MSc thesis

Generating robust schedules for train maintenance staff

Author:
J.W. den Ouden

Supervisors:
dr. J.A. Hoogeveen
dr. ir. J.M. van den Akker
ir. B. Huisman

ICA-3927806

ABSTRACT

Train units are cleaned, inspected, and parked on shunting yards when not needed in service. One of the components of automatically generating plans for such a train service yard is to assign all tasks that need to be performed to personnel. These tasks are subject to release dates, deadlines, precedence constraints, and are situated at different locations at the shunting yard. We propose an approach using a greedy heuristic to obtain an initial solution, and a local search improvement step, optimizing a combination of the flexibility, fairness, and walking distances. The method is able to consistently find succesful results for realistic scenarios.

ACKNOWLEDGEMENTS

There are a few people without whom I could not have written this thesis. First of all, I would like to thank my supervisor Han Hoogeveen from the Department of Information and Computing Sciences at Utrecht University for his excellent guidance during the writing of this thesis. His feedback and criticism were immensely useful. Furthermore, I would also like to thank NS and especially my supervisor Bob Huisman at NS for providing the opportunity to write my thesis at his department of Maintenance Development, and for the useful discussions we had on the scheduling problems at the service sites. Furthermore, I would like to thank my colleagues of Maintenance Development. Besides Bob, Roel van den Broek deserves special thanks, for his great help, and for acting as a sort of extra supervisor. I would also like to thank my fellow interns at InterNS for enriching my internship with many discussions and for the fun and insightful trips to other parts of NS. Finally, I would like to thank my family and friends for their moral support during the writing of my thesis.

— Joris den Ouden, February 2018

CONTENTS

1 INTRODUCTION

In this study, we focus on the personnel scheduling problem for service sites belonging to the Nederlandse Spoorwegen (NS), which are the locations where trains are cleaned, inspected, and parked during the night. There are around 35 service sites around the country. The goal is to distribute the tasks that need to be performed in the planning period over the members of staff, resulting in a line of work for each member of staff. Not all members of staff are identical, since we consider cleaners, engineers, and technicians, and since we consider different shifts. The objective is to find schedules that are resistent to delays, and have a fair balance between the workloads of each member of staff, while minimizing traveling times between locations on the service site.

The personnel scheduling is only a small part in the automated generation of plans for the service sites. The complete problem consists of several parts, including the assignment of parking tracks, job-shop scheduling of the cleaning and inspection activities on tracks, and calculating the shunting routes. The personnel scheduling is the final step in the process, as we take a solution for the other problems, and use it as input for the personnel scheduling problem. For the other parts, algorithms have been developed in the past. The algorithm currently in use is made by Van den Broek [5], and uses local search to find a solution for the shunting routes and job shop scheduling problems. We use the output of that algorithm as input for our problem.

Section 1.1 contains some information about the NS and the various types of maintenance performed. Section 1.2 discusses the complete problem concerning the automated generation of plans. Section 1.3 contains a brief problem statement, and Section 1.4 contains some information on related problems and solutions from literature. In Section 1.5, we discuss some methods to measure the robustness of a schedule. Finally, Section 1.6 contains the outlint of the rest of the thesis.

1.1 Context

The Nederlandse Spoorwegen (NS) is the largest railway operator in the Netherlands, transporting over a million passengers each day. To transport the passengers, NS owns over 600 Electric Multiple Units (EMUs). Of course, these EMUs need regular cleaning, inspections, and maintenance. NS makes the distinction between three types of maintenance on the fleet. These types are

First-line service This type of service includes cleaning and performing inspections on the EMUs. Small technical problems will be fixed if detected and if time allows it. The first-line service happens every day, generally at night, at one of around 35 service sites, or "servicebedrijven".

Technical maintenance This type of maintenance consists of more thorough inspections and large repair jobs. Parts can be taken off the EMU to be replaced. Technical maintenance generally happens every few months. This happens at the four "onderhoudsbedrijven", which are located in Amsterdam, Leidschendam, Maastricht, and Onnen.

