The Duty Allocation Problem at Netherlands Railways

A tabu-search approach

Martin Koppers

31-7-2009

Econometrie Erasmus Rotterdam

(281824)

Supervisor:

Dennis Huisman - Erasmus University / NS

Index:

Chapter 1 Introduction

Netherlands Railways (NS) provides public train services in the Netherlands. Traveling by train is not just an alternative if the car breaks down, it is a way of traveling that people depend on. Every day around 1.2 million passengers make use of the provided train services. The Dutch railway network is the most heavily used train network in all of Europe with nearly 5,500 trains a day. In 2006, almost nine million different passengers traveled about 15.8 billion passenger kilometers by train. This has gone up from 8 billion passenger kilometers in 1970. On average, each Dutch citizens travel approximately 1,000 kilometers by train per year.

Until 1995 NS was owned by the state, providing passenger and freight services and building and maintaining the railway network. Because of European Union regulations and liberalization of the railway market, NS was split into several companies. Nowadays the state still owns the railway network. They also own ProRail, a nonprofit company which takes care of maintaining the railway network and allocating the goods- and passenger trains on it. Of the entire collection of trains making use of the railway network around 95% are passenger trains. The part of our railway system that provides the transportation of passengers is now done by NS. (Kroon, L. et al 2009).

Within NS there are several branches working to improve the quality of its railway services. Everything which has to with the driving of trains and facilitating the customers is being done by NS Reizigers. Some of its main concerns is to work on an as high as possible punctuality and to make sure the customers are well informed especially during downtime of parts of the network in case of for instance maintenance or accidents. To make sure NS can provide all its services the company has a staff of around 8,000 employees working as drivers, conductors or information provider. The drivers and the conductors operate from 29 different crew bases throughout the country. (NS Reizigers).

With the pursuing of a high punctuality comes the need for work schedules. The trains cannot drive themselves so there must be a driver ready for each train. Besides a driver, a train must also have a number of conductors. Assigning drivers and conductors to each train is done by NS

Reizigers' logistics research department in Utrecht. They create the work schedules for the entire working staff of drivers and conductors. In the literature this is referred to as rostering.

Rostering is part of a planning process. The entire process involves some phases for instance a long term strategy for buying new trains or the construction of a schedule for an entire year. Other parts of the process are for adjusting current schedules, for instance, when tracks need to be repaired and trains need to be rerouted or unforeseen events happen like accidents or extreme weather circumstances. When new schedules are being made they are made in sequences of 3 steps. First the times and routes for the train lines are planned. Second the rolling stock is assigned to the train lines. The last step is assigning the drivers and conductors to each train.

With this thesis we will look into the last step of scheduling in the planning process, the assignment of drivers and conductors to a train. Currently this problem is solved by means of an integer programming model and solved with the program Cplex. We will work on the same problem but from another point of view. Instead of an exact solution method we will use a heuristic to solve the problem. To be more precise, a meta-heuristic called tabu-search. Heuristics are known for their great successes in solving difficult combinatorial problems. When going through the literature we quickly found that the tabu-search meta heuristic is one of the best ways to solve assignment problems such as ours. To discover if a tabu-search program can indeed solve our problem and maybe even improve already found solutions by the integer programming model, we will build such a tabu-search program.

The thesis will continue after this introduction with a problem description in chapter 2. In the chapter 3 a general explanation of tabu-search is given. Chapter 4 will give a thorough description of all the components of tabu-search and how they are applied. This chapter includes some advanced techniques of improving a standard tabu-search to increase its solving capabilities. A short explanation of the data used and a representation of the objective function used can be found in the first part of chapter 5. In the second part of chapter 5 will be spoken of the insights found while testing the tabu-search. Our tabu-search has been applied to 3 different cases, the results of these case studies are presented in chapter 6. Finally we end with conclusions and recommendations in Chapter 7.

Chapter 2 Problem description

The problem we solve in this thesis is dividing the work load of NS crew members. The crew consists of drivers and conductors divided over 29 crew bases in the country. Every individual crew member must know for every day at which time he needs to work. The work in each train, whether it is driving the train or a ticket control, is part of what is called a *duty*. More formerly, a duty is the work for one crew member on one day. A schedule which gives the crew member the information of which duty he must perform on which day is called a *roster*. Crew members who share work times, early shifts or late shifts, amongst other preferences are grouped together. These are the *roster groups*.

What makes this roster special is that it is a so called cyclic roster. This means that for every driver or conductor the roster has one week of duties. Now for example one crew member starts with the duties of week 1 and the next week the duties of week 2. After he reaches the last week of duties he will start over again with the duties of week 1. This will be the same for another crew member with the only difference that the other crew member starts with the duties of week 2 in week 1, then the duties of week 3 in week 2 etc. So in the end all crew members perform the same duties, but in a different week.

To create rosters, we can distinguish two steps:

1. The allocation of the duties, in a fair way, amongst the different roster groups of crew members.
2. The actual construction of the rosters for each group.

Although the construction of a good roster is an interesting topic we focus in this thesis on the allocation of the duties.

To allocate duties to the roster groups we have to consider 4 groups of constraints. These restrictions can be categorized in the following way:

Work time preferences:

Some groups of crew members are free from night shifts in particular crew members older than 55. Other groups do not want to work too early. Every group therefore has a starting time and an ending time. Duties may not be assigned to a group if the duty's starting time is earlier than the allowed starting time of the group and/or it ends later then the ending time of the group.

Average duty length:

The average duty length of all duties in a single group may not exceed 8 hours. This must hold because of the rule that there must be 9 working days of 8 hours per 2 weeks. This translates to 72 hours per 2 weeks which equals 36 hours per week. If we allow this to be exceeded, then it would not be possible to construct rosters with an average working time per week of at most 36 hours.

Labor rules:

These rules are either established by law or collective labor agreements.

Every group can handle a number of duties of the type early or late or a combination of the two and this amount may not be exceeded.

The amount of night shifts may not exceed 53.6% of the total amount of duties in a group.

The amount of duties with a duty length shorter than 5 hours may not exceed 5% of the total amount of duties in a group.

The amount of duties with a duty length longer than 9 hours may not exceed 5% of the total amount of duties in a group.

'Sweet and sour'-rules:

Starting in 1984, NS tried to simplify the duties of its crew. The idea was to make the crew keep doing the same duties to improve punctuality. The crew objected greatly, they were afraid the work they needed to do would become to repetitive. The plan was cancelled but in 1993 it was presented again as a solution. Without much change in the plan the crew even organized a strike to show their disapproval. Cancellation of the plan followed. In 1999 the plan shows up yet again and again the crew went on a strike even bigger than the last one. Whether it was stubbornness or a supreme faith in their plan to simplify, NS finally implements their plan in 2001 whether the crew liked it or not. With the hopes that the crew would eventually be proven wrong and improvements would be accomplished the opposite was realized. Punctuality dropped from 85.3% to 71% after only half a month. With the aim being 88% the plan clearly failed.

In 2002 a new plan was made to improve punctuality while at the same time keeping in mind the wishes of the crew. The plan was called the ‘Sweet and sour model’. The core of the plan consists of dividing the popular and not popular work as fairly as possible. The popular and unpopular work can be divided in 3 categories:

- % A-work: The percentage of time in a duty on an intercity train. Intercity trains make few stops which is pleasant for the conductors who will not have to step out and back in the train as often as with the non-intercity trains. This is a ‘positive’ category.

- % Aggression work: The percentage of time in a duty the crew works on routes which have been marked as an aggression route. These parts of the duty have had reports of assaults on the crew by passengers in the past and are therefore marked as problematic and undesirable. This is a ‘negative’ category.

- % Double Deck: The percentage of time in a duty on old double-deck trains. These trains are not comfortable to work in because of having to walk up and down the stairs, low ceilings and a lot of bending because of low seats. This is a ‘negative’ category.

Besides the 3 categories the model also tries to allocate the same route to as many different crew members as possible. In this way the model gives the crew enough variety in their work.

With the ‘sweet and sour’-rules as an objective together with the other restrictions the problem is essentially an assignment problem. Such a problem can be solved via an exact method or an approximation method better known as a heuristic. This thesis will use a tabu-search meta heuristic to solve the problem.

We now define the Railway Duty Assignment Problem as the assignment of all duties of a certain crew base to different roster groups such that from every roster group a feasible roster can be constructed and the popular and unpopular work is as fairly as possible distributed amongst the different roster groups.

Chapter 3 Tabu-search

Tabu-search is a heuristic method to solve various combinatorial problems. It is often used to approximate optimal solutions for NP-hard problems. It falls in the category of meta-heuristics. Examples of such methods are simulated annealing (Eglese 1990), ant colony optimization (Blum 2005), particle swarm optimization (Behdinan & Perez 2007) and various forms of genetic algorithms. They are all extensions of local search approaches starting with a solution and evolving or directing it for further improvement. It was originally introduced by Glover in 1986 and has been successfully applied to many cases after its introduction. The usefulness of the tabu-search method especially shows in large combinatorial problems such as the vehicle routing problem (Laporte 1992) and the capacitated plant location problem (Sridharan 1995).

