9	7	0
6	5	0	0	7	0	10	9	7	0
7	8	5	0	9	8	8	0	0	0
8	4	0	5	7	0	12	0	0	0
9	1	10	10	0	9	12	0	0	0
10	4	0	8	7	0	12	0	0	0
11	3	0	4	5	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 37		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	4	1	1	2	1	11	4	0	0
3	7	0	7	8	0	11	7	5	0
4	2	10	0	9	0	6	5	0	0
5	9	4	9	4	10	10	8	0	0
6	6	6	0	9	0	7	0	0	0
7	9	0	7	4	0	9	0	0	0
8	3	10	0	0	0	9	0	0	0
9	5	0	0	10	6	12	0	0	0
10	6	5	0	2	0	12	0	0	0
11	9	0	0	6	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 38		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	6
2	7	0	1	2	0	7	5	0	0
3	8	1	10	4	3	8	5	0	0
4	2	9	0	5	6	8	5	0	0
5	8	5	3	0	0	11	10	9	0
6	4	0	0	0	3	11	10	9	0
7	1	0	9	7	0	8	0	0	0
8	9	0	0	0	8	10	9	0	0
9	2	0	8	10	6	12	0	0	0
10	5	0	5	7	10	12	0	0	0
11	3	9	0	7	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 39		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	6	0
2	10	0	1	0	0	5	3	0	0
3	4	0	0	1	1	11	9	7	0
4	4	1	6	0	10	9	5	0	0
5	7	7	7	9	0	11	8	0	0
6	7	7	0	0	0	7	0	0	0
7	5	0	0	8	3	8	0	0	0
8	10	10	7	0	0	10	0	0	0
9	4	7	6	5	10	10	0	0	0
10	9	6	8	7	0	12	0	0	0
11	8	4	7	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 40		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	8	9
2	9	0	0	0	0	3	0	0	0
3	9	1	1	1	1	7	5	0	0
4	10	6	7	6	0	7	5	0	0
5	10	8	7	8	6	6	0	0	0
6	8	0	8	0	0	11	10	0	0
7	7	8	4	5	0	11	10	0	0
8	8	8	3	0	0	11	10	0	0
9	4	0	8	0	7	11	10	0	0
10	10	5	0	10	10	12	0	0	0
11	7	0	10	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 41		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	7	0	2	0	2	8	3	0	0
3	3	1	5	0	0	7	5	0	0
4	5	10	0	1	0	8	7	6	0
5	6	0	8	0	0	11	6	0	0
6	3	8	10	9	3	10	9	0	0
7	10	3	5	7	10	10	0	0	0
8	9	5	0	7	0	9	0	0	0
9	5	0	0	0	8	12	0	0	0
10	6	7	0	0	7	12	0	0	0
11	10	8	6	6	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 42		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	7	0	2	0	2	8	3	0	0
3	3	1	5	0	0	7	5	0	0
4	5	10	0	1	0	8	7	6	0
5	6	0	8	0	0	11	6	0	0
6	3	8	10	9	3	10	9	0	0
7	10	3	5	7	10	10	0	0	0
8	9	5	0	7	0	9	0	0	0
9	5	0	0	0	8	12	0	0	0
10	6	7	0	0	7	12	0	0	0
11	10	8	6	6	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 43		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	7	0
2	3	0	3	1	0	3	0	0	0
3	4	0	9	0	0	6	5	0	0
4	10	1	4	0	0	5	0	0	0
5	7	10	0	0	0	11	10	8	0
6	4	0	5	8	1	8	0	0	0
7	9	5	10	0	10	8	0	0	0
8	4	0	5	0	0	9	0	0	0
9	3	7	4	0	9	12	0	0	0
10	1	6	4	9	3	12	0	0	0
11	3	7	10	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 44		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	6	0
2	6	0	1	1	0	3	0	0	0
3	7	1	6	6	1	7	5	0	0
4	6	7	0	8	4	7	5	0	0
5	3	0	0	0	9	11	10	9	8
6	4	9	9	8	0	9	7	0	0
7	9	8	10	5	0	8	0	0	0
8	5	0	4	0	7	12	0	0	0
9	6	0	0	6	9	12	0	0	0
10	3	5	0	8	6	12	0	0	0
11	2	0	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 45		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	5	0	2	1	0	7	6	5	0
3	10	1	7	0	3	8	7	6	0
4	6	9	5	0	4	5	0	0	0
5	3	6	6	0	0	11	10	8	0
6	3	7	0	0	0	11	9	0	0
7	6	8	0	0	4	10	9	0	0
8	6	0	4	0	8	9	0	0	0
9	9	0	7	9	7	12	0	0	0
10	5	0	10	0	10	12	0	0	0
11	1	5	7	8	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 46		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	2	1	0	0	0	8	4	0	0
3	6	0	1	1	1	11	8	7	0
4	10	7	7	0	6	7	6	0	0
5	1	0	0	8	5	10	6	0	0
6	4	0	0	0	10	11	9	0	0
7	6	5	9	7	0	10	9	0	0
8	5	0	0	5	7	10	9	0	0
9	9	7	0	9	8	12	0	0	0
10	10	9	0	0	6	12	0	0	0
11	4	7	7	0	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 47		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	6
2	6	1	0	0	1	3	0	0	0
3	5	10	2	1	10	8	7	0	0
4	5	0	0	0	9	8	7	0	0
5	1	0	6	0	6	10	7	0	0
6	6	0	0	0	6	11	10	9	0
7	10	6	10	5	0	11	9	0	0
8	10	7	4	7	6	10	9	0	0
9	10	6	8	9	0	12	0	0	0
10	4	0	0	8	5	12	0	0	0
11	6	0	0	0	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 48		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	0
2	8	3	2	1	1	8	3	0	0
3	2	2	7	9	0	11	6	0	0
4	3	0	8	0	0	11	6	0	0
5	9	0	4	7	0	11	7	0	0
6	7	0	0	4	8	7	0	0	0
7	8	0	10	6	0	10	9	0	0
8	10	9	5	0	0	9	0	0	0
9	2	0	6	7	8	12	0	0	0
10	6	10	0	8	6	12	0	0	0
11	6	0	0	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 49		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	8	0
2	1	1	1	1	1	5	4	0	0
3	6	6	4	10	6	5	4	0	0
4	4	0	8	0	0	7	6	0	0
5	2	0	6	0	8	6	0	0	0
6	5	0	10	0	0	10	9	0	0
7	5	7	0	0	5	11	9	0	0
8	1	0	9	6	10	9	0	0	0
9	1	0	4	5	0	12	0	0	0
10	5	8	6	8	0	12	0	0	0
11	9	8	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 50		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	5
2	6	1	0	0	1	8	7	6	0
3	5	0	1	0	7	8	7	6	0
4	5	7	6	1	8	11	7	6	0
5	9	6	0	0	0	7	6	0	0
6	9	0	0	0	7	10	9	0	0
7	6	6	9	0	0	10	9	0	0
8	4	6	8	7	5	11	9	0	0
9	1	6	6	0	8	12	0	0	0
10	4	9	0	10	6	12	0	0	0
11	8	7	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 51		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	4	0	0	0	0	10	7	4	0
3	5	0	1	0	0	10	9	7	6
4	4	1	9	0	1	9	6	0	0
5	1	9	5	3	8	10	9	0	0
6	4	9	0	0	5	8	0	0	0
7	6	6	0	0	7	8	0	0	0
8	7	0	0	4	6	11	0	0	0
9	2	4	0	7	8	11	0	0	0
10	8	6	7	10	0	11	0	0	0
11	10	7	8	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 52		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	1	3	0	1	0	7	6	5	0
3	1	3	0	0	1	4	0	0	0
4	3	7	1	8	0	11	7	5	0
5	8	6	0	0	8	10	9	8	0
6	3	7	0	9	0	11	10	9	0
7	9	0	6	6	7	8	0	0	0
8	8	0	0	0	2	12	0	0	0
9	5	0	8	5	8	12	0	0	0
10	9	10	9	7	0	12	0	0	0
11	7	6	0	0	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 53		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	10	0	1	0	1	8	4	0	0
3	8	1	4	0	0	7	6	0	0
4	2	0	10	0	0	11	7	0	0
5	2	7	0	1	0	11	8	0	0
6	2	0	8	0	8	11	9	0	0
7	7	0	0	0	0	9	0	0	0
8	9	6	5	0	9	9	0	0	0
9	2	7	5	10	3	10	0	0	0
10	1	9	6	5	8	12	0	0	0
11	1	0	9	8	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 54		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	1	0	0	1	1	5	4	0	0
3	6	1	0	8	0	4	0	0	0
4	6	6	1	0	6	8	6	0	0
5	5	8	0	9	0	11	6	0	0
6	3	9	7	0	0	7	0	0	0
7	9	0	0	3	6	9	0	0	0
8	1	6	10	0	0	10	0	0	0
9	4	6	6	0	7	12	0	0	0
10	2	0	0	9	9	12	0	0	0
11	2	6	6	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 55		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	5	6	0
2	10	1	0	0	0	8	4	3	0
3	2	4	1	0	0	9	7	0	0
4	1	0	7	0	0	9	7	0	0
5	1	6	6	1	2	11	8	0	0
6	5	0	4	9	0	7	0	0	0
7	2	0	0	0	0	11	10	0	0
8	5	8	8	6	7	9	0	0	0
9	5	6	0	9	0	10	0	0	0
10	10	8	6	5	5	12	0	0	0
11	8	9	10	0	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 56		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	9	0	0	0	2	5	4	0	0
3	1	0	3	1	0	11	7	6	5
4	4	0	6	6	0	11	10	7	6
5	8	0	2	5	7	10	9	8	0
6	6	0	7	7	0	9	8	0	0
7	4	0	6	10	10	8	0	0	0
8	5	0	10	6	0	12	0	0	0
9	9	0	10	10	0	12	0	0	0
10	5	2	4	0	0	12	0	0	0
11	1	10	6	3	5	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 57		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	7
2	4	1	1	0	0	8	5	0	0
3	1	3	0	1	0	8	6	0	0
4	5	0	0	5	1	11	8	0	0
5	8	0	0	4	7	6	0	0	0
6	6	0	4	6	0	11	10	9	0
7	1	0	7	9	0	11	10	9	0
8	9	0	7	0	7	10	9	0	0
9	10	0	7	0	0	12	0	0	0
10	8	10	9	8	8	12	0	0	0