Refurbishment EMUs are refurbished once or twice during their lifetime, which generally comes down to around every fifteen years. This generally includes stripping the EMU back to the chassis, refurbishing or replacing components where needed, and putting it back together. This is generally paired with an overhaul of the interior. This process is performed to ensure that the EMUs still meet modern standards, and to extend the lifetime of the EMU. Refurbishments of EMUs happen in Haarlem, while components are refurbished in Tilburg.

The problem we cover in this thesis relates to the first-line service. First-line service happens outside of passenger peak hours, i.e. between the morning and evening rush, and in the night. It consists of several types of tasks, which may require specific types of member of staff, or require different resources, such as specialized types of tracks. The regular types of activities are

Safety check A Set of inspections performed every twelve days. An A-check takes 8 to 27 minutes depending on the type of the EMU. These checks can be performed at all regular tracks.

Safety check B Set of inspections, performed every two days, depending on the type of the EMU. A B-check takes about 38 to 90 minutes to complete, depending on the type of EMU. These checks can be performed at all regular tracks.

Internal cleaning Cleaning of the interior of the train by a cleaning team. This happens every day and takes between 24 and 46 minutes per carriage, depending on the type of the EMU. These need to be performed at a track with a cleaning platform.

External cleaning Cleaning of the exterior of the EMU using a washing installation, either with soap or with oxalic. This is done once a week. Every ninth time is with oxalic, the rest is with soap. Oxalic cleaning takes 4 minutes per carriage, and 10 minutes for each end. Cleaning with soap takes 1 minute per carriage, and 10 minutes for each end. External cleaning is done in the washing machine.

Furthermore, some activities are only done incidentally, if the need arises. We do not take these into account, as they are not in the scenarios we import. These types of activities are

Small repairs Reparations of small problems that would prevent the EMU from leaving the facility.

Removing graffiti Removal of graffiti if needed.

Service requests Performing repairs requested by Materieel Bijsturing.

ATB maintenance Performing maintenance on the ATB security system (Automatische TreinBeïnvloeding).

1.2 Automated plan generation of service sites

The NS tries to automate the generation of plans for service sites. This is done both to serve as a decision support tool for human planners, and to service as a tactical aid for strategic decision making. Since NS will be expanding its fleet in the coming years with over 250 new EMUs, the current facilities need to be evaluated to determine whether they are sufficient to accomodate the extra EMUs, or whether expansions need to be made. This evaluation is done by estimating the capacity of a service site, measured as the number of EMUs the service site is able to service in one day/night.

The process at a service site is as follows. EMUs enter the service site from the main railway network at exact times, either individually or combined. EMUs also leave the facility at exact times, either individually or combined. Because the combinations when entering and leaving are not necessarily the same, and because the combined EMUs may be too long for certain tracks, the trains need to be split and combined. Furthermore, each EMU needs internal cleaning, and some EMUs need external cleaning or safety checks. Finally, EMUs are also simply parked on the service site. Since tasks like cleaning require specific types of track, the EMUs are driven around the facility.

The capacity is estimated by solving six subproblems:

1 *Matching EMUs to combinations:* As EMUs are able to drive in combinations, EMUs may enter the location combined and need to leave the location in different combinations. For example, it may occur that the combination ICM III+IV+III enters, while the combinations ICM IV and ICM III+III need to leave. A matching between the incoming and outgoing combinations needs to be found, where each incoming EMU is matched to a component in an outgoing combination.

2 *Job-shop scheduling for non-human resources:* The tasks that need to be performed on each EMU are distributed over the non-human resources. These can include the washing installations, tracks that allow the bottom of the train to be inspected, isolated tracks for inspections on the ATB security system, etc.

3 *Assigning stabling tracks:* The service sites also serve as a parking location for (combined) EMUs when they are not in service. For this, there is a set of stabling tracks. Each EMU needs to be assigned to a stabling track, for when there is no task being performed on the EMU.

4 *Finding shunting routes:* EMUs need to be routed through the location, as the maintenance may need to happen at specific locations. EMUs also need to be routed from the entrance/exit to the maintenance locations and to their stabling track. This results in a set of shunting operations, where an EMU is moved from one track to another.