The basic idea of a tabu-search is to overcome the problem of getting stuck in a local optimum. To show what we mean we visualize a simple maximization problem. In figure 1 a local optimum is shown and figure 2 shows for the same function the global optimum.

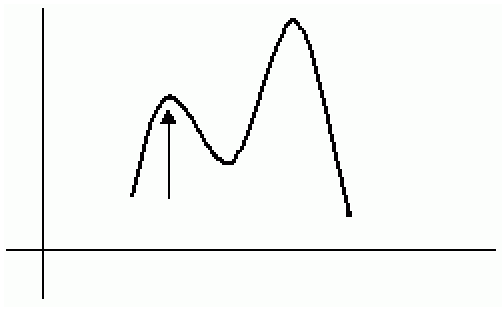


Figure 1: Local optimum

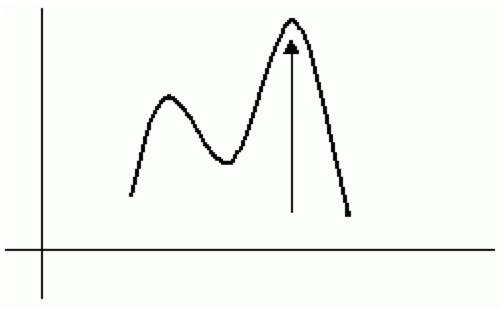


Figure 2: Global optimum

A local search method starts with a starting solution in a given search space and tries to find solutions which are better than the starting solution. The search will keep looking for improvements till it cannot find any solution which improves the current best found solution. For example take a steepest ascend method applied to above visualization. This ascend method looks for the best improvement of the objective function. When considering the polynomial as in figure 1 and 2 with a starting solution at the very left, the steepest ascend method would end up in the local optimum (figure 1). The desirable solution however is the one pointed at in figure 2.

To get to the global optimum the local optimum has to be abandoned. In order to proceed, a solution with an objective value worse than what has been marked as optimum so far has to be allowed. It means that in figure 1 the solution would move to a point lower than the local optimum. When the objective value of the solution goes down it can at some point go up again and eventually reach the optimum of figure 2. However allowing the solution to get worse also means that the search procedure can get back to where it has first started, in this case the starting point at the very left. To prevent this from happening the reversal of previous done movements are restricted. This restriction is called tabu. Tabu means something that may not be considered. A move in the solution space will be put on a tabu list for an amount of time given a static or dynamic value. The amount of moves that can be put on the tabu list and for how long they will stay there are important decisions to make. There are unfortunately no rules of thumb how to set these tabu parameters. The tabu list with the restricted moves on it is called the memory of the tabu algorithm. These memories, as there can be more of them, can also be used for remembering how much a move improved or worsened the objective and how many times a move has been put on tabu in the past. In more advanced tabu-searches there will also be a memory for keeping track of parameter settings when the tabu-search evolves its parameters as well as its found solutions. Getting the right parameters for these memories is often a process of trial and error to see what works best for the case being worked on.

Allowing movements that make the objective worse means that you have to keep track of the best found solution so far. This is because when the search has finished, the last found solution is most likely not the best like it is with a steepest ascend method. Unlike the memories, these best-so-far-solutions are not keeping track of recent history but of the entire history of the process.

The final part of a tabu-search is its termination. Unlike a steepest ascend method, a tabu-search often continues looking for better solutions even if it was not able to find an improvement during an iteration. Allowing decrement instead of only improvement makes it possible to keep looking for improvements. It is quite possible that without forcing the search to stop, that it will continue infinitely. This is not practical and a solution for this is to use a termination criterion that forces the tabu-search to stop when the criterion is met. Often tabu-search is used to run for a certain amount of time or iterations. The best solution it has found in this period will be the result of the search.

Some examples of termination criteria:

- after a fixed number of iterations,
- after a fixed amount of time,
- after a fixed number of iterations in which no improvement was found,
- when the objective value reaches a pre-specified value,
- when a feasible solution has been found.

The first two are the most commonly used criteria. The third criterion is often used in a more advanced tabu-search. With heuristics there is a tradeoff between time invested and the quality of a solution. The fourth criterion is useful when a fast result is desired. In case a problem is very difficult to solve the last criterion can be applied when the goal is to just find any feasible solution.

Chapter 4 Method

Section 4.1 Initialization

Tabu-search, like many other heuristics, evolves solutions in an attempt to find an optimal solution. Evolvement cannot happen without a solution to begin with. We will create such a starting solution in the initialization phase.

The problem we need to solve is the allocation of duties to roster groups. At first we put all duties in an additional group. We call this group the artificial group and it has as characteristics that no matter how many duties are in this group, it will not cause an infeasible solution. From there on we will move duties one by one out of the artificial group and into the roster groups. By means of single moves, see section 5.3.1, the duties will be moved.

The process of the initialization goes as follows. At random one duty from the artificial group is chosen to be moved. Then there will be a check in which of the roster groups the duty can be placed without losing feasibility. From amongst these feasible roster groups one roster group will be randomly chosen to move the duty into. If the duty cannot be moved to any roster group it will remain in the artificial group till further on during the search an opportunity arises to move the duty to a roster group.

The reason for the random selection of duties is because if the duties are ordered, for instance from largest to shortest duty length, and we pick the duties in the same order the roster groups can become packed with duties having a long duration and become infeasible for upcoming duties more easily. This especially becomes the case when roster groups are not randomly chosen. If for instance the order would be to start filling up the first roster group then the second one and so on, the result will often be that duties cannot be placed and will be kept in the artificial group. This can be dealt with by checking the duties in the artificial group several times but our goal in the initialization phase is to create a starting solution fast with as few duties in the artificial group as possible.

Section 4.2 Search Space & Neighborhood Structure

When the initialization step has finished the main part of the tabu-search can begin. The search will now attempt to find better solutions by means of moving duties around between different roster groups. Which movements are allowed and which not, is contained in the search space of the problem. For the RDAP the search space consists of all possible combinations of duties in roster groups with respect to the restrictions of the problem *(see chapter 3)*.

Within the search space we will try many different combinations of duties divided amongst roster groups. To start we mutate the first solution we have which has been generated during the initialization phase *(see previous section)*. In order for the tabu-search to know how it can mutate a solution at every iteration we must define what the neighborhood structure is for any given point in the search space. For this tabu-search we have chosen the neighborhood structure to be all possible combinations of duties in groups which can be obtained by executing one move between two roster groups. The definition of a move follows in the next section.

Section 4.3 Moves

The tabu-search meta-heuristic we use allocates duties to roster groups. The biggest part of the search contains of moves. Moving a single duty from one group to another or swapping two duties witch each other.

A move is a change in the list of duties of two different groups. The moves used in the tabu-search are the 'regular' or single move and the swap. Although they resemble each other quite a bit there are still some big differences which makes them both very useful in their own way. The next two subsections will briefly discuss the moves' properties.

4.3.1 Single Move

As the name implies it is a single move of a single duty going from one group to another. Of the two moves this is the only one that changes the number of duties in a group. The single move will be used the most at the start of the search and gradually be used less. This is because at the start of the search there is more space in the roster groups to change the amount of duties in the groups than later on in the search.

Computing a single move means checking an amount of duties equal to the number of duties in the first selected group plus the number of duties in the second selected group.

4.3.2 Swap

The swap is basically a double single move. Two single moves at the same time. From 2 different groups 1 duty will be moved to the other group and vice versa. Contrary to the single move the swap keeps the size of the groups the same. This makes the swap the only possible option when two groups are selected which both have no space for more duties.

Computing a swap means checking an amount of duties equal to the number of duties in the first selected group times the number of duties in the second selected group. This implies that the swap takes longer to compute and therefore single moves are preferred to save time, but especially later on during the search the swap move is likely to be the only option.

Section 4.4 Group Selection

To perform moves we need 2 groups for every iteration. Between these 2 groups we will look for duties that can be moved around in such a way that we gain better solutions. To decide which 2 groups to choose we consider 2 different approaches. One way is to randomly select and the other is based on the objective function, choosing 2 groups where we expect to achieve the biggest improvement.

We will first discuss the random selection approach. For this we randomly pick a group and afterwards pick randomly a group again that is not the same as the first chosen. This will be done at the start of every iteration. The reason for selecting randomly is because we want to make sure we search into as many groups as possible and therefore avoid getting the same 2 groups repeatedly.