11	8	10	7	9	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 58		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	4	0	2	1	1	7	5	4	0
3	2	0	10	0	8	7	6	5	0
4	5	0	7	6	0	9	6	0	0
5	2	1	4	9	0	11	10	0	0
6	2	7	4	5	0	8	0	0	0
7	5	6	0	6	0	8	0	0	0
8	1	0	5	8	0	10	0	0	0
9	4	7	0	0	0	11	0	0	0
10	7	0	6	7	9	12	0	0	0
11	6	9	10	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 59		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	9	0	1	0	1	8	6	5	0
3	6	3	0	0	9	8	6	5	0
4	5	0	0	0	7	5	0	0	0
5	3	4	10	0	6	11	9	7	0
6	2	0	2	1	0	10	9	0	0
7	1	0	0	6	0	10	0	0	0
8	10	0	9	9	7	9	0	0	0
9	3	10	0	9	7	12	0	0	0
10	3	8	8	0	5	12	0	0	0
11	4	5	0	5	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 60		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	0
2	10	1	0	0	0	3	0	0	0
3	9	0	0	1	0	11	7	6	0
4	8	0	0	8	4	11	7	6	0
5	5	6	1	8	0	11	9	8	7
6	4	5	0	6	0	9	8	0	0
7	2	0	10	8	0	10	0	0	0
8	2	10	0	0	2	12	0	0	0
9	2	6	7	10	10	12	0	0	0
10	7	9	0	4	0	12	0	0	0
11	3	5	6	3	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 61		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	1	1	1	1	1	11	9	7	6
3	5	4	0	0	9	6	5	0	0
4	2	9	5	8	10	11	9	6	0
5	1	0	10	0	0	11	9	8	0
6	3	3	8	0	0	8	0	0	0
7	3	0	0	7	4	8	0	0	0
8	9	6	0	0	0	10	0	0	0
9	2	10	0	0	5	10	0	0	0
10	5	7	0	8	6	12	0	0	0
11	10	8	0	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 62		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	10	5	0	0	1	7	3	0	0
3	9	4	0	1	3	6	5	0	0
4	1	0	0	10	7	7	5	0	0
5	3	10	1	0	0	11	10	9	8
6	8	3	0	0	0	9	8	0	0
7	1	7	4	5	10	8	0	0	0
8	2	5	0	7	7	12	0	0	0
9	10	0	0	7	0	12	0	0	0
10	8	8	9	0	8	12	0	0	0
11	5	6	10	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 63		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	6
2	2	3	0	2	0	11	5	0	0
3	4	0	1	5	1	11	5	0	0
4	5	0	10	10	7	11	9	8	0
5	8	0	0	4	7	9	7	0	0
6	8	4	0	9	7	11	8	0	0
7	6	8	5	5	0	8	0	0	0
8	3	6	0	7	8	10	0	0	0
9	4	5	5	0	0	10	0	0	0
10	9	0	9	0	0	12	0	0	0
11	3	10	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 64		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	3	4	2	1	1	10	8	7	0
3	6	0	0	6	0	7	5	0	0
4	5	0	0	7	10	8	6	0	0
5	4	4	0	8	5	10	9	0	0
6	1	0	0	0	7	7	0	0	0
7	8	6	0	0	0	9	0	0	0
8	8	10	4	0	7	9	0	0	0
9	6	0	6	9	0	11	0	0	0
10	10	5	10	8	0	11	0	0	0
11	8	7	8	3	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 65		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	6	0
2	5	0	0	1	1	8	4	0	0
3	6	0	3	6	7	8	4	0	0
4	8	0	0	10	0	5	0	0	0
5	8	1	8	0	7	7	0	0	0
6	8	0	4	7	0	7	0	0	0
7	10	0	6	4	9	10	9	0	0
8	4	0	4	7	0	11	0	0	0
9	4	10	10	7	0	12	0	0	0
10	1	0	7	6	0	12	0	0	0
11	1	7	6	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 66		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	8	1	1	0	5	7	6	4	0
3	5	6	7	1	6	6	4	0	0
4	7	0	0	0	0	11	10	9	8
5	9	0	0	0	10	6	0	0	0
6	6	8	8	0	4	10	9	8	0
7	3	10	7	9	6	10	9	8	0
8	2	0	4	9	4	12	0	0	0
9	5	0	7	0	0	12	0	0	0
10	8	5	0	0	0	12	0	0	0
11	7	0	8	5	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 67		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	5
2	10	1	4	0	1	7	6	0	0
3	8	6	4	0	0	9	7	0	0
4	3	8	0	0	10	9	7	0	0
5	10	6	4	1	0	11	7	0	0
6	3	0	0	0	0	10	9	0	0
7	6	9	7	9	0	8	0	0	0
8	10	6	8	0	5	10	0	0	0
9	9	6	7	8	8	11	0	0	0
10	1	0	4	6	0	12	0	0	0
11	10	0	10	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 68		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	1	0	0	0	0	11	5	0	0
3	9	0	0	0	4	6	0	0	0
4	5	0	1	1	0	5	0	0	0
5	5	2	4	7	0	8	7	0	0
6	7	4	5	7	0	8	7	0	0
7	2	8	0	6	0	10	9	0	0
8	2	8	8	0	4	9	0	0	0
9	9	0	0	7	6	12	0	0	0
10	3	4	9	8	6	12	0	0	0
11	5	10	9	0	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 69		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	9	0	0	1	0	6	5	0	0
3	3	5	0	0	2	4	0	0	0
4	1	0	1	3	5	7	5	0	0
5	4	0	0	10	8	11	10	8	0
6	6	5	0	0	4	8	7	0	0
7	2	0	0	8	0	9	0	0	0
8	9	10	10	6	0	12	0	0	0
9	6	4	4	8	10	12	0	0	0
10	1	4	7	0	7	12	0	0	0
11	5	8	8	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 70		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	8	1	0	0	1	5	4	0	0
3	2	0	0	1	6	11	10	8	6
4	8	6	2	0	8	11	10	6	0
5	8	0	0	7	7	11	10	6	0
6	3	0	0	8	0	7	0	0	0
7	4	0	0	6	7	9	0	0	0
8	2	7	0	6	8	9	0	0	0
9	4	4	0	5	6	12	0	0	0
10	4	8	10	0	5	12	0	0	0
11	1	10	0	9	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 71		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	0
2	6	1	1	0	1	7	3	0	0
3	3	9	7	0	0	11	6	0	0
4	5	0	4	0	0	10	7	0	0
5	4	0	5	2	5	10	7	0	0
6	10	0	0	4	0	8	0	0	0
7	3	7	8	8	6	11	9	0	0
8	8	5	10	0	10	10	9	0	0
9	9	9	0	0	0	12	0	0	0
10	10	5	6	10	8	12	0	0	0
11	3	0	7	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 72		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	9	4	0	0	0	7	4	3	0
3	4	2	1	0	1	11	6	5	0
4	7	10	0	1	0	11	10	6	0
5	1	3	0	9	0	10	9	8	0
6	7	7	0	9	9	9	8	0	0
7	7	6	8	0	0	9	8	0	0
8	1	7	9	0	0	12	0	0	0
9	6	0	0	0	8	12	0	0	0
10	6	10	5	5	6	12	0	0	0
11	2	5	7	6	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 73		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	2	0	0	5	1	6	5	0	0
3	8	3	0	5	5	10	8	6	0
4	1	0	0	8	5	7	6	0	0
5	8	2	0	0	9	9	7	0	0
6	3	10	1	10	5	9	0	0	0
7	10	0	9	0	7	8	0	0	0
8	9	9	0	0	0	11	0	0	0
9	10	0	10	2	0	11	0	0	0
10	3	0	6	0	9	11	0	0	0
11	5	6	4	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 74		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	6	7
2	1	1	0	1	0	4	0	0	0
3	7	7	1	0	2	4	0	0	0
4	6	0	7	7	0	5	0	0	0
5	8	0	9	8	6	11	10	8	0
6	10	10	0	8	0	11	8	0	0
7	1	3	4	6	0	10	9	0	0
8	5	10	4	0	0	9	0	0	0
9	3	5	0	0	0	12	0	0	0
10	5	0	8	4	0	12	0	0	0
11	3	0	9	8	10	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 75		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	5	0
2	2	1	1	0	1	3	0	0	0
3	7	7	10	1	8	11	7	6	0
4	1	7	0	0	4	11	7	6	0
5	3	8	6	0	0	10	8	0	0
6	7	8	4	4	0	10	9	0	0
7	3	7	9	0	8	8	0	0	0
8	4	4	0	0	0	9	0	0	0
9	10	0	6	10	6	12	0	0	0
10	8	0	0	0	0	12	0	0	0
11	7	0	0	9	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 76		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	10	3	1	0	0	6	4	3	0
3	4	0	0	1	1	7	5	0	0
4	5	0	7	6	8	5	0	0	0
5	4	0	7	8	0	11	10	9	8
6	9	5	0	6	8	10	9	0	0
7	5	6	4	0	0	8	0	0	0
8	10	10	4	0	7	12	0	0	0
9	9	0	7	5	0	12	0	0	0
10	6	0	9	10	0	12	0	0	0
11	5	6	9	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 77		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	4	0	1	1	0	6	5	0	0
3	1	1	0	0	2	8	6	0	0
4	8	0	0	0	3	10	6	0	0
5	6	0	8	8	8	8	7	0	0
6	3	0	0	0	5	11	7	0	0
7	5	9	0	5	10	9	0	0	0
8	7	8	5	8	0	10	0	0	0
9	2	0	0	0	8	12	0	0	0
10	9	5	8	5	6	12	0	0	0
11	3	7	8	9	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 78		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	3	0	2	0	2	6	4	3	0
3	8	0	0	2	3	11	8	7	0
4	8	1	4	3	0	5	0	0	0
5	1	5	10	9	10	10	8	0	0
6	9	10	4	10	7	11	8	0	0
7	1	0	0	0	0	10	9	0	0
8	6	0	8	0	10	9	0	0	0
9	8	0	6	0	5	12	0	0	0
10	1	8	0	0	7	12	0	0	0
11	2	0	8	0	4	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 79		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	7	1	1	0	1	8	7	6	0
3	5	0	0	1	0	8	7	5	0
4	3	8	4	5	7	7	6	5	0
5	9	6	7	4	0	11	10	9	0
6	5	8	0	0	8	11	10	9	0
7	1	0	8	0	0	11	10	0	0
8	1	0	6	10	0	9	0	0	0
