Final Report: Optimore

Isak Bohman
MAI, LiU

Victor Bergelin
MAI, LiU

Akdas Hossain
MAI, LiU

Emelie Karlsson
MAI, LiU

Claes Arvidson
MAI, LiU

Jonathan Andersson
MAI, LiU

Hendric Kjellström
MAI, LiU

January 18, 2016

Status

Controlled	2015-12-22	Torbjörn Larsson
Accepted		

Document History

Version	Date	Changes	Made by	Supervised
0.1	2015-12-11	First draft	All	IB
0.2	2016-01-12	Second draft	All	EK
0.3	2016-01-13	Third draft	All	EK

Project Identity

TATA62, CDIO project 2015, Optimore
MAI, Linköpings Tekniska Högskola

Name	Responsibility	Phone number	Mail
Akdas Hossain	2nd Document manager	073-784 12 58	akdho545@student.liu.se
Claes Arvidson	Requirement manager	070-345 73 04	claar324@student.liu.se
Emelie Karlsson	Project manager	070-519 48 20	emeka813@student.liu.se
Hendric Kjellström	GUI manager	073-671 27 84	henkj080@student.liu.se
Isak Bohman	Document manager	076-209 40 35	isabo487@student.liu.se
Jonathan Andersson	Test manager	070-932 65 33	jonan860@student.liu.se
Victor Bergelin	Design manager	070-826 34 53	vicbe348@student.liu.se

Customer: Elina Rönnberg, Linköping University 581 83 LINKÖPING
Customer phone number: 013-28 16 45
Eximiner: Danyo Danev, phone number 013-281335, e-mail address danyo.danev@liu.se
Supervisor: Torbjörn Larsson, phone number 013-282435, e-mail address torbjörn.larsson@liu.se

Contents

1 Introduction **1**

2 Problem description **1**

3 System description **2**

4 Description of data **2**

5 Mathematical model **6**

6 Analysis of Tabu Search **7**
6.1 The tabu search heuristic . . . 7
6.2 Basic implementation . . . 7
6.3 Test philosophy . . . 8
6.4 Test data used . . . 8
6.5 Parameter model 1 (M1): Varying neighbourhood and phase . . . 8
6.5.1 Variation of parameters . . . 9
6.5.2 Parameters in models . . . 9
6.5.3 Result . . . 10
6.5.4 Discussion . . . 10
6.6 Parameter model 2 (M2): Varying tabu list construction . . . 11
6.6.1 Testing model . . . 11
6.6.2 Result . . . 11
6.6.3 Discussion . . . 12
6.7 Parameter model 3 (M3): Varying tabu list length . . . 12
6.7.1 Testing model . . . 12
6.7.2 Result . . . 13
6.7.3 Discussion . . . 13
6.8 Parameter model 4 (M4): Varying phase stopping criteria . . . 14
6.8.1 Stopping criteria models . . . 14
6.8.2 Result . . . 14
6.8.3 Discussion . . . 15
6.9 Parameter model 5 (M5): Varying dynamic weights . . . 15
6.9.1 Weight models . . . 15
6.9.2 Results . . . 16
6.9.3 Discussion . . . 16
6.10 Final realizations . . . 17
6.10.1 Combining the models . . . 17
6.10.2 Results . . . 17
6.10.3 Discussion . . . 18
6.11 Further development . . . 18

7 Analysis of LNS **19**
7.1 The LNS search heuristic . . . 19
7.2 Destroy function . . . 20
7.2.1 Random destroy . . . 20
7.2.2 Active destroy . . . 20
7.3 Results and analysis . . . 20
7.4 Further development . . . 21

8 Comparison between heuristics **22**

9 Conclusion **23**

Appendices **25**

A Mathematical model **25**

1 Introduction

In this project, we have addressed a large scale scheduling problem for avionics using two very different heuristic methods. The problem concerned scheduling of tasks distributed over avionic hardware modules where the tasks were to be placed within certain intervals and some had a fixed relative order. The first heuristic tested, tabu search, was chosen as a promising method, since it is independent of commercial solvers such as CPLEX and since it uses an adpative memory structure which avoids poor moves. The second heuristic was Large Neighbourhood Search, or LNS, which was chosen as it extends the CPLEX solver in order for it to work better on a specific problem. The project turned out to be a challenging programming task where we encountered many difficulties in parameter values and in the overall formulation of the problem.

The main challenge for our group was that we had to design the system from scratch. We had to create a data generator with little information about the data. We also had to design a system which could launch two heuristic methods for large data sets as well as the two algorithms themselves. In addition to this, we also equiped the launcher with tools such as plots and statistics in order to evaluate the performace of the algorithms.

2 Problem description

As part of a CDIO project course at Linköping University, the group was given the problem of scheduling avionics tasks on hardware modules in aeroplanes. Such tasks could consist of reading of sensor data, processing of data or likewise. The tasks were to be placed within a given interval and dependencies could exist between tasks so that some tasks have to be placed before others. The hardware modules are modeled as discrete "timelines" with length one second. We were not given specific information about the data such as the number of tasks to schedule during this time and the duration of these tasks. However, the following guidlines were given to strive towards:

- A timeline has 10^9 discrete time points.
- A maximum of 10^4 tasks could exist on a timeline.
- Every task has an earliest possible starting time, a latest possible ending time and a fixed length.
- Between tasks, there can be dependecies that say that one task has to follow upon another and that the distance between the tasks should be within a certain interval.
- There could be chains of dependencies that forces many tasks to be in a specific order.
- Heuristic methods should be used to find a feasible solution to the problem, that is, a solution where all tasks are placed on the timelines with all constraints satisfied.

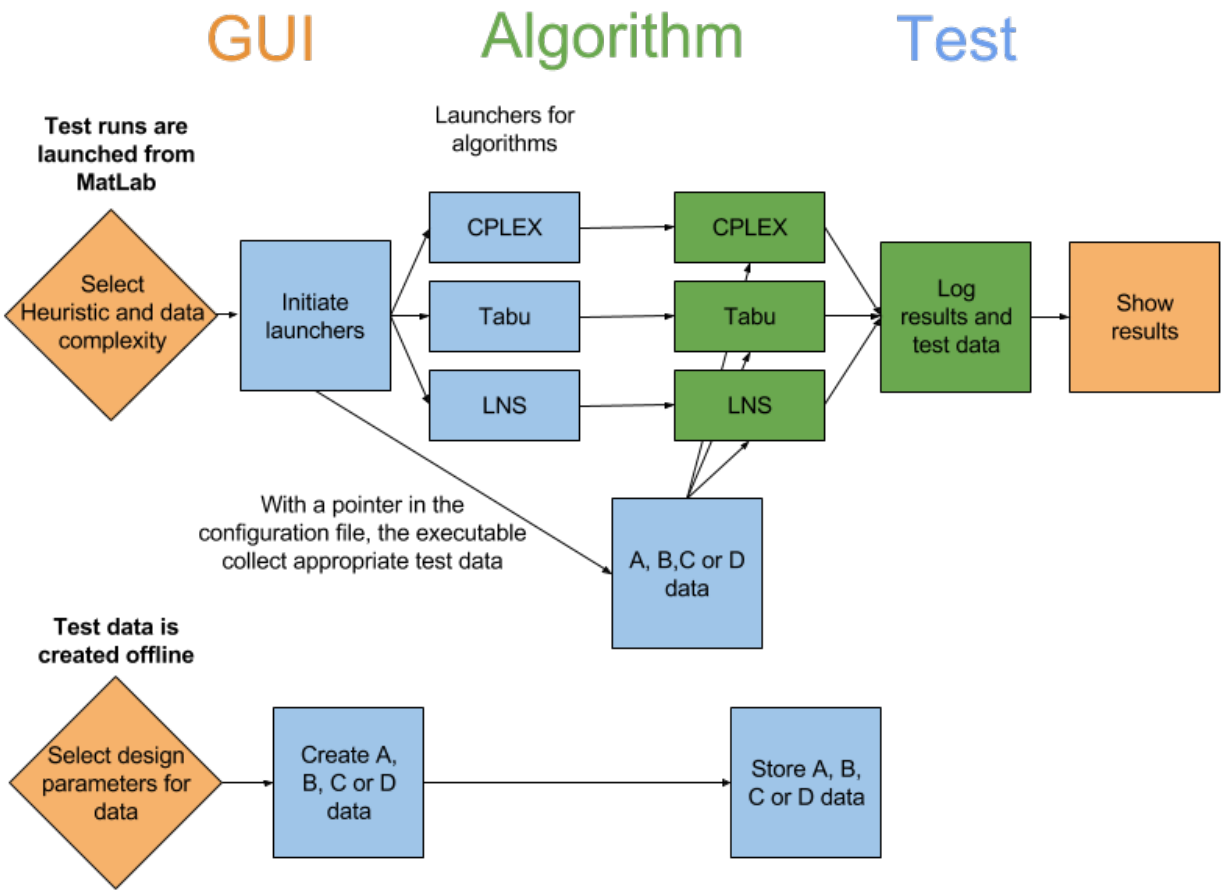
The suggested strategies for solving this problem was to use one of two heuristic optimization methods; tabu search and Large Neighbourhood Search (LNS). In order to evaluate these heuristics, we needed to create data according to the above instructions. No specific constraints were given on this data and is was part of our challenge to design appropriate data for the problem. An example of how such data could look like is illustrated in Figure 3 and the data is described more thoroughly in Section 4.