We also tried a different approach. This approach involved looking specifically for each group to how much it could possibly improve the objective value. The objective function is based on the sweet and sour rules including diversity amongst routes (not having to go to the same place and back all day long) *(see section 5.1.3)*. In the perfect situation every group would have the same percentage of each sweet and sour category like A-work. However when trying to search for a good solution to the problem this is not the case. Some groups have very few A-work and others have a lot. With this approach we select the 2 groups which in total, summed over all categories, are furthest away from the average of each category. These 2 groups are then marked as most promising groups and we expect to find the best improvements with the selection of those 2 groups. The problem we found with such a greedy objective based approach is that it works fine as long as the search is able to find improvements. However, when with the chosen groups improvements cannot be found, the groups are still selected again for the next iteration. Because the objective function does not change when an improvement cannot be found the entire situation for selecting groups stays the same. This will partly be offset by making use of a tabu list *(see section 4.6)* but it cannot be prevented that basing the selection on the objective function will cause problems when the most promising groups selected yield no improvement at all.

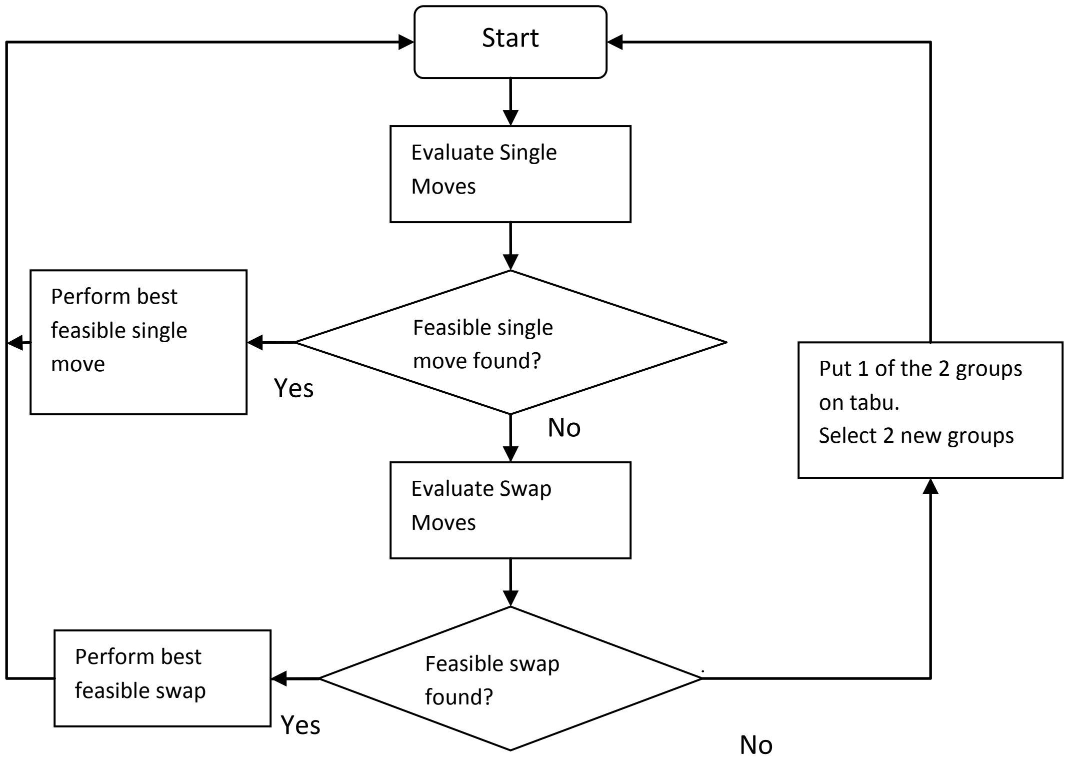
The fact that a random selection can choose any of the groups at any given iteration makes it a better approach. It allows for more freedom and with the greedy objective based approach it is always uncertain that the groups where you expect the best improvement really will give the best improvement. Unfortunately, the latter is often not the case.

Section 4.5 Duty Selection

In the previous section, we explained how to select the groups. We now need to choose one or two duties to move.

As explained in section 4.3 the options for moving duties around consist of single moves and swaps. After we know within which groups to search we have to decide which of these 2 moves we will perform and which duty in case of a single move and which duties in case of a swap we will move. How we evaluate which decision is best is shown below in figure 3.

The order in which the tabu-search looks for improvements can be summarized with a simple schedule:



Start
Evaluate Single Moves
Perform best feasible single move
Feasible single move found?
Yes
No
Put 1 of the 2 groups on tabu.
Select 2 new groups
Evaluate Swap Moves
Perform best feasible swap
Feasible swap found?
Yes
No

Figure 3: Schedule for each iteration

The idea is to check all possible single moves between the two groups and acknowledge the best one. If a feasible move was found then the best feasible single move will be performed. This is not necessarily a single move which lowers the objective value because decrements are allowed. If no feasible single moves were found all swap moves will be checked. This includes all swap moves with the artificial group if it is not empty yet. For the swaps with the artificial group there will be a check of all swaps between the first chosen group and the artificial group and then all swaps between the second chosen group and the artificial group. After all swap moves have been evaluated the best swap move is performed based on which feasible swap gave the lowest objective value, whether it was an improvement or not. The reason to check the single moves first is because the computation time of a single move is larger than of a swap. For a single move we have to check *n* + *m* different moves with *n* and *m* the amount of duties in first and second selected group. For the swap move we have to check *n* x *m* number of moves.

To find the best possible single move the steps taken are very similar to those of the initialization process. The first duty of one of the two selected groups is transferred to the other selected group. Its effect on the objective value is measured as well as the feasibility of the move. The move is feasible only if the restrictions mentioned in chapter 2 are not violated. Only if the move was found to be feasible, the objective value is compared with the current objective value. This repeats itself until all duties from both groups have been checked for their effect on the objective. From all feasible moves the best one will be performed, that is the single move which had the lowest objective value of all feasible single moves and the statistics of the groups are updated. If this single move has a lower objective value then the current objective value so far then the solution resulting from performing the single move will also be the new best solution found.

The next step is checking the swap moves. As it can be seen in the schedule, the swaps will only be done if there was no feasible single move found. This is done to save time. There could in theory be a swap that will outperform the best single move but it will be, in this thesis, sacrificed to uphold a high solution speed.

For the series of swaps the first duty in one of the selected groups is interchanged with the first duty of the other group. The feasibility and its effect on the objective value are measured. Afterwards the second duty in the other group is chosen and this is repeated till all duties in the second group have been exchanged with the first duty in the first selected group. Then the second duty in the first selected group gets exchanged with all the duties in the other group. We repeat this till all duties of both groups have been exchanged with each other.

After the swap moves have been checked and the best swap has been found its corresponding objective value is compared to the current objective value. Because the checking of the swap moves can only be reached if the single move test did not find an improvement, the best feasible swap will be performed and the statistics of the groups are updated. If the swap improves the current best found solution then the new solution found by doing the swap will now be stored as the new best found solution.

If both the best single move and the best swap move do not improve the current objective value the tabu-search puts one of the 2 groups on a tabu list. It basically means that one of the 2 groups may not be chosen again which will allow for a new group to be selected. With a new group the search can continue. We will continue now with an explanation of how we implemented and used tabu lists, in the next section.

Section 4.6 Memory of the search – Tabu Lists

4.6.1 Group Tabu

Duties are assigned to several roster groups. To find a better allocation of the duties between the roster groups 2 groups are selected at each iteration of the algorithm. Between those 2 groups duties will move around unless it causes infeasibility. If at some point no more moves between the 2 groups can be found the system needs a new group to continue the search. This means that 1 of the 2 groups will be put on a tabu list, the group-tabu list, so that this group cannot be chosen again while it is on the tabu list. To make the decision which of the 2 groups will be put on the tabu list 2 different strategies are tested. These are as follows:

1. random ‘flip of the coin’ decides which group to put on tabu
2. an objective based decision puts a group with least expected improvement on tabu

For strategy 1 we decided to let both groups weight equally. This means that both groups should have the same probability to be selected to be put on the group tabu list. As explained later on, without a clear indication what the better option is a random choice with equal probability to be selected for both groups is the closest we can get to the best decision given the current position in the search space. Also because the decision of which group to put on tabu resembles the decision of selecting 2 groups we thought randomness, just like in group selection, would be the best option. Nonetheless we tested a rule based strategy as well to know if randomness is indeed the better option.

To further explain strategy 2 let us take a look at an example:

Group	A-work	Aggression work	Double-deck work
1	75	6	5
2	73	5	6
3	74	4	5
4	73	4	5
5	74	5	3
6	75	6	4

Table 1: Example statistics for 6 roster groups

In this case we assume that there are 6 groups, 3 relevant objective categories and that group 2 and group 5 are currently chosen. We will look at the 'double-deck' category now because that is the category where the values of group 2 and 5 are the furthest away from each other. The average of this category of all groups is 28/6 ≈ 4.333. To know whether group 2 or group 5 must be put on the group-tabu list we look at the difference between the value of the group and the average value. The difference for group 2 is 6 – 4.333 ≈ 1.666 and for group 5 it is 4.333 – 3 ≈ 1.333. Because the difference of group 5 is smaller than that of group 2 we expect to find the largest improvement with group 2. Therefore group 5 will be put on tabu.