9	8	6	10	10	0	12	0	0	0
10	5	0	6	0	8	12	0	0	0
11	4	7	0	0	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 80		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	6	0	1	0	1	7	3	0	0
3	8	0	6	1	0	11	6	5	0
4	1	1	8	6	0	11	5	0	0
5	2	10	9	9	0	10	9	8	0
6	4	8	5	0	7	10	9	8	0
7	6	5	6	7	0	9	8	0	0
8	8	0	4	0	7	12	0	0	0
9	5	0	0	0	0	12	0	0	0
10	6	0	6	7	8	12	0	0	0
11	2	0	9	0	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 81		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	7	6	1	0	4	8	7	5	0
3	9	0	9	0	3	7	6	0	0
4	10	1	0	0	8	7	5	0	0
5	9	0	0	1	10	11	9	0	0
6	1	7	0	0	5	11	9	0	0
7	2	10	0	5	0	11	10	0	0
8	10	10	0	0	0	11	10	0	0
9	9	0	8	10	0	10	0	0	0
10	1	2	8	5	3	12	0	0	0
11	5	0	4	9	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 82		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	10	1	0	2	1	7	3	0	0
3	6	5	0	0	10	9	6	5	0
4	2	0	0	0	9	9	7	0	0
5	7	0	0	0	0	11	8	0	0
6	4	7	1	0	0	11	10	0	0
7	1	0	8	6	7	8	0	0	0
8	5	2	0	0	4	10	0	0	0
9	3	9	7	0	8	10	0	0	0
10	8	9	6	10	6	12	0	0	0
11	8	9	8	0	3	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 83		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	6	0	1	1	1	5	4	3	0
3	4	1	0	10	0	11	9	8	0
4	4	6	0	3	7	6	0	0	0
5	1	7	0	10	0	11	8	0	0
6	1	0	0	4	7	7	0	0	0
7	1	6	6	0	9	11	10	0	0
8	3	0	10	0	0	10	0	0	0
9	8	7	6	10	0	10	0	0	0
10	4	9	0	4	6	12	0	0	0
11	1	0	7	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 84		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	7	0	0
2	6	1	1	2	0	5	3	0	0
3	3	6	0	6	1	9	4	0	0
4	1	9	5	10	0	11	6	0	0
5	1	0	10	8	6	11	8	0	0
6	7	0	9	0	9	8	0	0	0
7	10	0	0	0	5	9	0	0	0
8	9	8	0	4	7	10	0	0	0
9	3	0	6	6	0	10	0	0	0
10	10	0	0	0	0	12	0	0	0
11	3	6	5	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 85		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	1	1	1	0	1	7	4	0	0
3	10	0	4	1	6	6	0	0	0
4	3	7	0	0	6	11	10	8	0
5	1	0	7	0	6	11	10	8	0
6	4	0	0	0	0	7	0	0	0
7	1	0	0	6	7	11	10	9	0
8	4	7	5	5	9	9	0	0	0
9	6	0	10	7	7	12	0	0	0
10	7	0	9	8	0	12	0	0	0
11	9	9	0	9	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 86		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	7	0
2	9	1	0	1	0	6	4	0	0
3	10	0	0	9	0	6	4	0	0
4	2	9	2	6	1	9	8	5	0
5	1	0	4	9	9	11	10	0	0
6	4	10	10	0	0	11	10	0	0
7	8	7	8	0	6	8	0	0	0
8	6	0	0	8	0	10	0	0	0
9	5	5	0	2	6	10	0	0	0
10	2	4	0	7	0	12	0	0	0
11	1	0	6	0	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 87		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	8	5	1	1	1	6	5	4	0
3	10	0	10	8	7	7	6	5	0
4	6	4	7	0	0	11	10	7	0
5	10	10	8	0	7	11	10	9	0
6	6	6	0	6	5	10	8	0	0
7	1	8	0	0	7	9	0	0	0
8	4	3	0	0	0	9	0	0	0
9	9	0	4	6	0	12	0	0	0
10	6	0	0	0	8	12	0	0	0
11	8	0	0	9	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 88		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	7
2	7	0	1	1	0	6	4	0	0
3	2	3	8	4	0	11	6	0	0
4	6	8	9	0	0	11	8	0	0
5	3	5	6	9	1	11	8	0	0
6	6	6	0	10	10	8	0	0	0
7	4	0	0	7	8	8	0	0	0
8	1	10	0	0	5	10	9	0	0
9	2	0	8	0	0	12	0	0	0
10	9	4	0	8	0	12	0	0	0
11	1	0	4	3	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 89		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	3	2	0	0	1	6	5	4	0
3	9	7	0	3	0	6	4	0	0
4	10	0	0	1	7	11	10	8	7
5	2	5	1	6	6	10	7	0	0
6	10	0	6	9	8	10	9	0	0
7	4	0	10	10	0	9	0	0	0
8	4	0	6	8	7	9	0	0	0
9	8	0	0	0	6	12	0	0	0
10	5	10	7	0	0	12	0	0	0
11	7	0	0	5	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 90		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	7	0
2	1	0	1	4	1	6	4	0	0
3	10	0	0	2	0	5	4	0	0
4	7	1	0	0	0	11	10	8	0
5	5	7	9	10	9	11	10	8	0
6	10	0	8	0	0	11	10	9	0
7	8	4	4	0	0	11	10	9	0
8	9	10	6	7	4	9	0	0	0
9	6	0	0	0	10	12	0	0	0
10	5	8	8	6	6	12	0	0	0
11	5	0	0	7	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 91		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	5	0	0	5	1	7	4	0	0
3	10	0	1	0	0	5	4	0	0
4	10	3	0	3	0	11	8	6	0
5	7	4	7	9	10	6	0	0	0
6	3	10	0	10	0	10	9	0	0
7	2	0	4	4	5	11	10	0	0
8	9	0	7	2	0	9	0	0	0
9	6	0	0	0	0	12	0	0	0
10	7	5	8	0	10	12	0	0	0
11	8	8	9	9	4	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 92		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	7	0	1	0	1	7	6	0	0
3	8	1	0	0	10	11	4	0	0
4	9	0	10	1	7	7	0	0	0
5	1	0	0	10	0	6	0	0	0
6	10	4	5	9	6	11	10	9	0
7	10	6	0	0	0	10	8	0	0
8	9	7	7	7	7	9	0	0	0
9	2	0	7	0	3	12	0	0	0
10	6	9	0	0	0	12	0	0	0
11	9	9	0	3	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 93		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	9	1	1	1	1	5	4	0	0
3	6	0	9	9	5	11	5	0	0
4	5	0	8	0	0	11	10	7	0
5	2	3	0	6	7	9	6	0	0
6	1	8	0	0	7	10	8	0	0
7	9	8	0	0	0	9	8	0	0
8	3	7	0	7	0	12	0	0	0
9	3	0	6	7	10	12	0	0	0
10	3	7	6	0	0	12	0	0	0
11	8	8	6	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 94		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	5	0	0
2	8	0	0	1	0	3	0	0	0
3	9	2	1	9	0	8	6	4	0
4	3	5	0	0	0	11	10	7	0
5	10	7	8	8	1	10	8	7	0
6	7	5	8	6	7	10	9	0	0
7	2	0	0	0	8	9	0	0	0
8	9	0	6	0	8	9	0	0	0
9	7	10	5	0	4	12	0	0	0
10	10	0	0	6	8	12	0	0	0
11	2	7	8	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 95		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	0	0	0
2	3	1	0	0	1	6	3	0	0
3	9	0	3	2	4	11	8	4	0
4	2	8	0	0	0	5	0	0	0
5	3	4	0	8	0	10	7	0	0
6	2	9	7	0	7	7	0	0	0
7	2	5	10	10	0	9	0	0	0
8	2	7	5	0	8	10	0	0	0
9	3	0	5	0	0	12	0	0	0
10	7	6	0	4	10	12	0	0	0
11	4	8	0	0	6	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 96		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	4	0	0	1	1	7	6	4	0
3	1	0	0	0	9	8	7	6	5
4	5	1	1	0	6	8	5	0	0
5	8	4	6	0	7	11	9	0	0
6	3	10	10	0	0	11	9	0	0
7	1	0	7	7	6	10	9	0	0
8	3	8	0	9	4	10	0	0	0
9	2	7	0	4	0	12	0	0	0
10	4	0	0	0	8	12	0	0	0
11	6	0	6	9	7	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 97		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	0	0
2	7	1	0	1	1	7	6	5	4
3	10	0	2	0	0	4	0	0	0
4	10	0	5	0	8	11	10	8	0
5	9	0	7	10	0	11	10	8	0
6	3	6	0	8	0	11	10	9	0
7	6	8	0	5	6	10	9	0	0
8	1	0	5	3	0	9	0	0	0
9	2	0	7	6	7	12	0	0	0
10	4	0	0	9	0	12	0	0	0
11	8	9	10	6	8	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 98		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	5	0
2	9	1	4	2	1	6	4	0	0
3	8	0	0	0	0	11	6	0	0
4	8	5	6	0	6	11	10	8	0
5	10	0	5	0	9	11	9	7	0
6	1	0	6	6	0	8	7	0	0
7	1	0	0	10	7	10	0	0	0
8	8	0	10	0	6	9	0	0	0
9	8	10	8	6	6	12	0	0	0
10	6	0	3	0	4	12	0	0	0
11	2	8	0	0	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 99		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	6	0
2	2	0	0	1	1	5	4	0	0
3	9	0	0	10	5	4	0	0	0
4	8	0	0	3	0	11	10	9	7
5	2	2	0	7	0	11	10	9	7
6	2	0	0	5	0	9	7	0	0
7	5	7	1	0	9	8	0	0	0
8	2	4	6	9	7	12	0	0	0
9	1	0	7	0	5	12	0	0	0
10	2	7	0	6	0	12	0	0	0
11	9	10	10	7	9	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 100		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	4	0	0
2	5	1	1	1	0	3	0	0	0
3	3	0	2	0	0	8	5	0	0
4	2	4	10	0	1	11	7	6	0
5	1	10	0	0	0	7	6	0	0
6	1	4	0	6	9	10	0	0	0
7	7	3	0	6	6	9	0	0	0
8	5	6	0	8	0	11	0	0	0
9	3	10	10	6	0	12	0	0	0
10	4	10	0	9	8	12	0	0	0
11	3	0	7	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