This project took place at Linköping University in the autumn 2015. Seven fifth-year students studying Teknisk Fysik och Elektroteknik with the mathematical profile formed the project group.

3 System description

The system designed basically consists of three parts: Grafical user interface (GUI), launcher scripts and the algorithms, in form of the implementation of the chosen heuristics. Data is created using the data creation interface seen at the bottom of the illustration in Figure 1. With test data available, the launcher GUI is used to run tests. The process is illustrated on the top row of Figure 1. After the test has been initiated, the user can analyze the results using the result interface at the top right of the figure. All code was developed in MATLAB, except for the LNS-solver which was written in AMPL and laucnhed with Linux shell scripts.

Figure 1: An overview illustration of the system



4 Description of data

Data were created using a specialized GUI, built for this very purpose. The user is able to specify parameter values using which data sets are subsequently created. When creating data, a feasible solution to the problem is guaranteed. Multiple data sets were created at once. For this project, four different types of data

settings were decided upon. In addition to these settings, a general setting of complexity was devised, which determines the number of tasks and dependencies that are created for that data set.

For all data types, approximately 25% of the admissible intervals of dependencies and tasks (the intervals which constrain the times at which dependencies and tasks may be placed) were generated using a distribution with a mean as the order of the length of the timeline, approximately 25% of intervals were generated using a distribution with a mean on the order of the length of the each task. The remaining 50% of interval lengths were generated using a distribution with an expected value equal to the geometric average of the expected values of lengths of the short and the long intervals. The distribution of choice for these intervals was a folded normal distribution, in order to avoid negative values.
Tasks are distributed across timelines randomly, according to a folded normal distribution.
Task lengths and the spacing between tasks were generated from χ^2 distributions, thereby automatically avoiding the problem of negative values. Chains of dependencies have been implemented in the test data generator, although this feature has not been utilized in the creation of the data sets upon which tests were carried out.

Overview of Data Types

Data Type	Description
A	Standard data. For data of this type, the number of timelines is at most 5, the level of occupancy for all timelines is approximately 40%, and the total number of dependencies is twice the total number of tasks. Dependencies are randomly distributed, but sufficiently evenly distributed as to guarantee that every task partakes in at least one dependency.
B	High dependency data. For data of this type, the total number of dependencies is approximately four times the total number of tasks, but other data parameter values are identical to those for the standard data.
C	High occupancy level data. For data of this type, timelines are approximately 70% covered by tasks.
D	Many timelines data. For data of this type, the number of timelines is 30 for sufficiently large data sets.

Data complexity is specified as a number from 0 to 100, where 0 corresponds to the smallest data sets and 100 corresponds to the largest data sets. In this project, however, only level 60 data were generated, as no larger data sets can be solved sufficiently quickly. For reference, type 0 data have approximately 20 tasks, while level 60 data have approximately 1,300. Level 100 data sets contain approximately 10,000 tasks.
A number of visualizations for data sets were created to assess the feasibility of the data sets which have been created. Figure 3 displays all tasks in a data set in an organized way, where the y position of a task displays the timeline it belongs to, with the x axis being time. Tasks are illustrated using boxes for the sake of convenience.
Figure 2 is identical to Figure 3 in every respect except for the fact that it also illustrates dependencies between tasks. Second-degree splines are used to illustrate dependencies between tasks. Red dots indicate the starting times of the second task in a dependency (so one may ascertain that the second task in a

dependency starts after the first one has been completed).

Figure 2: Visualization of a typical data set. The rounded boxes indicate tasks.

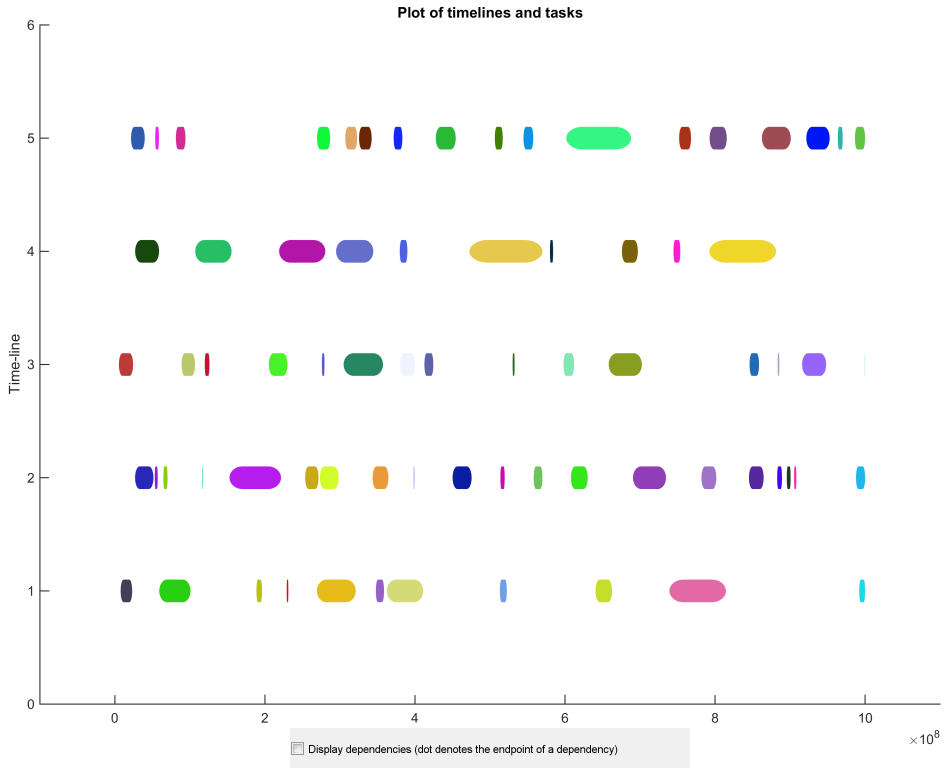


Figure 3: Visualization of a typical data set. Dependencies between tasks in an admissible solution are displayed using splines. Dependencies are evenly distributed across tasks, and longer tasks partake in more dependencies than short ones.