5 *Determining when to split and combine:* In order to change the combinations of the EMUs, the EMU combinations need to be split. Later, EMUs need to be connected in order to obtain the desired combinations. The problem in this step is to find out when and where to do the splitting and combining.

6 *Job-shop scheduling and routing human resources:* Finally, the human resources need to be assigned to the tasks. This includes the maintenance and cleaning tasks, as well as the shunting

of the EMUs. The human resources are planned at a later stage than the non-human resources since the assignment of human resources to tasks is often dependent on the location of the EMU. There may also be additional constraints on the workloads of the human resources, such as balance between workloads, mandated break times, walking distances, etc.

This thesis will focus on finding a solution for Subproblem 6, given a solution for the Subproblems 1 to 5. A solution for Subproblems 1 to 5 consists of an assignment of starting and ending times to each job that needs to be scheduled, as well as for any added jobs for splitting, moving, and combining the EMUs. Te solution also contains the start and end locations for each job, and the set of resources (such as tracks) required to execute the task.

1.2.1 Previous work

Several papers and theses have already been written about (parts of) this problem. Here, we will try to give a brief summary of the work so far.

NS have created a tool internally that is known as the OPG. This tool is based on the model by Kroon et al. [26]. The OPG is able to determine the matching for incoming and outgoing EMU compositions, determining the track assignment for parking, and determining the shunting movements. Note that the OPG does not take tasks into consideration in any way, i.e. it solves Subproblems 1, 3 and 4. The OPG works by solving a shortest path problem using Dijkstra's algorithm and by solving an MIP.

Van Dommelen [11] provided a method for solving Subproblem 2, i.e. scheduling the service tasks. To do this, a heuristic and a mixed integer programming approach were used. Here, the cleaning activities were modeled as a hold-while-wait flow shop problem. The matching and track assignments were done using the OPG.

Van den Broek [5] has created a model that integrates Subproblems 1 to 4 and solves the problem as a whole using simulated annealing. The performance of this heuristic has been compared to the OPG in both artificial and real scenarios. The heuristic is able to plan more EMUs than the OPG, even in testcases where no service tasks have to be performed.

Other authors who have worked on this problem are Van den Heuvel [19], who uses a decomposition approach to solve Subproblems 1 to 4, and Wolfhagen [38] who uses a mathematical programming approach to solve Subproblems 1, 3 and 4, with reallocation of trains during the planning period.

None of the models explicitly take into account Subproblems 5 and 6. For example, Van den Broek [5] assumes that all splitting of EMUs will happen immediately on entry and combining will happen just before exiting. It is also assumed that there is always a sufficient number of human resources at every location.

1.3 Problem statement

We designate Subproblem 6 as the NS Staff Routing Problem (NSSRP). Since the input of the NSSRP is a solution for Subproblems 1 to 5, the result of solving the NSSRP is a solution for Subproblems 1 to 6.

An instance of the NSSRP consists of a set of activities to be scheduled, with release dates, durations, deadlines, and start and finish locations. Furthermore, an activity may require one

member of staff of a specific type, such as a cleaning team or an engineer. Some tasks do not require any technician. The activities are ordered using precedence relations. The instance also contains a set of staff members, each with their own shift and skill.

The goal is to assign the activities to members of staff, resulting in a line of work for each member of staff. This line of work must also contain a break roughly in the middle of their shift. Note that these lines of work only add extra precedence constraints to the activities, they do not specify exact starting times. These starting times can be inferred using the constraints.

The objective is to find a schedule that is resilient to disruptions, contains a fair distribution of labour over the members of staff, and minimizes walking distances. The fairness is measured using the standard deviation of the lengths of the lines of work, where the length is the sum of the durations of the activities contained and the walking distances in minutes between the activities. The resilience is measured using a measure known as concurrent flexibility. For more details, see Section 1.5 and Chapter 3.

The complete definition of the NSSRP is given in much greater detail in Chapter 2. The release dates, deadlines, and precedence constraints are given based on a solution for Subproblems 1 to 5, which we obtain from Van den Broek [5]. For more information, see Chapter 6.

1.4 Related problems

In this section, we discuss some problems that are related to our problem. We will use the terms resource and staff member interchangeably.