Project 0		Resources				Successors			
Task	Duration	1	2	3	4	1	2	3	4
1	0	0	0	0	0	2	3	4	0
2	10	1	0	1	0	9	8	6	0
3	7	8	1	8	1	9	6	0	0
4	7	0	10	7	6	7	5	0	0
5	1	5	4	0	0	11	9	0	0
6	5	0	9	0	8	7	0	0	0
7	5	0	0	10	0	11	10	0	0
8	10	7	6	0	9	11	10	0	0
9	1	7	6	0	0	10	0	0	0
10	3	5	0	4	6	12	0	0	0
11	1	9	0	0	0	12	0	0	0
12	0	0	0	0	0	0	0	0	0

B Small instances

Time=36
Resource limit=25

Project	Profit
26	3741
21	4320
9	6201
68	3762
19	4560

Table 8: Small 1

Time=36
Resource limit=25

Project	Profit
82	4796
52	3735
28	4280
21	4095
32	5520

Table 9: Small 2

Time=36
Resource limit=25

Project	Profit
17	3838
32	4876
80	4956
90	4416
23	3960

Table 10: Small 3

Time=33
Resource limit=25

Project	Profit
49	2640
29	3649
37	5236
92	6148
31	3335

Table 11: Small 4

Time=31
Resource limit=25

Project	Profit
74	4180
86	2184
38	3128
2	3570
71	5782

Table 12: Small 5

Time=32
Resource limit=25

Project	Profit
79	3708
58	4080
39	4512
80	4704
60	2992

Table 13: Small 6

Time=37
Resource limit=25

Project	Profit
89	5778
52	3780
59	3045
36	3320
94	6102

Table 14: Small 7

Time=33
Resource limit=25

Project	Profit
49	2640
29	3649
37	5236
92	6148
31	3335

Table 15: Small 8

Time=36
Resource limit=25

Project	Profit
71	5047
5	4532
93	4280
81	4664
12	4324

Table 16: Small 9

Time=39
Resource limit=25

Project	Profit
24	3852
41	5040
40	5472
73	5520
91	4914

Table 17: Small 10

C Big instances

Time=64
Resource limit=25

Project	Profit
3	5424
53	2975
86	2712
63	3569
70	3737
46	3910
96	3552
2	5040
33	4897
72	4998