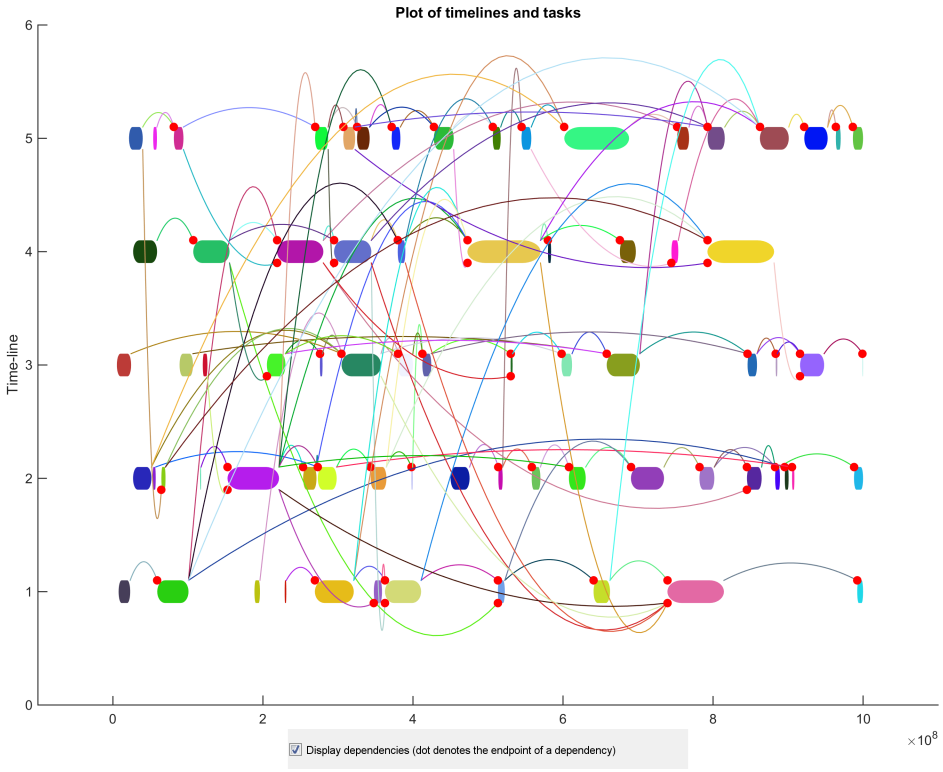
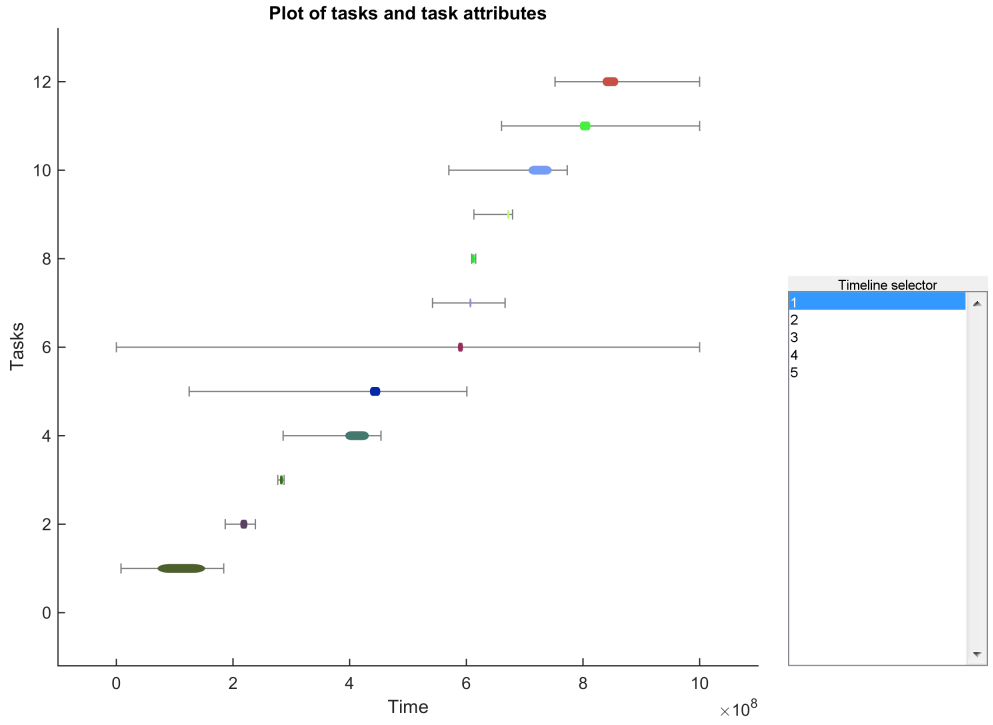


Figure 4: Visualization of a typical data set. Here, all tasks on a timeline are displayed, and the corresponding admissible intervals of placements. The pattern of certain tasks having much longer admissible intervals is clearly visible.



5 Mathematical model

The group received a mathematical model for the scheduling problem under consideration from the customer, see Appendix A. The group decided to add artificial variables, $a_1 - a_5$ in the constraints as a way to measure the violations of conditions (6) and (8) in the appendix. A feasible solution is found when a set of y-variables, which describe in what order the tasks are scheduled, are found while, at the same time, the artificial variables are zero. No objective function was given to the group, therefore the group had to make its own. The objective function for the problem minimizes the sum of the artificial variables, weighted with a value proportional to how much of the total violation it contributes to.

The same mathematical model was used both when solving with and without LNS added to the mathematical model. The difference is how the group has programmed the different run files in AMPL. Without the heuristic, CPLEX tries to solve the problem at once with a single solve command whereas for the LNS heuristic the run file is programmed in a manner which breaks down the problem, which allows CPLEX to solve only a part of the problem instead of trying to solve ethe whole problem at once.

6 Analysis of Tabu Search

6.1 The tabu search heuristic

Tabu search is a heuristic method developed in the 1980's in order to solve optimization problems by steering a search process in such a way as to not get trapped in local optima [1]. The heuristic chooses the cheapest action from a set of possible actions, which is called the neighbourhood of the search. The neighbourhood can be defined arbitrarily, but is usually variations of the current solution. Actions determine how the solution moves in the neighbourhood. Before performing an action, the heuristic checks if the chosen action is in the tabu list. If so, it will not be selected and the second most suitable action is evaluated.

The tabu list can be designed in different ways, but its main purpose is to avoid recently visited solutions. If an action is in the tabu list, the next best action will be chosen. If this action is also in the tabu list, the next next best action will be chosen and thus the algorithm is guiding the search out of a local optimum. Tabu search generally moves in phases of diversification and intensification. The diversification process involves moving away from a local optimum while the intensification process performs a deep-search in a promising area.

Tabu searchis especially useful when searching for a good, but not certainly optimal solution. As tabu search can be implemented in any imaginable formforms, this project focuses on evaluating a brief set of variations.

6.2 Basic implementation

The tabu solver designed during the project follows the general pattern for a tabu serach with an action defined as the movement of a task a number of time steps. Several parameter models containing small adaptations of the basic implementation were developed during the project as is explained in the next sections. However, the basic implementation follows the following algorithm:

1. A data file with information about tasks and their properties is read by the program.
2. An initial solution is created where the tasks are placed at the mid point between their earliest starting time and latest ending time.
3. Three different costs are calculated for the solution. The first one for overlapping tasks, the second one for tasks that are outside of their interval and the third one for violations of dependencies.
4. If the total cost is zero or if a maximum number of iterations is reached, the search is finished.
5. If the total cost is larger than zero and the maximum number of iterations is not reached, the first phase is entered.
6. For all tasks on all timelines, each task is moved a few time steps on the timeline. This creates a set of possible new solutions.
7. Costs for all possible solutions in the set are found and sorted. The cheapest one is selected.
8. If the selected solution is not in the tabu list or is in the tabu list but the cheapest solution so far, it is set as the new solution and the process is repeated from step 3.

9. If the solution is in the tabu list and is not the cheapest solution so far, the next solution in the set of possible solutions is chosen. The process is then repeated from step 8.

10. If, during steps 5 and 9, choosing the best possible move results in a deterioration of the cost function several iterations in a row the next phase is entered. If the solver is already in the last phase, the first phase is entered again.

The implementation consists of two or more phases. The first phase contain moves with longer time step lengths and corresponded to the diversification process described above. The other phases only provide shorter steps, meaning that the process was directed towards the local optimum thus providing intensification. A switch from one phase to another is induced by a few subsequent moves leading to worse results. Thus, the search moved from a local optimum by switching to the diversification phase if we could find no local improvements and vice versa for the intensification process.

In order to guide the search further, the realization contained adaptive weights controlling the three different costs. The adaptive weights were updated with a spacing of a few iterations, in such a way that larger costs were penalized more according to the formula $weight_i = \frac{cost_i}{TotalCost}$, where i is one of the three types of costs.

6.3 Test philosophy

The implementation described above was developed as a stable basic model with fairly good performance, as was concluded after some testing and discussing with our supervisor. Using this for an initial evaluation, several interesting parameters were found for further investigation. The parameters that seemed most interesting include: the number of phases used, variations of tabu list representation and length, variation of number of bad iterations before changing phases, and the number of iterations before updating the cost weights. These are presented in the following sections. Using the statistical information obtained through these tests, we then designed four parameter models, each adapted to one of the four data types, thus creating an A, B, C and D tabu model. These are further analyzed in the subsequent section.

6.4 Test data used

The tests were performed using A, B, C and D type data for the parameter tests. The complexity of the data varied from 15 to 25. The distribution of data is shown in Figure 1.

Table 1: Number of test data sets used for M1-M5

	A	B	C	D
15	10	3	3	3
20	10	3	3	3
25	10	3	3	3

6.5 Parameter model 1 (M1): Varying neighbourhood and phase

The implementation solves the problem by moving tasks on a timeline in the neighbourhood. Different neighbourhoods were examined in different phases.

6.5.1 Variation of parameters

Moving with fixed time step lengths: The overall tests were performed using a time step length, for moving one task forward or backward a predetermined number of time steps. In a tabu search manner, the solution with the lowest cost was chosen when not present in the tabu list. The cost of the current solution was evaluated with the following change to the time position of each task: [- x, ... , - z, + z, ... , + x]. In this setting, several time step lengths in different combinations were evaluated: longer and shorter.