The problem with strategy 2 is like what we saw with selecting groups. Selecting groups or duties based on a rule gives good results for the current iteration the search is at. However, several iterations later the search might find a local optimum. After finding a local optimum the search can attempt to move away from this solution but because of the rule for selecting groups it is likely that the search goes in the direction of the local optimum again. This strategy prevents the search to be able to change its search direction after going a way which failed to get close to the global optimum.

Because of this inability to adapt, strategy 2 makes more sense. If there is no way to know which group to put on tabu, then decide randomly.

Besides knowing which group to put on tabu we also need to know for how long it must stay on the tabu list. For this no universal rule exists and in general testing a lot of different values gives insight in what values work well. A different value can have great effect on the results. This makes parameter testing important. Because of this fine tuning the construction of a good tabu-search can take a lot of time.

4.6.2 Duty Tabu

During the search duties will move around every iteration unless no move is possible. One of the biggest problems when using a search method that exchanges objects in a search space is that every mutation can also be reversed. Because such a behavior is undesired a second tabu list is introduced. We call this list the duty tabu list. Compared to the group tabu list the duty tabu list has the same characteristics. As much as we do not want a group to be selected after it was selected in the recent past, we do not want a duty to be selected again, which could possibly mean that it reverses a move recently done. Cycling should be avoided at all times because it slows the search down or prevents the ability to escape local optimums.

As explained before the purpose of the tabu list is to prevent solutions going back to a solution exactly the same as one found in the recent past. To do this a duty which has just been moved, one duty for a single move and two for a swap, will be put on the duty tabu list. For a chosen amount of iterations these duties cannot be moved again. Now it is sure the duty will not return to the group it recently originated from. However this is not enough to prevent the solution to return to a near identical situation. This is caused by the fact that it can be possible that there are duties with identical statistics as the duty just put on the duty tabu list. When such duties are moved the solution can still return to what it was before. To prevent this from happening, when a duty is put on the duty tabu list all duties with identical statistics are put on the duty tabu list as well. With all these duties on tabu it is ensured that the solution will not return to the exact same combination of statistics.

Just like with the group tabu, in the experiments we test many different values for the amount of iterations a certain duty will be kept on the tabu list.

Section 4.7 Termination Criteria

In this thesis we use 3 termination criteria. The first criteria used is that every run will stop after a fixed amount of iterations. This is simply done to prevent the search to take too long when the other termination criteria are not set correctly. In general this restriction is not needed because of the second criteria to stop after a sequence of non-improving iterations. This is most often the reason the search stops.

The third criteria being used is the one for specific difficult to solve problems. In some cases it proved to be extremely difficult to fit in all duties and satisfy all constraints at the same time. In these kind of cases the termination criteria used is to stop when a feasible solution was found.

Section 4.8 Advanced Techniques

4.8.1 Intensification & Diversification

To allocate duties to roster groups we have a large search space available to find a good solution. Because the tabu-search meta-heuristic we use is based on the steepest ascend method it is often the case that the solution ascends fast at the beginning and later on improves very little or not at all. In this case the search is in an area near a local or global optimum.

When a problem has one or more local optimums the tabu-search often ascends to one of them and stays there for a large amount of iterations trying to find the local optimum. The tabu-search intensifies its search in this case. Because the state of the search is close to the local or global optimum, all of the neighboring solutions will only have a small improvement or none. However there is no measure whether the search is near a local or global optimum. For this we use diversification when going through the search space. Diversification means that instead of looking more closely at a small set of solutions, the search will look at a greater set of solutions. It forces the search to also look at other places in the search space where it has not looked yet.

In this way we cover more of the search space and greatly increase the possibility of finding a global optimum as opposed to the local optimums.

To force the search away from its current position we assign higher penalties on the searching in roster groups which have been on tabu in past. When the search intensifies a small number of roster groups are used to look for improvements. If a combination of roster groups does not yield anymore possible moves, one of the two groups will be put on a group tabu list. The tabu being put on this group is set for a fixed number of iterations. This means that if the search is still looking in the same area when the tabu marked group is released from the tabu list, the group is likely to be considered again. It is also likely that this group will quickly be put on the tabu list again after its been chosen again because of a lack of possible moves. Therefore, the search will not leave its current search area and will spent many iterations in this area.

In order to reach the parts of the search space which have not been considered yet we penalize groups for getting on the group tabu list multiple times. For this we keep track of how many times a group has been on the group tabu list in the past. The amount of iterations the group stays on the group tabu list will now be a fixed number, coming from experiments, multiplied by the number of times that the group has been on the group tabu list in the past. For instance, instead of putting a group for 10 iterations on the group tabu list each time, it will stay on the list for 10 iterations for the first time, 20 iterations for the second time it is put on the list, 30 iterations for the third time etc.

A problem that arises with this adjustment is that the list that keeps track of the tabu history of each group is never reset. Once the search is pushed away by not allowing to select groups on the tabu list for a long time, the high amount of tabu time remains and the search can hardly ever return in the direction it came from. For example the search is at an area with an assumed local optimum. It is forced to move away from it. The neighboring area which will be investigated next might be worse than the area you were in, before moving away. However the tabu on the groups which were used before are so high that these groups are nearly excluded for the rest of the search. To solve this problem the tabu for a group will be reset when its tabu time becomes too long.

By releasing the tabu for a particular group we allow the search to intensify again. In this way the search can continuously alternate between intensifying and diversifying leaving, eventually, no part of the search space untouched.

4.8.2 Allowing Infeasibility

We will now look into infeasibility and explain the importance of allowing infeasible solutions. Basic heuristics like steepest ascend update their found solutions only if it improves its current best found solution. While this leads to fast solutions, the quality of the result is not always desirable. Basic search methods like the one just mentioned are known to often get stuck in local optimums. In chapter 3 we explained that the strength of a tabu-search meta-heuristic is in the way it can get out of local optimums and continue searching for global optimums.
We continue the search for better solutions, after finding local optimums, by allowing the solutions to get worse, including infeasible solutions.

Feasibility is what we desire and infeasibility is undesired but without allowing infeasibilities during the search, it might be difficult to find a good solution. Sometimes a better solution is only accessible after moving from a feasible solution to an infeasible one. By adding infeasible solutions to the search space there will be a lot more solutions to search through, increasing the chances of finding better solutions in the long run.

To prevent that the search only looks into infeasible solutions or gives an infeasible solution as end result we introduce a penalty factor. Whenever an infeasible solution is generated a penalty will increase the objective value making the solution less desirable. In this way it is likely that during the process at least one feasible solution is found.

This section described that allowing infeasibility can lead to solving more complicated problems. However we do not allow every restriction to be violated. Of all constraints we have we only allow violations of a few constraints. These are the average duty length, the amount of duties with a length longer than 9 hours, the amount of duties with a length shorter than 5 hours and the amount of nightshift duties.

The restrictions on the amount of duties that may be done on a single day must be obeyed at all times.

4.8.3 Controlling the search – putting a limit on decrement

One of the biggest improvements of tabu-search over a simple steepest ascend method is allowing the solution to decree. In this way the search can attempt to avoid getting stuck in a local optimum like we explained in chapter 3. There is however no limit on this decrement. Therefore it is possible that the search decrees so far away from a previous found reasonable solution that it can hardly ever return to the better solution.

To help guide the search towards a solution we limit the amount of decrement. This means that we allow the solutions to decree but not more than a fixed percentage. By doing this we do not consider all possible solutions that can be made by moves which clearly do not improve and will not lead to improvement in a near future.

Besides limiting the decrement we also do not want to decree for too long. Every time we find a better solution we store this solution. This gives us information of the best found solution when the search has finished but that should not be the only place where we make use of this source of information.

To enhance the strength of the tabu-search we also put a limit on the amount of non-improving iterations. Whenever the search does not improve after a fixed amount of iterations the search will be put back in the situation it was when it found the currently best found solution. This way we can quickly return to a better solution when the search has gone in the wrong direction for too long.

4.8.4 Resetting to the best found solution

The tabu-search we apply has several memories. One of its memories is the best found solution. This memory will be updated every time the search finds a better solution than the best one it found in the past. This information can however be useful for the tabu-search in another way. When the search has found a solution, in its following iteration there will be many new possible solutions. The next found solution as a result of one of many moves takes the search in a direction which hopefully leads to solving the problem. The problem that presents itself is that it can happen that the search goes in a bad direction, worsening the solution, but is not able to get back to where it once was with a better solution. Even if the search only has 2 possible moves at every iteration it will happen that at some iteration it would have been better to choose the other option. Because of the tabu lists, diversification and intensification and other rules the chances of getting back to the solution where that other options would have been better are small.