Resource flows In the standard resource-constrainted project scheduling problems (RCPSP), individual units of resources are generally considered as equal. However, in some cases, it might pay off to individualize resource units. This entails finding a line of work for each unit of the resource, which results in additional precedence constraints. One technique used to individualize resource units at this stage is to use resource flow networks, defined by Artigues and Roubellat [3]. In a resource flow network, the activities are represented in a graph, and the resources are represented as flow through the graph. When resources are allocated, precedence constraints are added between tasks performed in succession by each unit. Since the result is a set of precedence constraints, no starting or finishing times are fixed. An example of the application of resource flow networks on the RCPSP is given by Leus and Herroelen [28].

Finding a resource flow is similar to solving the NSSRP in that it finds a line of work for each individual resource unit, adding precedence constraints for each line of work. Furthermore, the starting and finishing times of activities are not fixed. However, no release dates or deadlines are used. Furthermore, the tasks are not geographically distributed, so there are no travel times.

A heuristic for finding resource flows is provided by Policella et al. [33], who propose a method known as Chaining in order to improve robustness. Chaining is a greedy construction heuristic that assigns tasks to specific resource units in such a way that the number of synchronization points is minimized. For example, assigning tasks related by precedence constraints to the same technician improves the reliability, as domino effects are less likely.

Deblaere et al. [9] propose three heuristics based on integer programming and one constructive procedure. One of these works by minimizing the number of extra precedence constraints, one

works by maximizing the amount of time by which each activity may be delayed without delaying the start of each other activity. The third heuristic works similar to the second heuristic, but it is more selective in the selection of pairwise floats.

Technician and task scheduling problem (TTSP) Dutot et al. [12] give a description of a challenge problem, for which solutions were provided by several authors. An instance of the technician and task scheduling problem (TTSP) consists of a set of tasks that need to be performed during the planning period, and a set of technicians. The planning period is several days. Each technician is proficient in a number of skills, with varying levels of proficiency. Tasks vary in difficulty and may require more than one technician. In the problem, technicians are grouped into teams in order to perform the tasks. Teams must stay together on a given day, but can be broken up on other days. Technicians can be unavailable on specific days. Furthermore, each task also has a duration, outsourcing cost, a set of predecessor tasks, a set of successor tasks, and a priority. The objective is to minimize the makespan. Furthermore, a budget is available for outsourcing.

The TTSP is similar to the NSSRP, as both are variations of the RCPSP, with precedence constraints, multiple skills, and unavailable resources. The differences are that the TTSP does not consider release dates, deadlines, and geographic locations. The TTSP also fixates the starting and finishing times of tasks in a solution, whereas the NSSRP only returns an ordering on the tasks. Furthermore, the NSSRP does not consider overlapping skills, multiple proficiency levels, or outsourcing.

Firat and Hurkens [13] give an MIP-formulation for this problem, while Cordeau et al. [7] use an adaptive large neighbourhood search approach, using construction heuristics to obtain an initial solution.

Service technician routing and scheduling problem (STRSP) Kovacs et al. [25] build upon the approach by Cordeau et al. [7]. They define a new problem, known as the service technician routing and scheduling problem (STRSP). The STRSP is an extension of the TTSP, where besides the required skills, duration, priority, and precedence relations, each task now also has a release date, deadline, and geographical location.

The STRSP is very similar to the NSSRP, since both are variations of the RCPSP with precedence constraints, multiple skills, unavailable resources, time windows, and geographic locations. Like with the TTSP, the main difference is that the STRSP fixates the starting times and finishing times of each task in the solution, whereas the NSSRP should only return an ordering on the tasks. The STRSP also does not consider tasks where the starting and finishing locations are different, as can be the case in the NSSRP.

Kovacs et al. [25] use adaptive large neighbourhood search to find solutions, both for the problem with teams as for the problem without teams. Pillac et al. [30] also provide a solution for the STRSP, using a regret constructive heuristic, parallel adaptive local search, and a mathematical-based post-optimization step, based on solving a set-covering problem.