Multiple phases: To fully leverage the benefits when using different time step lengths, and to add both precision and diversification, the solver was developed to use a cyclic phase behavior. This means that different sets of neighbourhoods were assessed in different phases of the solution process. Using the time step length as neighbourhood, a two-phase solver could be expressed as follows:

Table 2: Example of phases

Phase 1: Move the most beneficial task in time steps of length: +/- 2E7 or 1E8.
This represents a diversifying, longer time step length on the current timeline
Phase 2: Move the most beneficial task in time steps of length: +/- 2E5, 5E6 or 1E7
This is a precision phase with smaller updates

6.5.2 Parameters in models

The final evaluation was carried out using varying time step lengths and different phases. This section describes the actual values and phases for parameters in the evaluation.

Table 3: Phase description M1-1

Phase 1: Long	-1.5E8, -0.75E8, 0.75E8, 1.5E8
Phase 2: Long and short	-1.5E8, -0.75E8, -4E7, -8E6, -4E5, 4E5, 8E6, 4E7, 0.75E8, 1.5E8

Table 4: Phase description M1-2

Phase 1: Long	-1.5E8, -0.75E8, 0.75E8, 1.5E8
Phase 2: Medium	0.75E8, -2E8, -5E7, 5E7, 2E8, 0.75E8
Phase 3: Short	-5E7, -1E7, -5E6, -1E6, -5E5, 5E5, 1E6, 5E6 1E7, 5E7

Table 5: Phase description M1-3

Phase 1: Longest	-1.5E8, -0.75E8, 0.75E8, 1.5E8
Phase 2: Long	-0.75E8, -2E8, -1E8, 1E8, 2E8, 0.75E8
Phase 3: Medium	-1E8, -5E7, -1E7, 1E7, 5E7, 1E8
Phase 4: Short	-1E7, -5E6, 5E6, 1E7
Phase 5: Shortest	-5E6, -1E6, -5E5, -1E5, 1E5, 5E5, 1E6, 5E6

An additional test, which was discussed but dismissed before evaluation, was changing places of tasks. It was dismissed after logical resoning and due to the complexity issues it would have involved in the implementation.

6.5.3 Result

Two summary graphics have been added to show how the models perform for different data and data complexity.

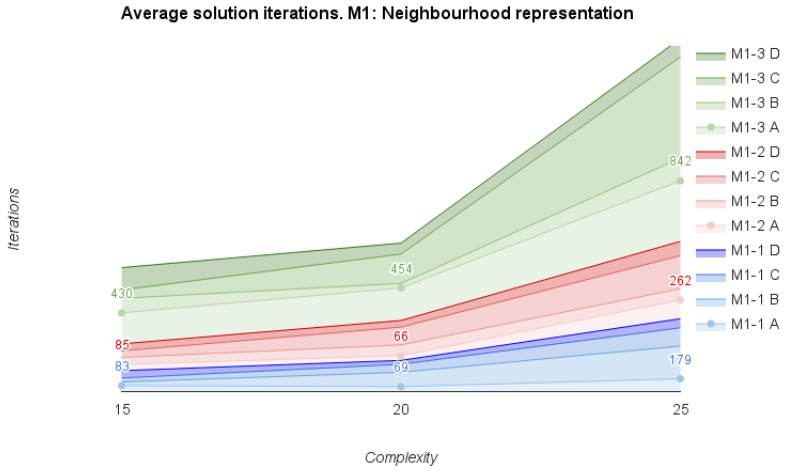


Figure 5: Average iterations when varying neighbourhood representation.

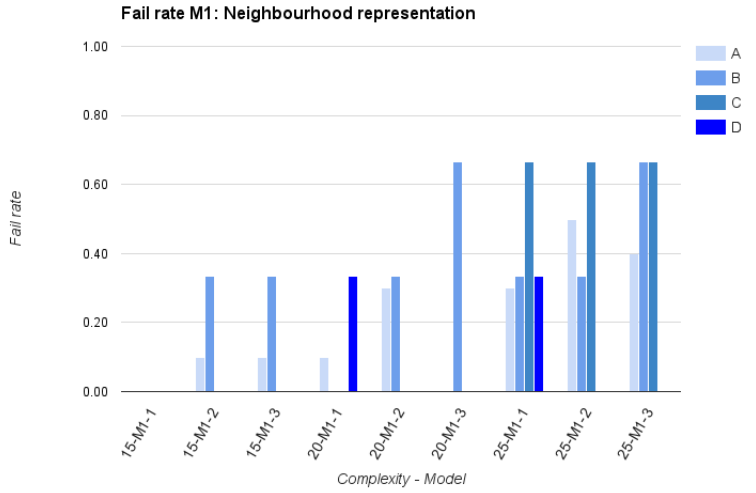


Figure 6: Failrate when varying neighbourhood representation.

6.5.4 Discussion

The algorithm show sensitivity to choice of time step length. When facing data with similar complexity the model is able to solve some of the data while some remain unsolvable (characteristics include high failrate with low average iterations). M1-1 is noted to be the most robust with an overall low fail rate. This

parameter setting does show difficulties in the smaller test sets (A15) which has a higher required rounds of iterations than for the more complex B20 data.

6.6 Parameter model 2 (M2): Varying tabu list construction

As previously mentioned, the tabu search uses a tabu list to avoid local optima. The tabu list can, however, be constructed in different ways. Two implementations were developed and tested of which the first was quickly dismissed. This section discusses learnings about the specific tabu list representation needed for this challenge.

Tabu list 1: Full solution tabu. A tabu list item contains the starting time and lengths of all tasks in the solution. When exploring neighbourhoods using many different time step lengths it was confirmed that, when using such a specific tabu list item such as the full tabu solution, the tabu list is almost never used. The search is very unlikely to return into a previously visited solution in the neighbourhood and the idea of the tabu list is lost.

Tabu list 2: Task tabu. Tabu representation 2 contains the last moved tasks in a sorted list of a fixed length. Having every previously moved task in the tabu list forces the solver to vary the neighbourhood in a manner that by its setup diversifies the solution.

6.6.1 Testing model

For the sake of testing, trials were performed using the full solution tabu list as a comparison between the two mentioned tabu lists. The M1-1 realization uses tabu list 2 (task tabu) with a tabu list length of 20. The M2-1 model does not hold a tabu list at all to represent the tabu list 1 scenario.

6.6.2 Result

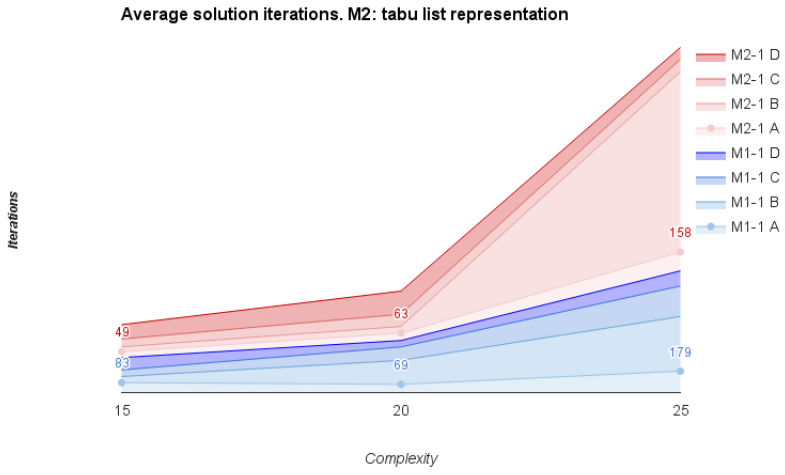


Figure 7: Average iterations with and without tabu list.

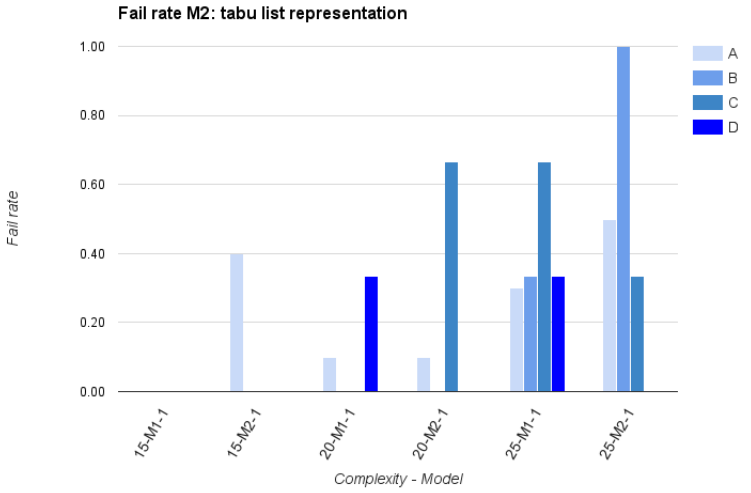


Figure 8: Failrate with and without tabu list.