For this we keep track of how many iterations it has been since the solution improved. If the solution has not improved after a set amount of iterations the entire search will return to its last found best solution. This criterion is the same as the third termination criterion explained in chapter 3, only now we do not use it to terminate the entire search but to set the search back to a previous promising solution.

Section 4.9 Parameter settings

The tabu-search algorithm has six parameters which need to be set to a fixed value. All the values that have to be set are:

1. the amount of iterations a group will be put on tabu *see section 4.6.1*,
2. the amount of iterations a duty will be put on tabu *see section 4.6.2*,
3. a number for the chosen stopping criteria if it needs one *see section 4.7*,
4. the amount of extra penalty when diversifying *see section 4.8.1*,

5. the limit for decrement *see section 4.8.3*,
6. the amount of iterations before the solution will be reset to a previous promising solution *see section 4.8.4*

All of these values are very important and changing them can have great impact on the final result. Because the parameter values are so important a big part of the research is to find values that work well, not just for a single case but generally good values.

To find values which give good results we run a lot of tests with a wide range of values. This is because as explained before, there is no rule for how to set the values. Therefore we use trial and error to analyze the effect the values have on the performance of the search. At first extreme values like zero iterations tabu time and a very large number of iteration tabu time. From there on we either increase or decrease the values till a good performance of the tabu-search is found.

Chapter 5 Computational Experiments

Section 5.1 Data description

The data we use is given by the company NS. There are data for all the duties which have to be allocated to the groups and we have data for the groups so we know the restrictions for each group.

We will first explain the duty information. After that we will describe the group information.

After the data descriptions we will show and explain the objective function we used.

5.1.1 Duty information

For each base in the country there is a set of duties which need to be assigned to roster groups. Each set varies in size. For instance Roosendaal2008 has 297 duties while Utrecht2009 has 707 duties.

Each duty has the following characteristics:

1. Base,
2. NS#, Weekday,
3. V/L/N,
4. Duty length, Start time, End time, Driving time,
5. %A-work, %Aggression-work, %Double deck-work,
6. A list of all routes with a corresponding 0 or 1 for each route

The base represents the station where the duties start and end. The NS# is a number NS gives to different duties. This means that duties can have the same NS# but differ in which weekday they are performed. The number for weekday, varying from 1 to 7, simply expresses for which day of the week this duty is meant with 1 being Monday and 7 means Sunday.

V/L/N means whether the duty is of the category early (V), late (L) or night shift (N).

Next we have the duty length which is the total time a duty takes. The duty length can vary between a minimum of 4 hours and a maximum of 9.5 hours. The total time the crew member actually spends working is given by the driving time. This amount of time is smaller than the duty length because it does not include meal and other breaks and the required time to stop at each station to allow passengers to get off and on board. We also have the starting time and ending time of the duty.

Characteristics 2,3 and 4, as explained on previous page, together give the information we need to deal with the labor rules *(see chapter 2)*.

Following the time information is the popular and unpopular categories information. For every duty we know how much percent of its time the duty involves A-, Aggression- or Double deck-work.

Finally, for each duty we have an overview of how many times a particular route is used in the duty.

Characteristics 5 and 6 give us the data we need to deal with the sweet and sour rules *(see chapter 2)*.

5.1.2 Group information

Group name	Type	Mo	Tu	We	Th	Fr	Sa	Su	Start time	End time
A	V	2	2	2	2	2	1	1		
	L	2	2	2	3	3	1	1	6:30	23:30
B	V	5	5	5	5	5	2	2		
	L	5	5	5	4	4	3	3	4:00	2:00
C	V	3	3	3	3	3	2	1		
	L	4	4	4	4	4	3	3	4:00	2:00
D	V	6	5	5	5	5	3	2		
	L	5	5	5	5	5	3	3	4:00	2:00
V	V	8	8	8	8	8	5	5	4:00	18:00
L	L	10	10	10	10	9	6	7	10:00	7:00
Z	B	100	100	100	100	100	100	100	4:00	7:00

Table 2: Group information

An example of restrictions for the groups is shown in table 2. Each group has its own name and own restrictions. Note that group Z is the artificial group which we talked about in section 4.1.

The "type" column shows whether a group may perform early (V), late (L) or both (B) types of duties. Next are the amount of duties that are allowed to be in every group on every day. For example, group A may only have 2 early duties for Mondays. Last are the intervals in which the duties must lie for each group. Each duty may not start earlier than the start time and may not end later than the end time.

5.1.3 Objective function

In our objective, we try to minimize the deviation in the different 'sweet and sour'-rules from the average. For example if the average aggression-work is 50% we give a penalty to all groups with more than 50% aggression-work. The same goes for double deck-work because both aggression-work and double deck-work are considered unpleasant. That is why too much of it is unwanted and gets a penalty. For A-work it is the other way around, this is a positive category and therefore only gets a penalty when there is too few of it in a group.

The objective function is given by:

$$Z_i = \max\left(\overline{A} - A_i\,, 0\right)w_A + \max\left(Agres_i - \overline{Agres}\,, 0\right)w_{Agres} + \max\left(Ddeck_i - \overline{Ddeck}\,, 0\right)w_{Ddeck} + \sum_l \left(\left|t_{li} - \overline{t_l}\right|\right)|p_l|$$

$$\min \sum_i Z_i + 2\max_i Z_i$$

with:

i = 1,...,n with n the number of groups

A_i = total amount of A-work time in group i
$Agres_i$ = total amount of Aggression-work time in group i
$Ddeck_i$ = total amount of Double deck-work time in group i

$\overline{A}, \overline{Agres}, \overline{Ddeck}$ = the average amount of time the category is present in all duties of the case

l = 1,...,m with m the number of routes

w = weight belonging to the category

t_{li} = the total amount of times route l is present in group i

$\overline{t_l}$ = average amount of times route l is present in all duties of the case

p_l = penalty belonging to route l, this can be either a positive or a negative penalty

In some cases, we also allow violation of some of the constraints. These violations will however, still be given a penalty to show whether it is a small violation or a large one. An example is the violation of the restriction that at most 5% of the duties may be of length shorter than 5 hours.

Section 5.2 Finding a balance between speed and performance

The way we set the parameters has a huge impact on the outcome of the tabu-search. This was the first and most important result which showed after hundreds of testing runs The parameters presented in section 4.9 as well as how we set the objective function and the penalties for violating restrictions can be set in such a way that the tabu-search performs fine but a small adjustment can also turn it into a disaster.

We will now explain the effect of the different parameter settings.

In general for any case we tested the tabu-search for, we found that the speed with which we can solve a case is negatively correlated with how good the final result is or even the probability that a feasible solution can be found.

5.2.1 Resetting the solution

The correlation between speed and performance shows best when we set the amount of iterations before the solution will be reset to a previous promising solution, to a low value. If for instance we let the search reset after 10 iterations then the search will have only 10 iterations of time after every reset of finding a new improvement. When all pieces of the puzzle fall in place a fast reset can result in finding a solution very fast. The problem here however is that all possible improvements which need more time than 10 iterations to be found, will not be used. In this way the amount of possible solutions gets cut down a lot which changes the search procedure from a tactical search to a 'hope we get lucky' search.

In figure 3 we show, for one of the many test run we did, how many iterations it took before an improvement was found. In this example the reset value has been set to 60. From these results are only those shown which needed more than 10 iterations. These results are afterwards rearranged from low to high values.

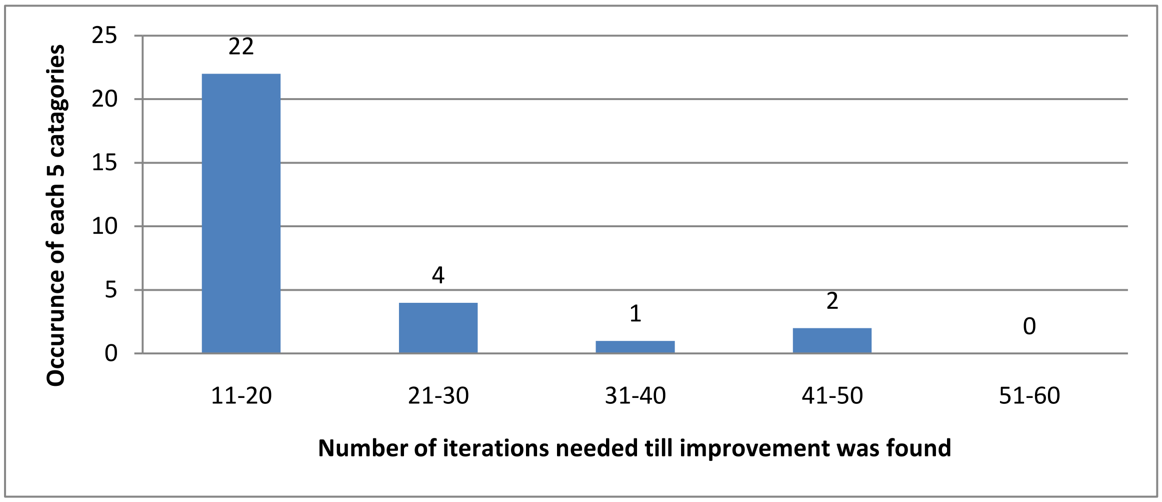


Figure 3: Graph of number of iterations till improvement

Clearly it shows that when the reset value is set too low, the search will not be able to perform possible crucial improvements. For instance if we had set the reset value to 20 we would have missed out on 7 possible crucial improvements. When we would have set it to 40 we would have only missed 2.