1.5 Robustness as objective in scheduling

We use the robustness as one of the terms in the objective function. Of course, the robustness of a solution can be defined in several ways. We use proactive scheduling, meaning that we try to find schedules that are resistant to disruptions. In other words, schedules that are still valid when disruptions occur. We do not consider stochastic runtimes.

One of the robustness measures proposed is the measure of fluidity by Cesta et al. [6], which is based on the temporal slack associated with each activity. The higher the fluidity, the less the risk of a domino effect if a task is delayed. Another measure is the measure of flexibility by Aloulou and Portmann [2], which counts the number of pairs of activities in the solution which are not ordered by precedence constraints, both implicitly or explicitly. The higher this value, the lower the degree of interaction. Policella et al. [34] also introduce the measure of disruptability, which takes into account the impact of disruptions on the schedule.

Deblaere et al. [9] use three different robustness measures. The first of these is based on minimizing the number of precedence constraints the solution adds to the problem instance. The other two are based on the pairwise floats, i.e. the amount of time between the finishing time of an activity, and the starting time of an activity that starts sometime after the first activity has been completed. One of the measures uses the minimal sum of pairwise floats on all paths from one activity to another. The measure itself is the sum of these minimal sums for each pair of activities for which a positive resource flow is possible.The measure itself is the minimal sum of all these pairwise floats on all paths between. High values indicate a more stable resource allocation. The second measure using pairwise floats is more selective in the selection of pairwise floats: a higher priority is given to activities occuring at the end of the schedule, when compared to activities that occur early in the schedule. This is done since the activities that occur later in the schedule have a higher probability of being delayed.

A more general way of measuring robustness is by encoding the problem instance and added precedence constraints as part of the solution as a simple temporal problem (STP), defined by Dechter et al. [10]. A simple temporal problem consists of a set of variables, representing events in time, and a set of temporal constraints between these variables. Scheduling problems are generally represented with two variables for each activity, one representing the starting time of the activity, and one representing the finishing time, with a constraint between them to indicate the duration, or minimal duration. Using the graph representation, it can be determined whether the problem instance with added precedence constraints is still feasible. It can also be determined what the earliest and latest execution times of each variable are, by using a shortest path approach.

Using this representation, the flexibility of an STP can be determined. One such way is to simply add the size of the allowed interval for each variable (i.e. the amount of time between its latest and earliest execution time). The measure by Hunsberger [21] extends this approach, by also taking into consideration the flexibility between pairs of activities, instead of just the activities on their own. However, Wilson et al. [37] have shown that both these measures fail to capture the correlations between the temporal variables in a satisfactory way. They introduce their own method, known as concurrent flexibility, which finds a set of intervals for each variable, such that any value can be chosen from these intervals without affecting the ability to choose values for other variables. The concurrent flexibility is the set of intervals such that the total size is maximized.

Out of these robustness measures, we use the concurrent flexibility. For more details, see

Chapter 3.

1.6 Structure of this thesis

The structure of the rest of this thesis is as follows. Chapter 2 contains the complete specificiation of the problem, including a MIP-formulation and NP-completeness proof. Chapter 3 contains the method we use to model our problem, in order to be able to measure the flexibility. Chapter 4 contains the details on the method used to solve the problem. Chapter 5 contains an alternative for the MIP-formulation, which uses column generation. Chapter 6 contains the method used to evaluate our method, which is done by a case study for NS. The results are shown in Chapter 7. Finally, Chapter 8 contains the conclusion and some remarks on how the model could be enhanced in the future.

2 PROBLEM DESCRIPTION

In this chapter, we will give a description of the problem at hand. Section 2.1 gives a description of the contents of a problem instance. Section 2.2 gives the contents of the solution, and Section 2.3 gives the contents of the objective function. Section 2.4 contains a method to check the feasibility of a solution. Section 2.5 contains a MIP formulation of the problem. We give an NP-Completeness proof in Section 2.6.