6.6.3 Discussion

The M2-1, "no tabu list" method was showing a stable number of average iterations until the complexity was increased to 25. Here, we see that the fail rate reach 100% for type B data. Also a generally higher fail rate is observed. The use of a tabu list in the solver proved to be beneficial.

6.7 Parameter model 3 (M3): Varying tabu list length

Tuning the tabu list means changing the tabu list length. A longer list equals more diversification and should yield a higher chance of reaching more solution neighbourhoods and thus reaching the global optimum. With a shorter list, the solver prioritize the most rewarding solution in a higher degree and thus might get stuck in a local optimum easier due to the lack of diversification. However, it provides a larger degree of intensification meaning that each neghbourhood is thoroughly investigated. In order to know what length is optimal for our problem, we tested different lenghts.

6.7.1 Testing model

For testing the effects of varying the tabu list length three different list lengths were established as presented below.

Table 6: Test parameters

Short tabu list length = 10 tasks : M3-1
Normal tabu list length = 20 tasks: M1-1
Scaling tabu list length = $\frac{nrtasks}{2}$ tasks: M3-2

6.7.2 Result

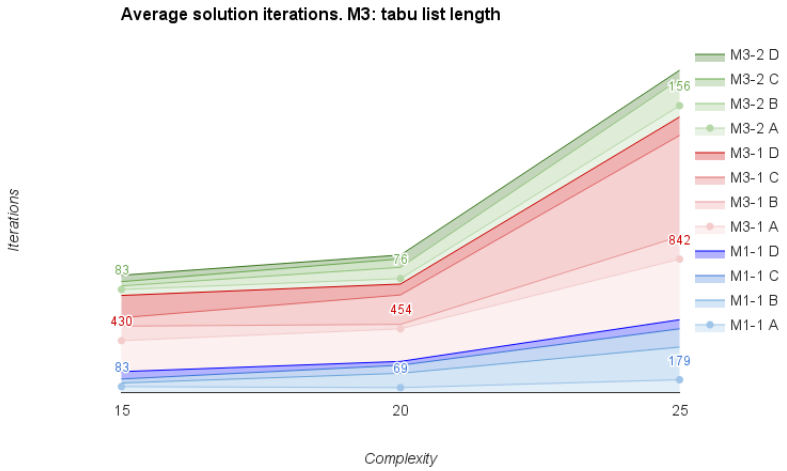


Figure 9: Average iterations when varying tabu list length.

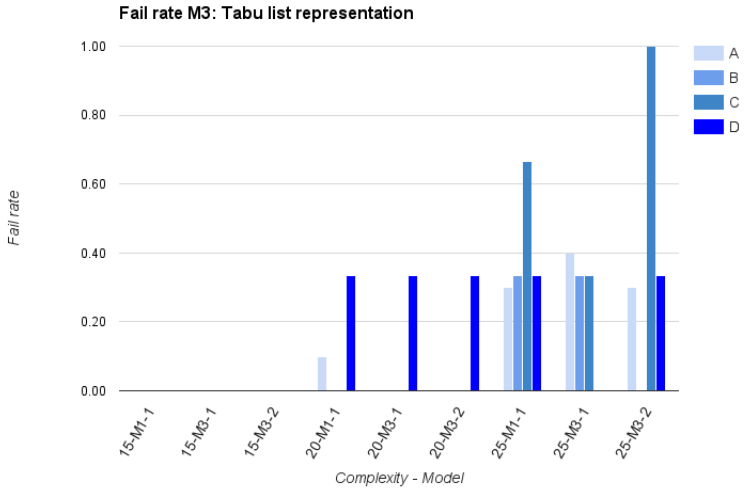


Figure 10: Fail rate when varying tabu list length.

6.7.3 Discussion

The model using a short list (M3-1) was the slowest but showed slightly better performance, in terms of low fail rate. The longer list (M3-2) was overall the fastest but showed issues dealing with more complex data as the fail rate increased rapidly.

6.8 Parameter model 4 (M4): Varying phase stopping criteria

6.8.1 Stopping criteria models

In the basic model, three bad moves in a row generated a phase change to intensification, while five bad moves in a row generated a phase change to diversification. We developed two models which had a smaller or larger numer of bad moves before a phase change. The advantage of fewer bad moves is that we change as soon as the solution is getting worse. This also means that we can move away from a good area since we are not patient enough to wait for improvements.

Table 7: Number of bad iterations before phase change M4-1 and M4-2

Mode/Phase:	Phase 1	Phase 2
M1-1 (basic model):	3	5
M4-1:	3	3
M4-2:	5	5

6.8.2 Result

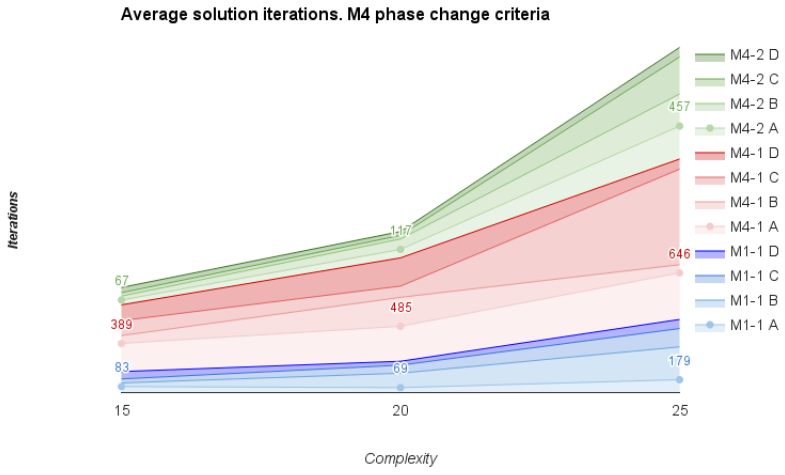


Figure 11: Average iterations when varying number of bad iterations before phase change.

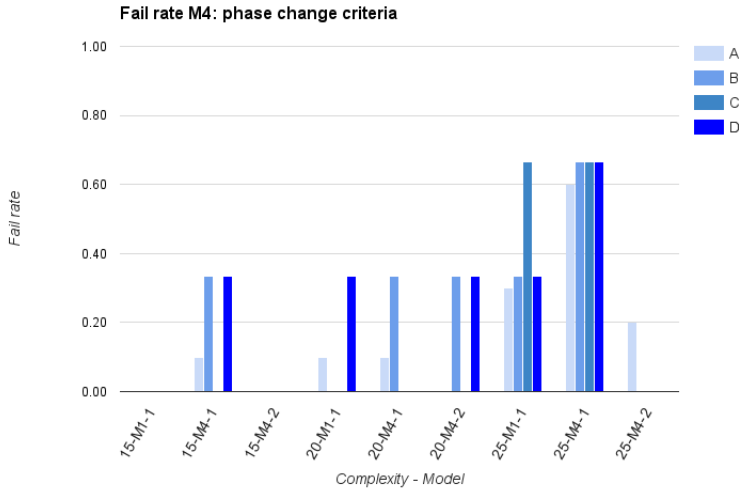


Figure 12: Fail rate when varying number of bad iterations before phase change.

6.8.3 Discussion

Conclusions which can be made from the graphs is that if the phases were changed too quickly it lead to an increase in the number of iterations before the problem could be solved. Also the failure rate was very high for this model. Having a larger number of bad moves allowed before changing phase gave little worse iteration times compared to the basic model, but much better success rates. We concluded that 5 iterations was the best, but other models could also be tested.

6.9 Parameter model 5 (M5): Varying dynamic weights

6.9.1 Weight models

As described in the basic implementation section, the method uses dynamic updating of the weights in the cost function. The highest cost gets a proportionally higher weight. The idea was to guide the search so that the largest infeasibilities are resolved first. If, for example, there are many tasks overlapping, this infeasibility would be given higher priority as this cost gets heavily weighted in the cost function.

Updating the weights too often is problably confusing the solver as focus on what to solve first is shifted. Updating too seldom however, means that the solver might not be able to resolve any problem as all are equally prioritized. This could for example lead to deadlocks. We tested the different setups described in Table 8.