From the many test runs we did we may deduct that the form of the graph in figure 3 generally holds true. The tabu-search will always have the highest frequency of finding an improvement in the lower bracket, where a low amount of iterations are needed. The times an improvement was found after many iterations declines rapidly. This information can help set the value of the reset parameter. For the test run of figure 3 a reset value of 50 would have been ideal, however in other test runs it did happen that the frequency of the last bracket, 51-60, was greater than zero. Therefore a reset value between 50 and 60 would be the most useful. For our case studies in chapter 6 we use a reset parameter set to 60.

5.2.2 Stopping criteria

Stopping the tabu-search can be a case of having found a feasible solution or stopping after an amount of non-improvements. What we found was obviously that the longer the search is allowed to continue looking for improvement the better the end result gets.
Stopping after 100 iterations gives less result then stopping after 1000 iterations, although the effect of continuing when the current result is already very good diminishes fast.

This parameter is however quite save to change. A change will not cause the search to be unable to find a solution; it simply is a great tool to allow a choice between speed and performance.

Section 5.3 Tabu parameter influence

The tabu parameters control the memory of the tabu-search. As explained in chapter 4 these parameters are needed for progression. Without the tabu on groups and duties the search can reverse to a situation it was just in. Reversing what just happened is very bad for finding a solution. A value of zero iterations for the tabu parameters is therefore very undesirable.

We found what eventually worked best for the tabu parameters by trial and error. However the range of values to test could be narrowed down because of 2 important findings.

A tabu parameter set **too low** will cause the same problems as no tabu at all. With a low value it is often the case that the search returns to a solution already found in the recent past. In this way the search continues to go in circles. Especially in combination with a low value for the 'resetting of the solution'-parameter the search will keep repeating its previous steps without getting any closer to a solution of the problem.

On the other hand a tabu parameter set **too high** causes a problem opposite of what happens when the tabu parameter is set too low. Instead of the same duties or groups being available frequently the amount of choice changes to nearly none. The duties and groups put on tabu will take so long to be removed from the list that the tabu lists will increase in size so fast that soon all duties in a group or simply all groups are tabu. When this happens the tabu-search cannot move on anymore, it is stuck and with a proper termination criteria the search will end very early without a good solution.

Although, as explained, it is obvious that the tabu parameter should be set with a value higher than zero there is no way to find below which value it should be. The way we found good working values was to start with a high value and gradually decrease it.

Section 5.4 Approaching the problem as a whole

The beauty of the tabu-search we created is that it solves the problem as a whole. It does not solve in a staged manner in a way like solving one of the constraints first and then the other etc. All restrictions are being solved at the same time as well as looking for a better objective value.

What is important to achieve this goal is to keep the values of the objective function and the penalty costs for violation of restrictions not too far from each other. A problem which occurred during our testing showed that if either the objective function or the penalty costs are much higher compared to the other, the tabu-search has a hard time to find a solution. It is often the case that when the objective value improves that it comes at the cost of a restriction being violated more or when a violation of a restriction gets closer to being solved that the objective value gets worse. If for instance the penalty costs for having too much average duty time in a group would be much higher than the weights of the objective function of the problem, the tabu-search would only focus on getting the restriction solved, neglecting nearly all possible improvements for the objective function. This is a big problem because at some point the search will not be able to improve the violation of the restriction anymore and will still not consider any improvements for the objective function because of its low weight.

The most important part of the tabu-search is to keep searching for improvements and in as many areas as possible. Emphasizing on a difficult to solve restriction or trying to solve the problem one restriction at a time only ensures that a local optimum is found with the search ending prematurely.

Section 5.5 Robustness

As has been explained the tabu-search we build has a lot of settings. Every change made to the tabu parameters has an impact on the results. With some parameters, like the tabu time parameter, a change can mean the difference between a solved problem or no feasible solution at all. For other parameters, like the stopping criteria, it is a fine tool to choose between speed and performance explained in section 5.2.

The amount of freedom the user of the tabu-search has is what we call the robustness of the program. The user can choose for instance to stop searching after half an hour for some fast results or keep searching for a longer period of time to get better results. When the problem is a difficult to solve case the user can choose to stop the search as soon as a feasible solution has been found when any feasible solution will suffice instead of having to know the best solution.

Not only the parameter settings are tools for the user, the objective function can be changed to its liking as well. For our railway duty assignment problem we have put greater weights on the fair allocation of routes then we did with the fair allocation of the sweet-and-sour rules. This way we could give hearing to the outcry of the crew for more variety in their work.

Chapter 6 Case Studies

In this chapter we will apply our tabu-search to real cases. We will show how well it can perform compared to solving the same problem with the integer programming solver Cplex. Besides complete solving of the cases we will also show the influence of drastic changes in different parameter settings.

For each case we will show in bold the best found objective values and for the Utrecht 2008 case also the least amount of violation or the least leftover duties.

We also calculate a lower bound with the help of Cplex and an Linear Programming relaxation for each case.

For each case we use the same parameter settings. These will be called the standard settings and are:

- Duty tabu: 10 iterations
- Group tabu: 20 iterations
- Reset to best found solution: 60 iterations
- Limit on decrement: 20%
- Stopping criteria: after *300* iterations of no improvement

The cases we will apply our tabu-search on and their characteristics are as follows:

Case	Roosendaal 2008	Utrecht 2008	Utrecht 2009
Groups	6	15	16
Duties	297	641	707
Stopping criteria	Standard	Standard	First to occur of standard & after a feasible solution is found

Table 4: Characteristics of the 3 cases we solved

We will first show the results for the fairly easy to solve Roosendaal 2008 case. There are no difficulties for this case and is therefore a great case to see the effect of a change in the parameter settings.

Section 6.1 Roosendaal 2008

At first we will run the case 5 times with the standard settings as written at the start of this chapter. We will keep track of the time it takes to solve the case and the best objective value that has been found at the end of the search. Because of the randomness of the search, the random group selection and the random tabu, the results will vary each run. We therefore run each variation of the case 5 times and take an average of the results. In the third column, besides the time and objective value, we will report how many duties are left at the end of the search. In other words, how many duties could not be assigned to a group.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	1106	12,783	0
2	1334	**12,575**	0
3	1283	14,914	0
4	1044	16,977	0
5	1446	13,931	0
Average	1243	14,236	0

Table 5: Tabu-search results of 5 runs

Next we run the same case again only this time we stop as soon as we find a solution that does not violate any restriction.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	104	42,515	0
2	96	49,560	0
3	136	**21,786**	0
4	120	39,100	0
5	87	34,893	0
Average	109	37,571	0

Table 6: Tabu-search results of 5 runs with stopping criteria: feasible solution found

Here we see that finding a feasible can be done more than 10 times faster than with the standard settings. However this gives around 2.5 times as worse solutions. The trade of between speed and performance is clearly visible here.

We also applied a solver program called Cplex to an integer programming model of the problem. 1 run was done. Its solution after half an hour and an hour were noted. However there were no further improvements found after half an hour.

Cplex run	Time(sec)	Objective value	Leftover Duties
1	1800	12,780	0
2	3600	12,780	0

Table 7: Cplex results of 2 runs

Lower bound:	*Leftover Duties*
3796	0

Table 8: Cplex lower bound Roosendaal 2008

To compare the results of the Cplex program with our tabu-search we run the case again, this time with a stopping criteria of 1 hour of running.

Tabu-search run	Objective value	Leftover Duties
1	12,632	0
2	13,561	0
3	**12,211**	0
4	12,491	0
5	13,280	0
Average	12,835	0

Table 9: Tabu-search results of 5 runs with stopping criteria: 1 hour of running

On average the results are slightly worse but it is worth noting that with 3 of the 5 runs the end result was better than the Cplex result. It is therefore advised to run the tabu-search simultaneous on several computers and take the best result. Also table 4, run 2 shows that a good solution does not have to take an hour to find.