2.1 Problem instances

Our problem consists of assigning activities that are to be performed on trains to members of staff. These activities include cleaning, inspections, and shunting. Cleaning can only be done by cleaners, inspections can only be done by mechanics, and shunting can only be done by engineers. In doing this, we need to take the constraints placed on the ordering of these activities into account. These constraints originate from the solution found by the algorithm by Van den Broek [5], and consist of an ordering of the activities concerning each individual EMU, and an ordering on the activities concerning each non-human resource, such as tracks. Furthermore, since the EMUs enter and exit the location at predetermined times, we also need to take the release dates and deadlines of activities into account. Like the constraints, these members of staff also have a specific time interval in which they are available to work. Finally, since the service sites we will be looking at can be quite large, and the activities are located at different parts of the service site, we also need to take the walking distances of the members of staff into account.

The main goal now consists of finding a *line of work* for each member of staff, where a line of work consists of the activities that the member of staff will perform. These lines of work should be chosen such that each activity is performed, each member of staff has a line of work containing only activities of the correct type, and the scheduling problem instance implied by the lines of work should still be feasible with respect to the precedence constraints, release dates and deadlines, and walking distances.

In our case, the release dates, deadlines, precedence constraints, etc. originate from a solution for Subproblems 1 to 5. For more information on the way these values are obtained, see Chapter 6.

2.1.1 Definitions

Definition 2.1. Let $\mathcal{L} = \{ 0, 1, \ldots, l, \ldots, |\mathcal{L}| - 1 \}$ be a set of geographic *locations*. The traveling times between locations in $\mathcal{L}$ is given by the function $dist : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$. The distance from $i \in \mathcal{L}$

to $j \in \mathcal{L}$ is given by $dist(i, j)$. Location $0 \in \mathcal{L}$ is the location of the depot.

Definition 2.2. Let $\mathcal{S} = \{ 1, \ldots, s, \ldots, |\mathcal{S}| \}$ be a set of *skills* members of staff can possess. These skills are disjoint.

Definition 2.3. Let $\mathcal{A} = \{ 1, \ldots, i, \ldots, n \}$ be a set of *activities*. An activity $i \in \mathcal{A}$ has the following properties:

- Processing time $p_i \geq 0$: completing i takes at least p_i time.
- Release date $r_i \geq 0$: i can only be started at or after r_i.
- Deadline $d_i \geq 0$: i must be finished on d_i.
- Starting location and finishing location $loc_i^s \in \mathcal{L}$ and $loc_i^f \in \mathcal{L}$.
- Required skill $rq_i \in \mathcal{S}$. If no member of staff is required, we have $rq_i = 0$.*

Definition 2.4. A *schedule* $\sigma : \mathcal{A} \to \mathbb{R}^+$ consists of an assignment of starting and finishing times for each activity. The starting time of an activity $i \in \mathcal{A}$ in schedule σ is denoted as $s_i \geq 0$. The finishing time of activity $i \in \mathcal{A}$ in schedule σ is $c_i \geq 0$. Because of the processing times, we require that $c_i \geq s_i + p_i$.

Definition 2.5. σ^{orig} is the *original schedule* containing the original starting times s_i^{orig} and finishing times c_i^{orig} for each activity $i \in \mathcal{A}$, as given by the solution for subproblems 1–5.

Definition 2.6. Let $i \prec j$ be a *precedence constraint*, with $i, j \in \mathcal{A}$, indicating that j may not be started until i is finished. In other words, in any feasible schedule $c_i \leq s_j$ must hold. The set of precedence constraints is $\mathcal{P}$.

Assumption 2.1. The precedence constraints in $\mathcal{P}$ do not form cycles. In other words, the constraint graph associated with $\mathcal{P}$ is a directed acyclic graph, with a topological ordering.

Assumption 2.2. The set of precedence constraints $\mathcal{P}$ does not contain any redundant constraints:

$$i \prec j \in \mathcal{P} \wedge j \prec k \in \mathcal{P} \implies i \prec k \notin \mathcal{P}$$

Definition 2.7. The set of *predecessors* and the set of *successors* of activity $i \in \mathcal{A}$ are the sets of activities that have a direct precedence constraint with activity i as the later activity or as the earlier activity respectively, defined as:

$$pre_i = \{ j \in \mathcal{A} \mid j \prec i \in \mathcal{P} \}$$
$$suc_i = \{ j \in \mathcal{A} \mid i \prec j \in \mathcal{P} \}$$

* We still consider these activities since they do use other resources, such as tracks. To ensure that we can shift the activities freely, without risk of using more resources than allowed, we have added precedence constraints for these activities. If we would not consider them in the planning, we can no longer make these statements.