Table 8: Number of iterations before changing weights M4-1 and M4-2

Model/Phase:	Phase 1	Phase 2
M1-1(basic model):	50	50
M5-1:	25	25
M5-2:	10	10
M5-3:	never	never

6.9.2 Results

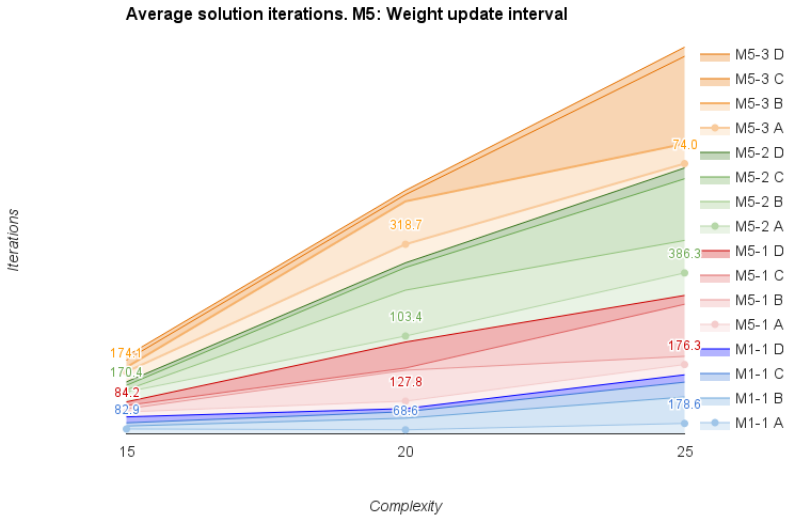


Figure 13: Average iterations for updating weights at different intervals.

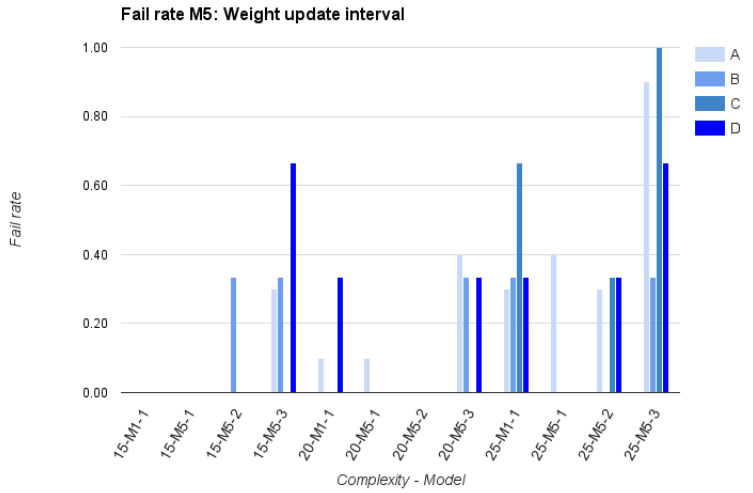


Figure 14: Fail rate for updating weights at different intervals.

6.9.3 Discussion

All models showed a similar number of average iterations although the best results were obtained when updating the weights every 25th iteration. This model also gave a low failure rate. Never updating the weights gave the worst result, indicating that dynamic weights is an important feature in our solver.

6.10 Final realizations

6.10.1 Combining the models

Using the results from the previous section we were able to create four new models: MA, MB, MC and MD. These models were specifically designed to fit the four types of data used by combining all the parameter values obtained in the previous section which worked best for the specified data type. This resulted in the models seen in Table 9. Model MB and MC turned out to be identical.

Table 9: Data specific parameter models

Model:	MA	MB	MC	MD
Number of phases:	2	2	2	3
Tabu list length:	10	10	10	10
Number of iterations before phase change:	5:5	5:5	5:5	5:5:5
Dynamic weight updated at iteration:	10:10	25:25	25:25	25:25:25

When evaluating the four models, MA to MD, it was found out that models MB and MD performed the best and thus model MA was discarded. Surprisingly, model MD, which has 3 phases, performed best, contrary to our experiences based on the previous experiments.

6.10.2 Results

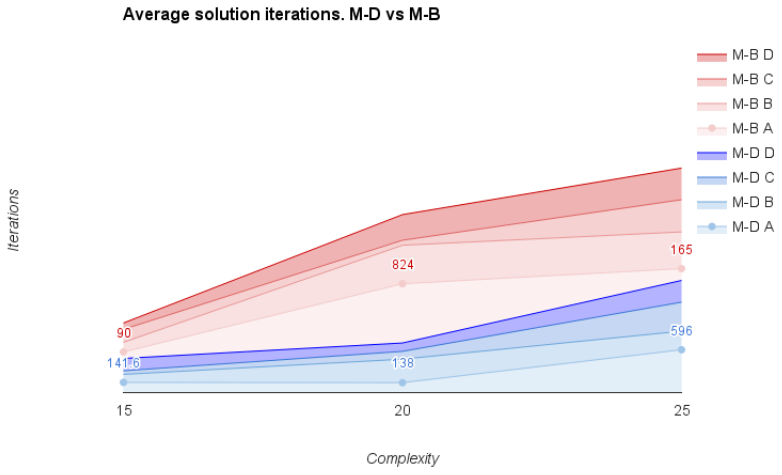


Figure 15: Average iterations of models MB and MD.

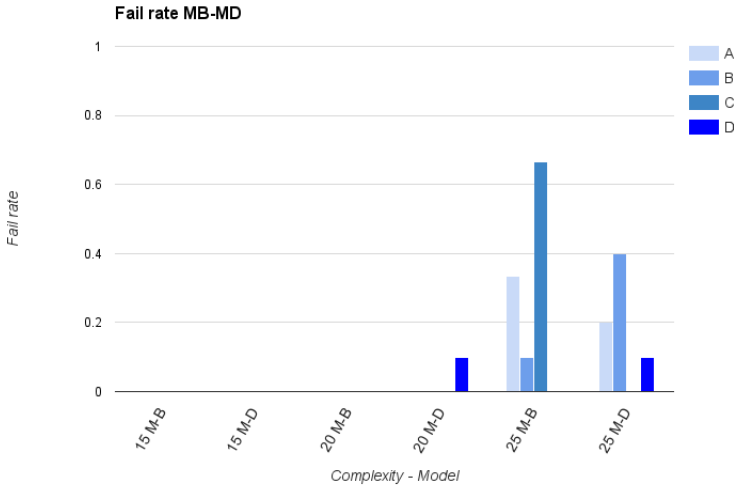


Figure 16: Fail rate of models MB and MD.

6.10.3 Discussion

The method of combining promising parameter settings in a new model was not necessarily bound to result in good models. However, Figures 15 and 16 suggest that both models are working well up to complexity-25 data. For these data, model MD performs better on type C and A data while model MB performs better on the other types of data. Overall, model MD seems like the better model from a fail rate perspective.

Looking at the average number iterations in Figure 15, model MD also seems to perform slightly better than model MB for all types of data.

6.11 Further development

Comparing all models found in this study and looking at the rate of failure of the different models, the best performance is given by models M4-2, M5-1 and MD. Their total failure rates, as a sum of are given in table 10. It can thus be concluded that MD is the best model developed.

Table 10: The total failure rates of the best models

Model:	M4-2	M5-1	MD
Total failure rate:	0.86	0.5	0.2

This model can perform quite well up to data with complexity 25. This corresponds to around 70 tasks. Complexity 30, which usually is unsolvable, corresponds to data with 100 tasks. As can be observed in Figure 17, the cost function oscillates heavily for the MD model and does not seem to approach a solution. This must be further researched in order to draw any conclusions about why it happens.

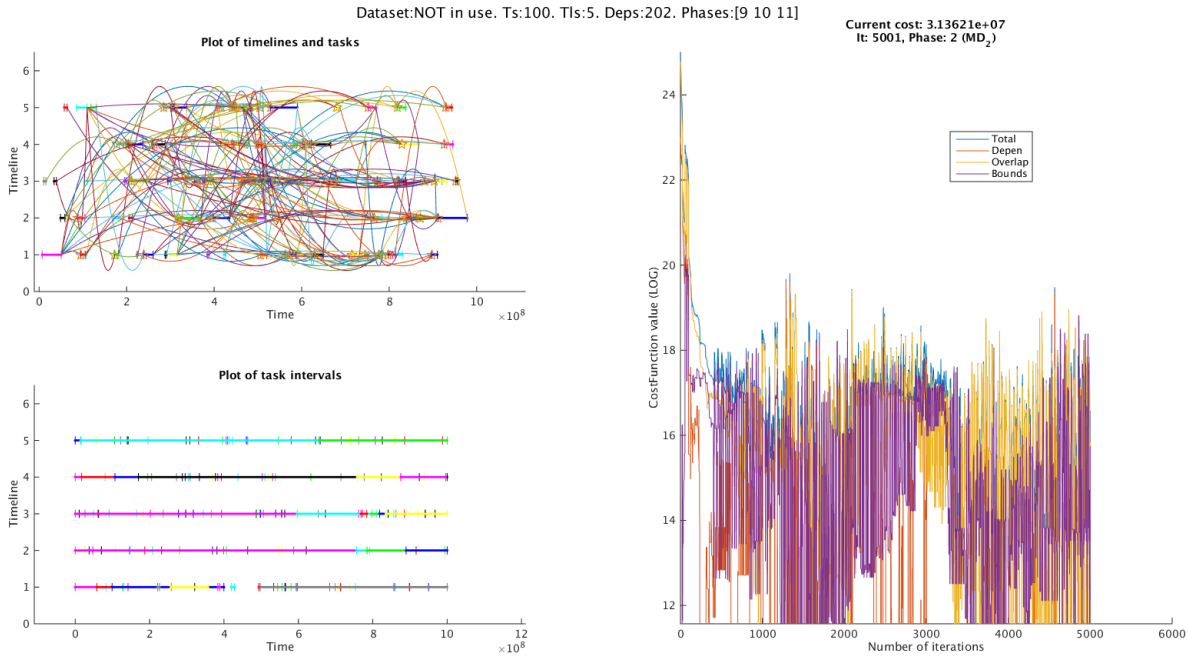


Figure 17: Failed run with MD for 30-complexity data.