The following 3 tests on the case of Roosendaal 2008 will be to see the effects of a major change in the reset parameter and the removal of the limit on decrement.

We will first set the reset parameter to 10 speed up the process but give less room to find improvements. After that we will do the opposite, increase the reset parameter to 100 to slow down the process and give it much more room to find an improvement.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	1640	**14,143**	0
2	529	15,200	0
3	1210	16,075	0
4	1433	15,104	0
5	1090	14,371	0
Average	1180	14,979	0

Table 10: Results with reset parameter set to 10

Here we see that compared to the runs with the standard settings the search does speed up but not by much. It also finds worse solutions which is caused by the low amount of iterations the search is allowed to look for improvements before it gets reset.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	453	19,057	0
2	845	16,154	0
3	1122	**13,039**	0
4	501	14,730	0
5	1149	15,289	0
Average	814	15,654	0

Table 11: Results with reset parameter set to 100

With the results of a very high reset parameter we see something strange happening. The search does not slow down but speed up, even more so then the low reset value. This is caused by the fact that with stopping criteria of no improvements after 300 iterations combined with a reset value of 100, it gives the search only 3 resets before it terminates. If with 3 resets no improvement was found the search ended which could happen fairly fast. However 2 out of the

5 runs took more than a 1000 seconds and still did not find a good solution. The average of the results also shows that this way of choosing the reset value is a bad choice.

The results of the low reset value proofs the explanation given in section 5.2 which told that a low value can give a fast result but it will most likely not be a good result. The high value shows that with the current standard settings the value is just too high to be useful.

The last test we run for this case is to see the effect of not having a limit on the amount the solution value may decree.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	706	**14,488**	0
2	592	15,072	0
3	933	16,006	0
4	1303	16,388	0
5	1078	15,662	0
Average	922	15,523	0

Table 12: Results without a limit on decrement

What happens now is that many iterations get wasted to look into an area of the search space that already is far away from the best found solution so far. Instead of looking at solutions which are a maximum of 20% worse we now allow the solution to be 5 times as worse. What often happened with these runs is that a solution worsened so much that it was impossible to improve or even get back to where it was. A lot of the search happens in the areas where we do not want to look. Because of this the search terminates early, resulting in a faster solution speed but worse objective values. Again this shows the negative correlation between speed and performance.

Section 6.2 Utrecht 2008

The Utrecht 2008 is a more difficult case than Roosendaal 2008. It has more groups and a lot more duties. Besides the larger size of this case there is also a special characteristic which the Roosendaal case does not have.

For the Utrecht 2008 case **it is not possible to solve the restriction of duties with a length shorter than 5 hours**.

We start this case off with the Cplex solutions followed by tabu-search results with the standard parameters.

Cplex run	Time(sec)	Objective value	Leftover Duties
1	1800	161,000	31
2	3600	161,000	31

Table 13: Cplex results of 2 runs

Lower bound:	*Leftover Duties*
18,688	1

Table 14: Cplex lower bound Utrecht 2008

From the lower bound we can see that this case cannot be solved. The Cplex results show the difficulty of the case by still having 31 duties unassigned at the end of a 1 hour run.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	2758	**47,603**	5
2	1840	56,114	6
3	2477	53,855	5
4	1948	60,953	4
5	3899	48,597	**3**
Average	2255	53,962	5

Table 15: Tabu-search results of 5 runs

Even with the tabu-search there are still duties not assigned when the search terminates.

With the Cplex program the emphasize lies on not violating any restrictions at the cost of not being able to allocate many duties. For testing the tabu-search we thought there was a better way to address the problem. All of the duties have to be performed and therefore we switch the emphasize around. We put want all duties to be assigned and afterwards violate any restriction the least possible. For this case it means that all restrictions will be met except for the restriction of the duties with a length shorter than 5 hours which will be violated in the least way.

For example, if at the end of the search group 1 has 5.5% of its duties with a length shorter than 5 hours and the same holds for group 2 while all others are at 5% or below, then the total violation will be 0.5% + 0.5% = 1%.

We will now test the effect of different parameter settings on the problem with all duties assigned and priority on reducing the total amount of restriction violation as much as possible.

A lower bound for this setting is given by:

Lower bound:	*Leftover Duties*	*Total Violation*
14,621	0	**13.6 %**

Table 16: Cplex lower bound Utrecht 2008, second setup

Tabu-search run	Time(sec)	Objective value	Total Violation
1	2660	49,352	2.46 %
2	2763	49,284	**2.18 %**
3	1830	54,618	2.25 %
4	2880	55,821	2.83 %
5	3734	**48,527**	2.22 %
Average	2773	51,520	2.39 %

Table 17: Tabu-search results of 5 runs

The difference in results between the 2 priorities are not much noticeable. Around the same speed and objective value. However we believe that assigning all duties is much more desirable because every duty simply must be done. This means that the crew otherwise has to work overtime to get the unassigned duties done or in the worst cause the duties cannot be performed at all.

Tabu-search run	Time(sec)	Objective value	Total Violation
1	4565	47,056	2.53 %
2	3316	42,375	3.45 %
3	5686	**39,196**	**2.47 %**
4	5825	47,666	2.57 %
5	4543	51,543	3.81 %
Average	4787	45,567	2.97 %

Table 18: Results with reset parameter set to 10

For the Utrecht 2008 case we see here that a strong reduction in the reset parameter results in an inflation of the run time of the program. This can be explained by the fact that it is a lot easier for the tabu-search to find in 10 iterations an improvement for the objective value than it is to find an improvement for the violated restriction problem. This means the focus of the search shifts to getting a better objective value instead of reducing the violation of the restriction as much as possible. As explained in section 5.4, it is never good to focus on 1 part of the problem only. While the total amount of violations stays high it is easier to find improvements for the objective value. As a result the objective value goes down but the total amount of violation goes up.

Tabu-search run	Time(sec)	Objective value	Total Violation
1	2396	55,665	**2.36 %**
2	1792	58,211	2.63 %
3	2060	**35,661**	2.56 %
4	1211	39,442	3.46 %
5	1800	53,913	2.90 %
Average	1852	48,578	2.78 %

Table 19: Results with reset parameter set to 100

The opposite is true for a very large value for the reset parameter. Because the search has a lot more iterations to look for improvements it tends to find an improvement more easily for the violations of the restriction which has priority in the search. As a result the total amount of violation goes down and the objective value goes up, compared to a reset value of 10. For the exact same reasons as with the Roosendaal 2008 case the run time of the program goes down.

We proceed with test runs without a limit on decrement.

Tabu-search run	Time(sec)	Objective value	Total Violation
1	3538	54,005	**2.28 %**
2	2334	50,943	2.46 %
3	1616	51,540	2.62 %
4	3596	**49,194**	2.46 %
5	2751	50,499	2.37 %
Average	2767	51,236	2.44 %

Table 20: Results without a limit on decrement

Contrary to the results of the Roosendaal 2008 case, the removal of the limit on decrement did not change much for the Utrecht 2008 case. The amount of time stayed about the same, the objective value stayed nearly the same and the total amount of violation got slightly worse but not significantly. However if we look at the best found solutions we see that both in objective value as well as in total violation there was a better found value with the standard parameter settings. So if we were looking for the best it would still be better to maintain the standard settings.

For the Utrecht 2008 case we did not perform 1 hour runs because most runs were already around the 1 hour mark and even with less time the tabu-search already greatly outperforms the integer programming solver Cplex.

Section 6.3 Utrecht 2009

The Utrecht 2009 case is much like the Utrecht 2009 case in means of size. However a few adjustments were made. These are:

- restriction for the amount of duties allowed to have a length shorter than 5 hours increased to 10%
- restriction of an average duty length not greater than 8 hours increased to 8.5 hours for group 16 only

Unlike the Utrecht 2008 case, the Utrecht 2009 case can be solved with all duties assigned and no restrictions violated. This does not mean the Utrecht 2009 case is an easy one to solve. Especially **the restriction of the average duty length was found to be very hard to satisfy**.

For our first test with the standard settings we choose to do twice as many runs as the previous 2 cases. This is because the Utrecht 2009 case cannot always be solved. To get a more reliable view of the tabu-search success rate we do 10 runs.

Tabu-search run	Time(sec)	Objective value	Leftover Duties
1	1456	160,512	0
2	1246	152,173	0
3	2103	157,532	0
4		NO	
5		NO	
6	1687	**147,954**	0
7	1195	170,030	0
8	1497	148,154	0
9	823	155,113	0
10	1260	157,533	0
Average	1408	156,125	0

Table 21: Tabu-search results of 10 runs

A NO means that no feasible solution could be found given the stopping criteria.

2 out of the 10 runs failed to find a feasible solution. This means that the tabu-search has an 80 percent success rate for this case.