Table 18: Big 1

Time=67
Resource limit=25

Project	Profit
68	3382
52	3870
63	4042
58	3706
76	6615
98	4472
53	2775
93	4040
15	3560
10	5074

Table 19: Big 2

Time=69
Resource limit=25

Project	Profit
20	5160
58	4046
50	4255
52	4815
12	4002
23	3870
73	4272
35	3808
87	5148
7	3916

Table 20: Big 3

Time=68
Resource limit=25

Project	Profit
99	3848
79	3852
20	4300
5	3828
61	3552
44	3520
37	4708
16	3915
40	5187
85	3772

Table 21: Big 4

Time=68
Resource limit=25

Project	Profit
42	3705
63	4214
75	3420
45	5382
4	6000
59	3605
27	4800
72	3864
19	4200
46	4462

Table 22: Big 5

Time=64
Resource limit=25

Project	Profit
68	3192
51	5040
100	3161
79	3564
23	5175
58	3026
2	4116
42	4602
7	4092
22	4277

Table 23: Big 6

Time=65
Resource limit=25

Project	Profit
79	3384
74	4312
18	4860
66	3780
4	5000
61	4070
80	3528
53	2575
17	3914
15	4640

Table 24: Big 7

Time=61
Resource limit=25

Project	Profit
83	2407
22	4324
82	5236
53	2700
6	3399
72	4074
66	4815
96	3488
73	5280
24	3384

Table 25: Big 8

Time=66
Resource limit=25

Project	Profit
98	4343
36	4520
72	3612
61	4255
83	2349
14	5508
25	4802
57	5883
31	3480
0	3465

Table 26: Big 9

Time=62
Resource limit=25

Project	Profit
86	2136
97	5336
35	3264
30	3052
7	3564
2	4872
75	3168
57	5459
78	3800
26	4042

Table 27: Big 10

D Python code

Listing 1: Python implementation of VNS applied to PPSSP.

```
import pandas as pd
import numpy as np
import copy
import pulp as plp
import gurobipy as gb
import itertools
import random
import time as clock
import matplotlib.pyplot as plt
import matplotlib.patches as mpatch
Massivenumber=1000
Penaltyvalue=100000
stoppingvalue=5
maxlengthtabu=100

#######################################################################
#Data importation and some standard functions
#######################################################################
choice1=input("Small problem 0,.., 9: ")
problem2=pd.read_csv(r"C:\Users\sebas\Documents\TUDelft\Jaar 3\Bachelorproject\
    Data\problems\problem"+choice1+".txt",header= None)
aop=problem2.iloc[0,0].split()[0]          #amount_of_projects
aor=problem2.iloc[0,0].split()[1]          #amount_of_resources
aot=problem2.iloc[0,0].split()[2]          #amount_of_timeperiods
rc=problem2.iloc[1,0].split()              #resource_capacity
Materials=["R1","R2","R3","R4"]

problem=np.zeros((int(aop),2),int)
for i in range(2,int(aop)+2):
    a=problem2.iloc[i,0].split()
    for j in range(len(a)):
        problem[i-2][j]=a[j]
all_projects=problem[:,0]
profit=pd.DataFrame(problem,index=problem[:,0],columns=[0,"P"])
profit=profit.drop(0, 1)

Duration=np.zeros((int(aop),12),int)
it=0
for x in all_projects:
    project=np.zeros((12,10),int)
    projects=pd.read_csv(r"C:\Users\sebas\Documents\TUDelft\Jaar 3\
    Bachelorproject\Data\projects\Pat"+str(x)+".rcp",header=None)
    for i in range(2,14):
        b=projects.iloc[i,0].split()
        for j in range(len(b)):
            project[i-2][j]=b[j]
    project=pd.DataFrame(project,index=list(range(1,13)),columns=["Periods","R1"
    ,"R2","R3","R4","#N","N1","N2","N3","N4"])
    for z in range(1,13):
        Duration[it,z-1]=project.at[int(z),"Periods"]
    globals()['project%s' % x] = project
    it+=1

Periods=[0]*int(aot)
for i in range(0,int(aot)):
```

```
    Periods[i]=str(i+1)

Parts=[0]*10
for i in range(1,11):
    Parts[i-1]=str(i+1)

Parts2=[0]*11
for i in range(1,12):
    Parts2[i-1]=str(i)

Projects=[0]*int(aop)
for i in range(0,int(aop)):
    Projects[i]=str(all_projects[i])

Parts12=copy.copy(Parts)
Parts12.append("12")

def Next(a,part):
    b=globals()['project%s' % a].at[int(part),"#N"]
    c=[0]*b
    if b==1:
        c[0]=globals()['project%s' % a].at[int(part),"N1"]
    if b==2:
        c[0]=globals()['project%s' % a].at[int(part),"N1"]
        c[1]=globals()['project%s' % a].at[int(part),"N2"]
    if b==3:
        c[0]=globals()['project%s' % a].at[int(part),"N1"]
        c[1]=globals()['project%s' % a].at[int(part),"N2"]
        c[2]=globals()['project%s' % a].at[int(part),"N3"]
    if b==4:
        c[0]=globals()['project%s' % a].at[int(part),"N1"]
        c[1]=globals()['project%s' % a].at[int(part),"N2"]
        c[2]=globals()['project%s' % a].at[int(part),"N3"]
        c[3]=globals()['project%s' % a].at[int(part),"N4"]
    return c

def Previous(a,part):
    d=[]
    for p in Parts:
        if int(part) in Next(a,p):
            d.append(p)
    return d

def prev2(project,part):
    prev=[]
    for s in range(1,13):
        for n in ["N1","N2","N3","N4"]:
            if int(part)==globals()['project%s' % project].at[int(s),n]:
                prev.append(s)
    return prev

def completeprev(project,part,previ=[]):
    """Returns the all the parsts needed to complete before another part can
    start"""
    for p in prev2(project,part):
        previ.append(str(p))
        a=completeprev(project,p,previ)
        previ=Union(a,previ)
    return previ