7 Analysis of LNS

7.1 The LNS search heuristic

Large Neighborhood Search (LNS) is a metaheuristic which given an initial solution gradually improves it by alternately destroying, repairing and accepting the solution [2]. Destroying the solution means to take a given solution and to destroy it, in our case by unfixing some of the y_{ijg}-variables, see Appendix A for variable definitions. The LNS will then repair the destroyed part, the accept function will check that the newly repaired solution is better, in which case it will set the new solution as the current one. The advantage of using an LNS method is that it quickly moves through the solution space and returns a local optimum of high quality. It is possible to customize the destroy and repair function depending on prior knowledge about the data, therefore a LNS heuristic implementation can be varied depending on the problem.

Based on an initial solution, created after inserting non-negative artificial variables, the solution will be alternately destroyed and repaired with the aim of reducing the objective function value to zero. This will be attained when all of the artificial variables are set to zero, which represents that a feasible solution to the original problem has been found.

The group was recommended to use CPLEX, a commercial mathematical programming solver, as a repair function and did therefore not develop one [3]. Otherwise a heuristic would have to be chosen and implemented to repair the solution.

By implementing an accept function, which the group did not do, it will be possible to accept a temporary solution as the current solution even though the objective function value does not improve. This results in

a wider scan of the search space as you more easily leave local optima. However, the accept function is only used when the repair function is not an optimal solver, which CPLEX is.

7.2 Destroy function

The destroy function destroys the solution according to a given degree chosen by the user. The degree is a value chosen between 0 and 1, where the value 0 corresponds to not destroying the solution at all and 1 corresponds to destroying a large part of the solution. Based on the destroy degree, the function calculates how many y-variables it needs to unfix depending on the instance size. An integer called "destroy count" represents that amount. The group has implemented two different ways to choose which y-variables will be destroyed. However, both ways unfix one y-variable at a time until the number of unfixed y-variables has reached the "destroy count".

7.2.1 Random destroy

The first version of the destroy function chooses which y-variables to unfix randomly. Each y-variables index is saved in a list and before a new variable is unfixed the function looks through the list to make sure it is unique. This ensures that even though the function is random it will always unfix the correct number of unique y-variables.

7.2.2 Active destroy

The second version of the destroy function an active choice on which y-variables to unfix is made. The active choice is made by finding all non-zero a-variables, and unfixing the y-variables that shares the indices. This makes sure that the function destroys the solution where the violations occur. If the number of violation variables are fewer than the destroy count, additional y-variables will be unfixed by using the random destroy function.

7.3 Results and analysis

The tests were run on data with a complexity ranging from 0 to 45. Ten different instances of each data type was used so that averages and success rates could be calculated. Data set A0 would for example have ten instances named A0_1-A0_10. This is described more thoroughly in Section 4. The success rate tells whether a feasible solution was found for the data set or not. As an example, if a function manages to solve eight out of ten instances for A0 then the success rate would be 80%. In the tables below, Table 11 and Table 12, the success rates and average number of iterations are presented. All data sets with lower complexity than of those displayed have success rates of 100%.

Table 11: Model: Active destroy, Destroy degree: 0.4, Time limit: 2000

Data set	A35	A40	A45	B35	B40	B45	C35	C40	C45	D35	D40	D45
Success rate, %	90	40	0	100	0	0	100	40	0	100	100	20
Number of iterations	339,8	414,8	99,6	396,9	417,7	97,8	362,7	627,1	76,4	229,5	231,5	421

Table 12: Model: Active destroy, Destroy degree: 0.5, Time limit: 2000

Data set	A40	A45	B40	B45	C40	C45	D40	D45
Success rate, %	50	0	20	0	70	0	100	100
Number of iterations	410,4	150,2	207,5	196,6	418,3	144,6	109	242,3

The results show that the LNS has a rate of success of 100 % for all but one data type with a complexity of 35 or lower. For complexities of 40 and 45 the LNS has a varied rate of success depending on the data type, which indicates that some types of data are easier to solve for the LNS than others. For example, the LNS is able to solve D40 with a success rate of 100 % as well as some instances of D45 data, with a success rate of 20 %. The success rate for all other data with complexity 45 was 0 %.

Any two data types with the same complexity have the same number of tasks. Therefore, it is not unreasonable that type D data are the easiest to solve due to the fact that they contain many timelines, allowing them to be placed in a parallel order.

In Table 13 the results with random destroy as model are presented. Due to the success rate not being 100 % for the data types with lower complexity, this table is larger.

Table 13: Model: Random destroy, Destroy degree: 0.4, Time limit: 600

Data set	A0	A5	A10	A15	A20	A25	A30	A35	A40	A45
Success rate, %	80	100	90	100	90	40	0	0	0	0
Data set	B0	B5	B10	B15	B20	B25	B30	B35	B40	B45
Success rate, %	100	90	100	70	60	10	0	0	0	0
Data set	C0	C5	C10	C15	C20	C25	C30	C35	C40	C45
Success rate, %	100	90	100	100	90	60	0	0	0	0
Data set	D0	D5	D10	D15	D20	D25	D30	D35	D40	D45
Success rate, %	90	80	100	100	90	40	0	0	0	0

Despite the lower time limit it is clear that the active destroy is more efficient than the random destroy.

7.4 Further development

Depending on how the heuristic is implemented the efficiency of solving a problem will vary greatly, and the number of ways to implement it is unlimited. The most effective way to solve new problems could be to identify if they follow a certain structure and implement the heuristic based on that information. Creating a general principle that would solve any large problem would either be difficult, time-consuming or produce slow results. For a certain problem there are a few complexity parameters that could differ, such as the number of time-lines, dependencies and tasks, all of which might require a different approach in order to effectively solve the problem.

An implementation that would be of interest to try is to destroy additional variables in the surroundings instead of just the ones violating the constraints. By doing this, the results would most likely improve since much of the difficulty lies in the ability to optimize around a small cluster of y-variables. One variable is connected to at least two others, the previous and following one, which is the reason for the wish of optimizing in small clusters. Due to lack of time, the group was not able to implement this functionality and therefore unable to explore this possible improvement for the heuristic.

Another thing to investigate is how the LNS heuristic is affected by the initial solution, which has not been done in this project. By choosing a different initial solution, the LNS heuristic will not have to optimize as much to find a solution as many of the constraints would already be satisfied.

8 Comparison between heuristics

Looking at the figures one can see that the tabu search finds a solution much quicker compared to LNS, although the latter displays a much lower failure rate. It should be noted though that the data sets compared are those for which both heuristics are able to find a solution for, taken from Figure 16 and Table 11.

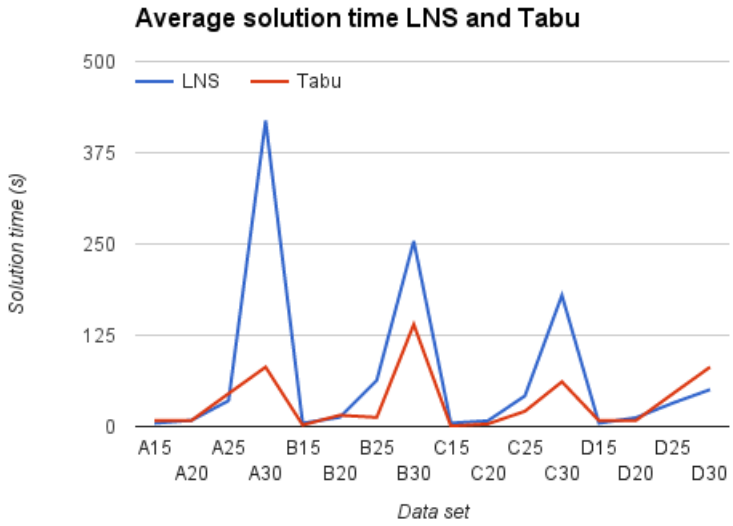


Figure 18: Average solution time for tabu search and LNS.