Cplex run	Time(sec)	Objective value	Leftover Duties
1	1800	122,093	2
2	3600	122,192	1

Table 22: Cplex results of 2 runs

Lower bound:	*Leftover Duties*
101,000	0

Table 23: Cplex lower bound Utrecht 2009

Proceeding the test runs with the standard settings were the test runs with a low and high reset parameter. Because with both settings no solutions were found at all there are no tables for these tests.

The problem that clearly showed when the reset parameter was set **too low** is that the focus of the search changes to almost only the objective value. This can be explained by taking the special characteristic of the Utrecht 2009 into account, the average duty length restriction. In this case it is difficult to not violate the average duty length restriction. However when the tabu-search proceeds there are most of the time between 2 and 4 groups violating this restriction. This means that this restriction can only be solved if at least 1 of those groups gets selected to search in. On the other hand, the objective value could be improved with all groups. What makes it even worse with a low reset parameter is that even if a group with a violation of the restriction is selected, the tabu-search gets only 10 iterations of time to find an improvement. In a difficult to solve case like Utrecht 2009 those 10 iterations are not enough. This further proves the explanation in section 5.4 that emphasizing on 1 part of the problem does not work.

The results when using a **too high** reset value were not much better than the low reset value. Out of 5 runs only 1 solution was found in 1673 seconds with an objective value of 159,047 and no duties leftover. What happens is that the search gets a lot more time to look for improvements but its possibilities to reset are very few. It does show that it is able to find a solution but it is just a lucky one. It does however mean that using a reset parameter of 100 is better than using one of 10 but the results are still far from desirable. Whether the results speed up or slow down cannot be concluded from these results but they do comply with section 5.4 that a **too high** reset value is not a good choice.

Next we did tests with no limit on decrement. Just like the reset value set too high the 5 runs we did only gave 1 solution. In 3224 seconds an objective value of 156,042 was found with 0 duties leftover. Again its effect on the results differ from the other cases. For the Roosendaal 2008 case we saw that it speeds up the search but with decreased quality of the objective values found. For the Utrecht 2008 case it resulted in no significant effect. However for the Utrecht 2009 it does have an effect on the results. The results resemble those of the runs with a reset parameter of 100. With no limit on decrement the search is allowed to keep looking in a part of the search space that is already far from the best found solution so far. The result is that the search wastes a lot of iterations. This clearly leads to a greatly lowered success rate of finding a solution. There is also a big increase in the time needed to find its only solution compared to the only solution found with a reset value of 100. This however we cannot be certain of with just one found solution. The main result here is that the limit reduces the possibility of finding a solution.

Finally we ran the tabu-search with its standard parameters with the exception of stopping the search after 1 hour of searching.

Tabu-search run	Objective value	Leftover Duties
1	140,446	0
2	147,068	0
3	130,016	0
4	**129,597**	0
5	140,195	0
Average	137,464	0

Table 24: Tabu-search results of 5 runs with stopping criteria: 1 hour of running

The conclusion we can draw from these results is that giving the tabu-search more time to look for improvements leads to better results in the form of lower objective values. This has everything to do with speed versus performance as we explained in section 5.2. Given time restrictions, a single run of an hour may take too long but from these results we know that if the goal is to find lower objective values and there is more time at hand that it is possible to find better solutions.

Chapter 7 Conclusion & Recommendations

Section 7.1 Conclusion

We have developed a tabu-search meta-heuristic solution method for the railway duties assignment problem. The problem addressed was a pre-rostering phase in which a large number of duties needed to be allocated to different roster groups. Each roster group consisted of a number of crew members with the same work preferences. Several restrictions had to be satisfied making the allocation a difficult problem.

The method was designed, created and tested on 3 different test cases. It was also compared to an already present integer programming solver, Cplex.

The results showed that the tabu-search is capable to solve an easy case, Roosendaal 2008, fast and effective. It showed that it had no problem solving a difficult case, Utrecht 2008. Even though the case was technically impossible to solve because of no possible feasible solutions, it could still allocate every duty with keeping the violations of any restriction to as low an amount as possible. The last results of the case Utrecht 2009 showed that a complicated problem, because of a very hard to satisfy restriction, can still be solved effectively but not always.

We also showed that a tabu-search solution method is very sensitive to how its parameter settings are chosen. In all 3 cases we tested different variations of parameter settings to show its effect on the quality of the solutions. It revealed that tabu parameters set too low means that the search starts to cycle through solutions it already found in the past and if we set it too high the search space gets too small to find any new improving solutions in.

Besides the tabu parameters we found that resetting the search to a solution found in the past which was much better then where the search was searching afterwards, is a good thing. However resetting too fast proved to decrease the quality of the solution or even resulted in no solution at all. The opposite, giving the search a long time to search for improvements before resetting, resulted in faster results but of less quality.

The last thing we tested was a prevention mechanic so the search could not worsen the current best found solution by more than a certain percent. This proved to cut down on searching in areas of the search space which do not hold improvements at all resulting in a higher quality of the solutions found. This holds for the Roosendaal 2008 case but it did not show as clearly for the Utrecht 2008 case.

A direct comparison with the already present integer programming solver Cplex, showed that after a time period of 1 hour, on average the Roosendaal results were slightly worse and the results of both Utrecht cases greatly outperformed the integer programming model. Note however that for the Roosendaal case table 9, run 3, proves that the tabu-search can find a better solution than the integer programming model.

Section 7.2 Recommendations

We have explained in section 5.5 that we have created this tabu-search solution method with keeping robustness in mind. This means that the user has the freedom to adjust the method to its likings. Like we did for the case Utrecht 2008 it was easy to change the focus of the tabu-search. We changed it from a focus on no violation of any kind and trying to assign as many duties as possible to the least amount of violation while still assigning all duties.

It is also easy to change weights in the objective function to make the different parts of the objective function more or less important. For instance the importance of a fair distribution of popular and unpopular work can be made more important than the variety in work for the crew. This change would be a welcome one in our opinion because the current Cplex solution method does indeed value the variety in work more than the fair distribution of popular and unpopular work. The difference in weight is so large that the fair distribution of the popular and unpopular work is hardly a concern when looking for a good solution. However to get a fair comparison with the Cplex program we could not adjust the objective function. Given the above we would advice to lower the weight of pursuing variety in work for the crew.

Besides the change of objective function weights the user is also free to adjust the time it wishes to spent on looking for a solution. It has been shown in chapter 6 that the tabu-search performs better if it is given more time to search.

Our last advice comes from the fact that the tabu-search involves randomness. As shown in the results of chapter 6 the solutions found vary each run. Our tabu-search can find better solutions than solving an integer programming model with the Cplex program, but this does not hold for every run. We advice to run the tabu-search on several machines and use the best solution.

References

[1] Behdinan, K. & Perez, R.E., 2007. Particle swarm approach for structural design optimization. *Computers & Structures*. 85(19-20), pp. 1579-1588.

[2] Blum, C., 2005. Ant colony optimization: Introduction and recent trend. *Physics of Life Reviews*. 2(4), pp. 353-373.

[3] Eglese, R.W., 1990. Simulated annealing: A tool for operational research. *European Journal of Operational Research*. 46(3), pp. 271-281.

[4] Gendreau, M., 2003. An introduction to tabu-search. *Handbook of Metaheuristics*, New York: Springer. Ch. 2. pp. 37-54.

[5] Glover, F., 1990. Tabu-search: A Tutorial. *Interfaces*. 20(4), pp.74-94.

[6] Glover, F. & Laguna, M., 1997. Tabu-search. New York: Springer.

[7] Hartog, A., 2005. Wiskundige modellen voor het maken van dienstroosters voor het treinpersoneel (in Dutch). Master thesis, Free University of Amsterdam.

[8] Hartog, A. & Abbink, E.J.W. & Huisman, D. & Kroon, L.G., 2009. Decision Support for Crew Rostering at NS. *Journal of Public Transport, 1*(2).

[9] Hertz, M. & Taillard, E. & de Werra, D., *A tutorial on tabu-search*. Available at: http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf [Accessed April 2008].

[10] Kroon, L. & Huisman, D. & Abbink, E. & Fioole, P.J. & Fischetti, M. & Maróti, G. & Schrijver, A. & Steenbeek, A. & Ybema, R., 2009. The new Dutch timetable: The OR revolution. *Interfaces*. 39(1), pp. 6-17.

[11] Laporte, G., 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research.* 59(3), pp. 345-358.

[12] NS Reizigers (in Dutch). Available at: http://www.ns.nl/cs/Satellite/over-ns/organisatie/reizigersvervoer/ns-reizigers [Accessed April 2008].

[13] Salhi, S., 2002. Defining tabu list size and aspiration criterion within tabu-search methods. *Computers & Operations Research.* 29, pp. 67-86.

[14] Sridharan, R., 1995. The capacitated plant location problem. *European Journal of Operational Research.* 87(2), pp. 203-213.