```

```python
def completenext(project,part,nex=[]):
    for n in Next(project,part):
        nex.append(str(n))
        a=completenext(project,n,nex)
        nex=Union(a,nex)
    return nex

def Union(a,b):
    uni = list(set(a) | set(b))
    return uni

def difference(a,b):
    anw=[]
    for i in a:
        if i not in b:
            anw.append(i)
    return anw

##########################################################################
#Optimization
##########################################################################
def relaxiationfeasible(Selection):
    global x,Periodsadded
    """The normal scheduling program, only with continuous varaibles, we changed the duration of task 12 to 1, since this will make the program better programmable, to compensate, the amount of timerperiods is added by one."""
    Durationadded=copy.copy(Duration)
    for i in range(0,int(aop)):
        Durationadded[i,11]=1
    Periodsadded=copy.copy(Periods)
    Periodsadded.append(str(int(aot)+1))

    #VARIABLES
    model=plp.LpProblem("Scheduling problem",plp.LpMaximize)
    x=plp.LpVariable.dicts("x", (Selection,Periodsadded,Parts12),0,1,cat="Continous")

    #OBJECTIVE FUNCTION
    model+=plp.lpSum([x[i][t]["12"]*profit.at[int(i),"P"] for i in Selection for t in Periodsadded])

    #CONSTRAINTS
    for i in Selection:
        for s in Parts12:
            model+=plp.lpSum([x[i][t][s] for t in Periodsadded])==Durationadded[Projects.index(i),int(s)-1]
    for i in Selection:
        for s in Parts12:
            model+=plp.lpSum([x[i][t][s] for t in Periodsadded])==plp.lpSum([x[i][t]["12"] for t in Periodsadded])*Durationadded[Projects.index(i),int(s)-1]/float(Durationadded[Projects.index(i),11])
    for t in Periodsadded:
        for k in Materials:
            model+=plp.lpSum([globals()['project%s' % i].at[int(s),k]*x[i][t][s] for i in Selection for s in Parts])<=int(rc[Materials.index(k)])
    for M in range(1,len(Periodsadded)+1):
        for i in Selection:
            for s in Parts12:
                for z in Previous(i,s):
```

```
                    model+=plp.lpSum([x[i][str(t)][z] for t in range(1,M)])>=x[i][str(M)][s]*Durationadded[Projects.index(i),int(z)-1]
    model.solve(solver = plp.solvers.GUROBI(Mip=False,msg=False,timeLimit=120))
    if model.objective.value()==None:
        return 0
    else:
        return 1

def relaxiationfeasible2(Selection=Projects):
    global x,Periodsadded
    """The normal program, only with continuous varaibles, we changed the duration of task 12 to 1, since this will make the program better programmable, to compensate, the amount of timerperiods is added by one."""
    Durationadded=copy.copy(Duration)
    for i in range(0,int(aop)):
        Durationadded[i,11]=1
    Periodsadded=copy.copy(Periods)
    Periodsadded.append(str(int(aot)+1))

    #VARIABLES
    model=plp.LpProblem("Scheduling problem",plp.LpMaximize)
    x=plp.LpVariable.dicts("x", (Selection,Periodsadded,Parts12),0,1,cat="Continous")

    #OBJECTIVE FUNCTION
    model+=plp.lpSum([x[i][t]["12"]*profit.at[int(i),"P"] for i in Selection for t in Periodsadded])

    #CONSTRAINTS
    for i in Selection:
        for s in Parts12:
            model+=plp.lpSum([x[i][t][s] for t in Periodsadded])<=Durationadded[Projects.index(i),int(s)-1]
    for i in Selection:
        for s in Parts12:
            model+=plp.lpSum([x[i][t][s] for t in Periodsadded])==plp.lpSum([x[i][t]["12"] for t in Periodsadded])*Durationadded[Projects.index(i),int(s)-1]/float(Durationadded[Projects.index(i),11])
    for t in Periodsadded:
        for k in Materials:
            model+=plp.lpSum([globals()['project%s' % i].at[int(s),k]*x[i][t][s] for i in Selection for s in Parts])<=int(rc[Materials.index(k)])
    for M in range(1,len(Periodsadded)+1):
        for i in Selection:
            for s in Parts12:
                for z in Previous(i,s):
                    model+=plp.lpSum([x[i][str(t)][z] for t in range(1,M)])>=x[i][str(M)][s]*Durationadded[Projects.index(i),int(z)-1]
    model.solve(solver = plp.solvers.GUROBI(Mip=False,msg=False,timeLimit=120))
    if model.objective.value()==None:
        return 0
    else:
        sel=[]
        for i in Selection:
            if sum(x[i][t]["12"].varValue for t in Periods)>=0.5:
                sel.append(i)
        return sel

############################################################################
#Neighborhood selection
```

```python
###########################################################################
def insertgivenproj(selection,project):
    """Insert a given project"""
    selection.append(project)
    selection.sort()
    return selection

def removegivenproj(selection,project):
    """Removes a given project"""
    selection.remove(project)
    selection.sort()
    return selection

def swapgivenproj(selection,proja,projb):
    """Replaces selected project a for non selected project b"""
    selection2=removegivenproj(selection,proja)
    selection=insertgivenproj(selection2,projb)
    return selection

def getselection(al):
    """For a given activity list, the selection is returned"""
    selection=[]
    for i in range(len(al)):
        if al[i][0] not in selection:
            selection.append(al[i][0])
    return selection

###########################################################################
#Neighborhood schedule
###########################################################################
def Possibleswap(al,task):
    """Gives the tasks which can be swapped with given task"""
    aldummy=copy.copy(al)
    x1=al.index(task)
    lowerbound1=0
    upperbound1=len(al)-1
    for p in prev2(task[0],task[1]):
        if aldummy.index([task[0],str(p)])>=lowerbound1:
            lowerbound1=aldummy.index([task[0],str(p)])
    for n in Next(task[0],task[1]):
        if aldummy.index([task[0],str(n)])<=upperbound1:
            upperbound1=aldummy.index([task[0],str(n)])
    possibleswaps=[]
    if upperbound1-lowerbound1<=2:
        return []
    for i in range(lowerbound1+1,upperbound1):
        possibleswaps.append(aldummy[i])
    possibleswaps2=copy.copy(possibleswaps)
    for ps in possibleswaps2:
        lowerbound2=0
        upperbound2=len(al)-1
        for p in prev2(ps[0],ps[1]):
            if aldummy.index([ps[0],str(p)])>=lowerbound2:
                lowerbound2=aldummy.index([ps[0],str(p)])
        for n in Next(ps[0],ps[1]):
            if aldummy.index([ps[0],str(n)])<=upperbound2:
                upperbound2=aldummy.index([ps[0],str(n)])
        if lowerbound2>=x1 or x1>=upperbound2:
            possibleswaps.remove(ps)
    return possibleswaps
```

```
def Swapping(al,taska,taskb):
    """Swaps two given tasks"""
    aldummy=copy.copy(al)
    indexa=al.index(taska)
    indexb=al.index(taskb)
    aldummy[indexa]=al[indexb]
    aldummy[indexb]=al[indexa]
    return aldummy

def Possiblereschedules(al,task):
    """Gives the indices where task can be placed"""
    aldummy=copy.copy(al)
    lowerbound=0
    upperbound=len(al)-1
    for p in prev2(task[0],task[1]):
        if aldummy.index([task[0],str(p)])>=lowerbound:
            lowerbound=aldummy.index([task[0],str(p)])
    for n in Next(task[0],task[1]):
        if aldummy.index([task[0],str(n)])<=upperbound:
            upperbound=aldummy.index([task[0],str(n)])
    lowerindex=lowerbound+1
    upperindex=upperbound-1
    return lowerindex,upperindex

def Rescheduling(al,task,index):
    """Reschedules a task to a given index"""
    aldummy=copy.copy(al)
    aldummy.remove(task)
    aldummy.insert(index,task)
    return aldummy

def Allpossible(al):
    global task
    All=[]
    for task in al:
        pswaps=Possibleswap(al,task)
        for pswap in pswaps:
            All.append([0,task,pswap])
    for task in al:
        lower,upper=Possiblereschedules(al,task)
        for i in range(lower,upper+1):
            All.append([1,task,i])
    return All

def Takenumberofneighboral(al,maxneigh):
    aldummy=copy.copy(al)
    tabual=[al]
    Neighbors=[]
    All=Allpossible(al)
    random.shuffle(All)
    for change in All:
        al=copy.copy(aldummy)
        if change[0]==0:
            Neig=Swapping(al,change[1],change[2])
            if Neig not in Neighbors:
                Neighbors.append(Neig)
        if change[0]==1:
            Neig=Rescheduling(al,change[1],change[2])
            if Neig not in Neighbors:
```

```
                Neighbors.append(Neig)
        if len(Neighbors)==maxneigh:
            return Neighbors
    return Neighbors

############################################################################
#Preemptive serial schedule generation scheme