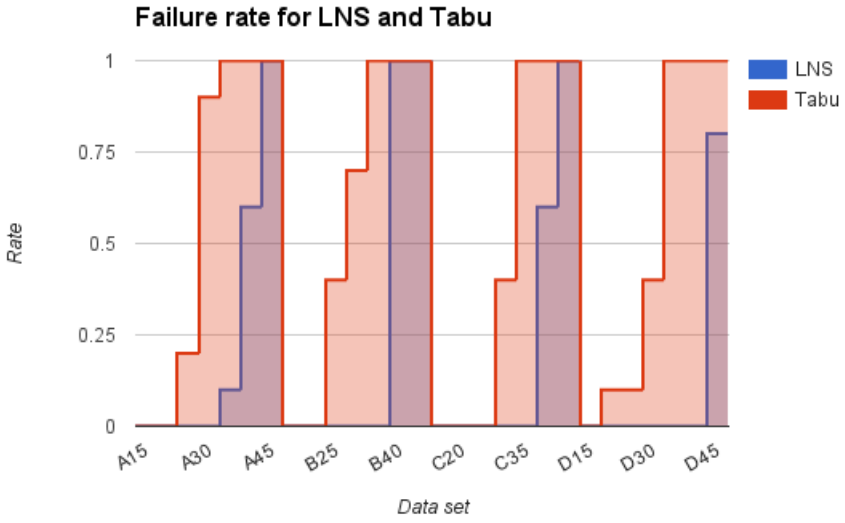


Figure 19: Fail rate for tabu search and LNS.

From these results, it is hard to conclude which heuristic is the better option. There is great potential for development in both cases and the results could probably be greatly improved for both heuristics.

9 Conclusion

We can see that during this half-year project we have been able to gain some knowledge about the avionics scheduling problem and the two heuristic principles, tabu search and LNS. We managed to build a testing environment for the problem, generate data and test a few versions of the heuristics. The final solvers we developed could solve the scheduling problem up to a certain complexity. Around 200 tasks were the limit for LNS. Further studies of the behaviour of the heuristics on this problem are needed in order to develop better solution methods.

References

[1] http://www.ida.liu.se/~zebpe83/heuristic/papers/TS_tutorial.pdf

[2] http://www.diku.dk/~sropke/Papers/lns.pdf

[3] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Appendices

A Mathematical model

The group received a mathematical model for the scheduling problem under consideration from the customer. This model was later modified. The first model presented is the given model and the model after is the modified one which the group used when developing the LNS.

The given model:

Sets:

G = set of index for modules
I = set of index for tasks
I_g^o = set of index for tasks on module g
J_{ig}^o = set of index for tasks that can be placed after task i on module g
D = set of index for dependencies
I_d^{Dep} = set containing the pair of indices for tasks in dependency d

Variables:

x_i^s = start time for task i
x_i^e = start time for task i

$$y_{ijg} = \begin{cases} 1, & \text{if } j \text{ follows right after } i \text{ on module } g \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ig}^o = \begin{cases} 1, & \text{if } i \text{ is first on module } g \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ig}^T = \begin{cases} 1, & \text{if } i \text{ is last on module } g \\ 0, & \text{otherwise} \end{cases}$$

Parameters:

u_d = distance between i and j in dependency d
l_i = length of task i (constant for this problem)
t_i^s = earliest possible start time for task i
t_i^e = latest possible end time for task i
f_d^{min} = smallest allowed distance, dependency d
f_d^{max} = largest allowed distance, dependency d

$$
\begin{aligned}
\min \quad & z = 0 \\
\text{subject to} \quad & \sum_{i \in I_g^o} y_{ig}^o = 1, \;\; g \in G && (1) \\
& \sum_{i \in I_g^o} y_{ig}^T = 1, \;\; g \in G && (2) \\
& \sum_{j \in J_{ig}^o} y_{ijg} + y_{ig}^T = 1, \;\; i \in I_g^o, \; g \in G && (3) \\
& \sum_{i \in I_g^o} y_{ijg} + y_{jg}^o = 1, \;\; j \in I_g^o, \; g \in G && (4) \\
& x_j^s = x_i^s + u_d, \;\; (i,j) \in I_d^{Dep}, \; d \in D && (5) \\
& x_j^s \geq x_i^e - M(1 - y_{ijg}), \;\; j \in J_{ig}^o, \; i \in I_g^o, \; g \in G && (6) \\
& t_i^s \leq x_i^s \leq t_i^e, \;\; i \in I && (7.1) \\
& t_i^s \leq x_i^e \leq t_i^e, \;\; i \in I && (7.2) \\
& u_d \geq f_d^{min} && (8.1) \\
& u_d \leq f_d^{max} && (8.2) \\
& y_{ig}^o, y_{ig}^T \in \{0,1\}, \;\; i \in I, \; g \in G && (8.3) \\
& y_{ijg} \in \{0,1\}, \;\; i \in I, \; j \in I, \; g \in G && (8.4)
\end{aligned}
$$

The group took the given model and modified it by adding and removing sets, parameters, variables, modifying the conditions and adding artificial variables and an objective function. The artificial variables measures and punish violations on conditions (7) - (11).

The modified model:

Sets:

G = set of index for modules
I = set of index for tasks
I_g^o = set of index for tasks on module g
D = set of index for dependencies

Parameters:

l_i = length of task i (constant for this problem)
t_i^s = earliest possible start time for task i
t_i^e = latest possible end time for task i
f_d^{min} = smallest allowed distance, dependency d
f_d^{max} = largest allowed distance, dependency d
I_d^{df} = index of first task in dependency d
I_d^{dt} = index of second task in dependency d

Variables:

x_i^s = start time for task i
x_i^e = start time for task i

$$y_{ijg} = \begin{cases} 1, & \text{if } j \text{ follows right after } i \text{ on module } g \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ig}^s = \begin{cases} 1, & \text{if } i \text{ is first on module } g \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ig}^e = \begin{cases} 1, & \text{if } i \text{ is last on module } g \\ 0, & \text{otherwise} \end{cases}$$

u_d = distance between the two tasks in dependency d
$a_{i,j,g}^1$ = overlap between task i and j
a_i^2 = the amount with which x_i^s is outside its earliest possible start time on its interval t_i^s
a_i^3 = the amount with which x_i^e is outside its latest available interval t_i^e
a_d^4 = the amount with which u_d is lower than the shortest required distance f_d^{min}
a_d^5 = the amount with which u_d is higher than the longest possible distance f_d^{max}

$$\min \quad z = \sum_{i \in I_g} \sum_{j \in J^o_{ig}} \sum_{g \in G} C^1_{ijg} a^1_{ijg} + \sum_{i \in I_g} (C^2_i a^2_i + C^3_i a^3_i) + \sum_{d \in D} (C^4_d a^4_d + C^5_d a^5_d)$$

subject to

$$\sum_{i \in I^o_g} y^o_{ig} = 1, \;\; g \in G \tag{1}$$

$$\sum_{i \in I^o_g} y^T_{ig} = 1, \;\; g \in G \tag{2}$$

$$\sum_{j \in J^o_{ig}} y_{ijg} + y^T_{ig} = 1, \;\; i \in I^o_g, \; g \in G \tag{3}$$

$$\sum_{i \in I^o_g} y_{ijg} + y^o_{jg} = 1, \;\; j \in I^o_g, \; g \in G \tag{4}$$

$$x^e_i = x^s_i + l_i, \;\; i \in I \tag{5}$$

$$x^s_i = x^e_j + u_d, \;\; i \in I^{df}_d, \; j \in I^{dt}_d, \; d \in D \tag{6}$$

$$x^s_j - x^s_i + a^1_{i,j,g} \geq -(t^e_i - t^s_j) * (1 - y_{i,j,g}), \quad g \in G, \; i, j \in I^o_g \tag{7}$$

$$x^s_i + a^2_i \geq t^s_i, \;\; i \in I \tag{8}$$

$$x^e_i \leq t^e_i + a^3_i, \;\; i \in I \tag{9}$$

$$u_d + a^4_d \geq f^{min}_d \tag{10}$$

$$u_d - a^5_d \leq f^{max}_d \tag{11}$$

$$y_{i,i,g} = 0, \;\; i \in I, \; g \in G \tag{12}$$

$$y^s_{i,g} + y^e_{i,g} \leq 1, \;\; i \in I, \; g \in G \tag{13}$$

$$y^s_{ig}, y^e_{ig} \in \{0, 1\}, \;\; i \in I, \; g \in G \tag{14}$$

$$y_{ijg} \in \{0, 1\}, \;\; i \in I, j \in I, \; g \in G \tag{15}$$