############################################################################
def Planning(al,last=0,unlimited=False):
    """Gives a schedule for a given activity list, preemption is allowed"""
    global Rkt,som,a,plan,d,time,t3,leftlimit,project,rectangles
    selection=getselection(al)
    if last==1:
        rectangles1={}
        rectangles2={}
        rectangles3={}
        rectangles4={}
        fig, ax=plt.subplots()
        fig2, ax2=plt.subplots()
        fig3, ax3=plt.subplots()
        fig4, ax4=plt.subplots()
    plan={}
    pen=0
    Rkt=np.full((int(aor),Massivenumber+1),int(rc[0]))
    if unlimited==True:
        Rkt=np.full((int(aor),Massivenumber+1),Massivenumber)
    for l in range(len(al)):
        project=al[l][0]
        task=al[l][1]
        r1=globals()['project%s' % project].at[int(task),"R1"]
        r2=globals()['project%s' % project].at[int(task),"R2"]
        r3=globals()['project%s' % project].at[int(task),"R3"]
        r4=globals()['project%s' % project].at[int(task),"R4"]
        d=Duration[Projects.index(project),int(task)-1]
        if str(task)=="1":
            plan[(project,task,0)]=0
        tillp=[0]*max(1,len(prev2(project,task)))
        som=0
        durp=[0]*len(prev2(project,task))
        for p in prev2(project,task):
            durp[som]=Duration[Projects.index(project),int(p)-1]
            tillp[som]=plan[(project,str(p),durp[som])]
            som+=1
        leftlimit=max(tillp)
        time=0
        while time<d:
            for t in range(leftlimit+1,Massivenumber):
                if r1<=Rkt[0][t-1] and r2<=Rkt[1][t-1] and r3<=Rkt[2][t-1] and
    r4<=Rkt[3][t-1]:
                    time+=1
                    #############
                    if Projects.index(project)==0:
                        col="yellow"
                    if Projects.index(project)==1:
                        col="blue"
                    if Projects.index(project)==2:
                        col="green"
                    if Projects.index(project)==3:
                        col="red"
                    if Projects.index(project)==4:
```

```
                        col="pink"
                    if Projects.index(project)==5:
                        col="brown"
                    if Projects.index(project)==6:
                        col="purple"
                    if Projects.index(project)==7:
                        col="black"
                    if Projects.index(project)==8:
                        col="orange"
                    if Projects.index(project)==9:
                        col="cyan"
                    if last==1:
                        if r1!=0:
                            rectangles1[str(project)+"-"+str(task)+"-"+str(time)]=mpatch.Rectangle((t-1,int(rc[Materials.index("R1")])-Rkt[0][t-1]),1,r1,fc=col,ec="black")
                        if r2!=0:
                            rectangles2[str(project)+"-"+str(task)+"-"+str(time)]=mpatch.Rectangle((t-1,int(rc[Materials.index("R2")])-Rkt[1][t-1]),1,r2,fc=col,ec="black")
                        if r3!=0:
                            rectangles3[str(project)+"-"+str(task)+"-"+str(time)]=mpatch.Rectangle((t-1,int(rc[Materials.index("R3")])-Rkt[2][t-1]),1,r3,fc=col,ec="black")
                        if r4!=0:
                            rectangles4[str(project)+"-"+str(task)+"-"+str(time)]=mpatch.Rectangle((t-1,int(rc[Materials.index("R4")])-Rkt[3][t-1]),1,r4,fc=col,ec="black")
                    ##############
                    Rkt[0][t-1]-=r1
                    Rkt[1][t-1]-=r2
                    Rkt[2][t-1]-=r3
                    Rkt[3][t-1]-=r4
                    plan[(project,task,time)]=t
                    leftlimit=t
                    break
    selection=getselection(al)
    lasttime=[0]*len(selection)
    for i in selection:
        for s in Parts:
            for tijd in range(1,Duration[Projects.index(i),int(s)-1]+1):
                if plan[(i,s,tijd)]>lasttime[selection.index(i)]:
                    lasttime[selection.index(i)]=plan[(i,s,tijd)]
        plan[(i,"12",0)]=lasttime[selection.index(i)]
        pen+=max(plan[(i,"12",0)]-int(aot),0)*Penaltyvalue
    maxtime2=max(plan,key=plan.get)
    maxtime=plan[maxtime2]
    ################
    if last==1:
        yellow = mpatch.Patch(color='yellow', label='Project' + str(Projects[0]))
        blue = mpatch.Patch(color='blue', label='Project' + str(Projects[1]))
        green = mpatch.Patch(color='green', label='Project' + str(Projects[2]))
        red = mpatch.Patch(color='red', label='Project' + str(Projects[3]))
        pink = mpatch.Patch(color='pink', label='Project' + str(Projects[4]))
        hand=[yellow,blue,green,red,pink]
        if len(Projects)==10:
            brown = mpatch.Patch(color='brown', label='Project' + str(Projects[5]))
```

```
            purple = mpatch.Patch(color='purple', label='Project ' + str(Projects[6]))
            black = mpatch.Patch(color='black', label='Project ' + str(Projects[7]))
            orange = mpatch.Patch(color='orange', label='Project ' + str(Projects[8]))
            cyan = mpatch.Patch(color='cyan', label='Project ' + str(Projects[9]))
            hand=hand+[brown,purple,black,orange,cyan]
        for r in rectangles1:
            dum=r.split("-")
            ax.add_artist(rectangles1[r])
            rx, ry = rectangles1[r].get_xy()
            cx = rx + rectangles1[r].get_width()/2.0
            cy = ry + rectangles1[r].get_height()/2.0
            ax.annotate(dum[1], (cx, cy), color='w', weight='bold', fontsize=6, ha='center', va='center')
        ax.set_xlim((0, int(aot)))
        ax.set_ylim((0, int(rc[0])))
        ax.set_xlabel('Time')
        ax.set_ylabel('Resources')
        ax.set_title('Resource 1')
        ax.legend(loc='center left', bbox_to_anchor=(1, 0.5),handles=hand)
        ax.plot()
        for r in rectangles2:
            dum=r.split("-")
            ax2.add_artist(rectangles2[r])
            rx, ry = rectangles2[r].get_xy()
            cx = rx + rectangles2[r].get_width()/2.0
            cy = ry + rectangles2[r].get_height()/2.0
            ax2.annotate(dum[1], (cx, cy), color='w', weight='bold', fontsize=6, ha='center', va='center')
        ax2.set_xlim((0, int(aot)))
        ax2.set_ylim((0, int(rc[1])))
        ax2.set_xlabel('Time')
        ax2.set_ylabel('Resources')
        ax2.set_title('Resource 2')
        ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5),handles=hand)
        ax2.plot()
        for r in rectangles3:
            dum=r.split("-")
            ax3.add_artist(rectangles3[r])
            rx, ry = rectangles3[r].get_xy()
            cx = rx + rectangles3[r].get_width()/2.0
            cy = ry + rectangles3[r].get_height()/2.0
            ax3.annotate(dum[1], (cx, cy), color='w', weight='bold', fontsize=6, ha='center', va='center')
        ax3.set_xlim((0, int(aot)))
        ax3.set_ylim((0, int(rc[2])))
        ax3.set_xlabel('Time')
        ax3.set_ylabel('Resources')
        ax3.set_title('Resource 3')
        ax3.legend(loc='center left', bbox_to_anchor=(1, 0.5),handles=hand)
        ax3.plot()
        for r in rectangles4:
            dum=r.split("-")
            ax4.add_artist(rectangles4[r])
            rx, ry = rectangles4[r].get_xy()
            cx = rx + rectangles4[r].get_width()/2.0
            cy = ry + rectangles4[r].get_height()/2.0
```

```
            ax4.annotate(dum[1], (cx, cy), color='w', weight='bold', fontsize=6,
    ha='center', va='center')
        ax4.set_xlim((0, int(aot)))
        ax4.set_ylim((0, int(rc[3])))
        ax4.set_xlabel('Time')
        ax4.set_ylabel('Resources')
        ax4.set_title('Resource 4')
        ax4.legend(loc='center left', bbox_to_anchor=(1, 0.5),handles=hand)
        ax4.plot()
        plt.show()
    ################
    return plan,fitness(selection,plan),pen

############################################################################
#Initial activity list
############################################################################
def makeal(selection):
    """Makes a random activity list statisfying the precedence constraints"""
    global postask,lst
    lst=[]
    postask=[]
    for i in selection:
        lst.append([i,"1"])
    for i in selection:
        for n in Next(i,"1"):
            postask.append([i,str(n)])
    while postask!=[]:
        tas=random.choice(postask)
        pro=tas[0]
        lst.append(tas)
        for n in Next(pro,tas[1]):
            som=0
            for p in prev2(pro,n):
                if [pro,str(p)] in lst:
                    som+=1
                if som==len(prev2(pro,n)):
                    postask.append([pro,str(n)])
        postask.remove(tas)
    return lst

def checkalmp(al):
    """Checks wether an activity lists fulfit the presendence constraints"""
    numbpro=int(len(al)/12)
    selection=[]
    for i in range(numbpro):
        selection.append(al[i][0])
    for i in selection:
        for s in Parts:
            for n in Next(i,s):
                if al.index([i,str(n)])<al.index([i,s]):
                    return 0
    return 1

############################################################################
#Fitness functions
############################################################################
def fitness(selection,planning):
    """For this case, time independent profit"""
    return sum(profit.at[int(i),"P"] for i in selection)
```

```
maxfit=[0]*len(Projects)
for i in range(len(Projects)):
    projec=[Projects[i]]
    extra1,extra2,extra3=Planning(makeal(projec),unlimited=True)
    extra1=0
    maxfit[i]=extra2-extra3

def maxfitness(selection):
    """Fitness if unlimted resources"""
    ma=0
    for i in selection:
        ma+=maxfit[Projects.index(i)]
    return ma

############################################################################
#Initial selection
############################################################################
def bestinitialselection():
    a=relaxiationfeasible2()
    a.sort()
    return a

############################################################################
#Search for best schedule
############################################################################
def searchbestscheduletabu(alist,numberofneighbors):
    """
    al1 = firts al
    al2= The one we make neighbor of, so the current one
    al3= Best of the 50 neighbors, not in tabulist
    almax= Best al so far
    alfeas= Best feas al so far
    """
    global neighbors,tabual,sortedprofit,maxsearchneigh,neighboral,ran,al2,
    alistrem
    schema1,winst1,pen1=Planning(alist)
    al1=alist
    tabual=[alist]
    schema2=schema1
    winst2=winst1-pen1
    al2=al1
    winstmax=winst2
    almax=al1
    winstfeas=-99999
    alfeas=[]
    winst3=-99999
    al3=[]
    if pen1==0:
        alfeas=al1
        winstfeas=winst1
    winstsame=0
    it=0
    while winstsame <= 4:      #If the maxprofit is the same for 5 iterations,
    the program stops, so 5*numberofneighbors neighbors have smaller or equal
    profit
        alistrem=copy.copy(al2)
        neighbors={}
        profitneigh={}
        it+=1
        winstsame+=1
```

```python
        print("iteration: ",it,winstsame)
        Neighbor=Takenumberofneighboral(al2,numberofneighbors)
        for neighboral in Neighbor:
            scheman,winstn,penn=Planning(neighboral)
            profitneigh[winstn-penn]=(neighboral,penn)
        sortedprofit=list(profitneigh.keys())
        sortedprofit.sort(reverse=True)
        for prof in sortedprofit:
            if profitneigh[prof][0] not in tabual:
                winst3=prof
                al3=copy.copy(profitneigh[prof][0])
                if profitneigh[prof][1]==0:
                    if prof>winstfeas:
                        winstfeas=prof
                        alfeas=copy.copy(profitneigh[prof][0])
                break
        if winst3>winstmax:
            winstmax=winst3
            almax=copy.copy(al3)
            winstsame=0
        winst2=winst3
        al2=copy.copy(al3)
        tabual.append(al2)
    return winstfeas,alfeas,winstmax,almax

###########################################################################
#Search for best selection neighbor
###########################################################################
def selectionneighbortabu2(initialselection,select,tabudifference):
    """Gives a selection allowed by the tabucriterion"""
    global select3,selectiondummy
    allsel=[]
    allsel2=[]
    selectiondummy=copy.copy(select)
    for i in selectiondummy:
        for j in difference(Projects,select):
            select2=insertgivenproj(select,j)
            allsel.append(select2)
            select=copy.copy(selectiondummy)
            select3=swapgivenproj(select,i,j)
            allsel.append(select3)
            select=copy.copy(selectiondummy)
        if len(select)!=1:
            select4=removegivenproj(select,i)
            allsel.append(select4)
            select=copy.copy(selectiondummy)
    for sl in allsel:
        difference1=difference(initialselection,sl)
        difference2=difference(sl,initialselection)
        comdifference=difference1+difference2
        comdifference.sort()
        if comdifference not in tabudifference:
            allsel2.append(sl)
    return allsel2

def bestprofitableselectionneighbortabu2(initialselection,select,tabudifference)
    :
    """Gives the most profitable allowed neighboring selection"""
    win=-9999999
    se=[]
```

```
    selectr=copy.copy(select)
    alls=selectionneighbortabu2(initialselection,select,tabudifference)
    for se2 in alls:
        win2=maxfitness(se2)
        if win2>win:
            se=se2
            win=win2
    select=copy.copy(selectr)
    se.sort()
    return se

###########################################################################
#Variable neighborhood search
###########################################################################
def vns(numberofneighborsschedule,numberofneighborsselection=1000,intselec=
    bestinitialselection()):
    global winstmax,tabudifference
    begintime=clock.time()
    winstmax=-99999
    selmax=[]
    almax=0
    winstfeasible1=-99999
    alfeasible1=[]
    selfeasible1=[]
    tabudifference=[[]]
    intselec.sort()
    selection=copy.copy(intselec)
    nochangeprofit=0
    numberneighbors=0
    while nochangeprofit<=49:
        selectiondum=copy.copy(selection)
        print(selection,nochangeprofit)
        if relaxiationfeasible(selection)==1:
            nochangeprofit+=1
            if maxfitness(selection)>=winstmax:
                alijst=makeal(selection)
                winstfeasible2,alfeasible2,winst,al=searchbestscheduletabu(
    alijst,numberofneighborsschedule)
                if winst>winstmax:
                    nochangeprofit=0
                    winstmax=winst
                    almax=copy.copy(al)
                    selmax=copy.copy(selection)
                if winstfeasible2>winstfeasible1:
                    winstfeasible1=winstfeasible2
                    alfeasible1=copy.copy(alfeasible2)
                    selfeasible1=copy.copy(selection)
        if numberneighbors==numberofneighborsselection:
            break
        selection=copy.copy(selectiondum)
        neighborsel=bestprofitableselectionneighbortabu2(intselec,selection,
    tabudifference)
        if neighborsel==[]:
            endtime=clock.time()
            fulltime=endtime-begintime
            file=open("Bestalproblem_small_v2_"+choice1,"w")
            file.write(str(intselec)+","+str(numberofneighborsschedule)+","+str(
    winstmax)+","+str(fulltime)+","+str(almax)+","+str(selmax)+","+str(
    winstfeasible1)+","+str(alfeasible1)+","+str(selfeasible1)+","+"No more
    neighbors")
```

```
            file.close()
            print("No more valid neighbours")
            return winstmax,almax,selmax,winstfeasible1,alfeasible1,selfeasible1,fulltime
        difference1=difference(intselec,neighborsel)
        difference2=difference(neighborsel,intselec)
        differencecom=difference1+difference2
        differencecom.sort()
        tabudifference.append(differencecom)
        if len(tabudifference)>maxlengthtabu:
            tabudifference.remove(tabudifference[0])
        neighborsel.sort()
        selection=copy.copy(neighborsel)
        numberneighbors+=1
    endtime=clock.time()
    fulltime=endtime-begintime
    file=open("Bestalproblem"+choice1,"w")
    file.write(str(intselec)+","+str(numberofneighborsschedule)+","+str(winstmax)+","+str(almax)+","+str(selmax)+","+str(fulltime)+","+str(winstfeasible1)+","+str(alfeasible1)+","+str(selfeasible1))
    file.close()
    return winstmax,almax,selmax,winstfeasible1,alfeasible1,selfeasible1,fulltime

def getresults(numberofneighborsschedule):
    global remove,j
    a,b,c,d,e,f,g=vns(numberofneighborsschedule)
    plannetje=Planning(b)
    if d!=0:
        return d,e,f,g
    remove={}
    for i in range(1,len(c)+1):
        for j in list(itertools.combinations(c,i)):
            remove[fitness(list(j),plannetje)]=list(j)
    keys=list(remove.keys())
    keys.sort(reverse=True)
    for k in keys:
        ali=makeal(remove[k])
        h,l,m,n,o,p,q=vns(400,0,remove[k])
        if n!=0:
            return n,o,p,g
    return 0,[],[]
```