The set of all predecessors and the set of all successors of activity $i \in \mathcal{A}$ are the sets of activities that must be finished before i can be started and the set of activities that cannot be started until i is finished respectively. These are defined recursively as:

$$apre_i = pre_i \cup \bigcup_{j \in pre_i} pre_j$$

$$asuc_i = suc_i \cup \bigcup_{j \in suc_i} suc_j$$

Definition 2.8. The set of *unrelated activities* to $i \in \mathcal{A}$ is defined as follows:

$$unrl_i = \mathcal{A} \setminus (apre_i \cup asuc_i \cup \{ i \})$$

Definition 2.9. Let $\mathcal{R} = \{ 1, \ldots, k, \ldots, |\mathcal{R}| \}$ be the set of *members of staff*. A member of staff $k \in \mathcal{R}$ has the following properties:

- Start and end of the shift $0 \leq rs_k < rf_k$.
- A skill $rsk_k \in \mathcal{S}$ indicating which activities the member of staff can execute.

Each member of staff has three dummy activities:

- $st_k \notin \mathcal{A}$, representing the start of the shift, with $r_{st_k} = d_{st_k} = rs_k$, and $p_{st_k} = 0$.
- $fi_k \notin \mathcal{A}$, representing the end of the shift, with $r_{fi_k} = d_{fi_k} = rf_k$, and $p_{fi_k} = 0$.
- $br_k \notin \mathcal{A}$, representing the break. The release date and deadline can be used to control when the break may take place. These are given in the problem instance.

All of these tasks require the skill rsk_k and have location $0 \in \mathcal{L}$ as start and end location. None of these activities have any precedence constraints in $\mathcal{P}$. The activities st_k and fi_k are always the first and last activities to be scheduled for $k \in \mathcal{R}$, see Definition 2.13.

Assumption 2.3. We assume that the number of staff members is significantly less than the number of activities: $|\mathcal{R}| \ll |\mathcal{A}|$.

Definition 2.10. The subset of members of staff that possess skill $s \in \mathcal{S}$ is $\mathcal{R}_s = \{ k \in \mathcal{R} \mid rsk_k = s \}$. These sets are disjoint. The subset of activities requiring a skill $s \in \mathcal{S}$ is the set $\mathcal{A}_s = \left\{ i \in \mathcal{A} \,\middle|\, rq_i = s \right\}$. These sets are also disjoint. The set of activities requiring any skill $s \in \mathcal{S}$ is the set $\mathcal{A}_\mathcal{S} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$.

Definition 2.11. We use the following notation to indicate the supersets of $\mathcal{A}$ with the dummy activities added:

$$\mathcal{A}^{st} = \mathcal{A} \cup \{ st_k \mid k \in \mathcal{R} \}$$
$$\mathcal{A}^{fi} = \mathcal{A} \cup \left\{ fi_k \,\middle|\, k \in \mathcal{R} \right\}$$

$$\begin{aligned}\mathcal{A}^{br} &= \mathcal{A} \cup \{\, br_k \mid k \in \mathcal{R} \,\}\\ \mathcal{A}^{st,br} &= \mathcal{A}^{st} \cup \mathcal{A}^{br}\\ \mathcal{A}^{br,fi} &= \mathcal{A}^{br} \cup \mathcal{A}^{fi}\\ \mathcal{A}^{*} &= \mathcal{A}^{st} \cup \mathcal{A}^{fi} \cup \mathcal{A}^{br}\end{aligned}$$

Definition 2.12. A problem instance for the NS Staff Routing Problem (NSSRP) consists of the tuple $\langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$.

2.1.2 Handling other resources

Besides the members of staff, an activity may require other resources, such as (specific types of) tracks, washing installations, machines, etc. In order to keep the problem contained to the staff scheduling phase, we assume the following:

Assumption 2.4. The earliest starttime schedule for an instance of this problem is always resource feasible, w.r.t. to the non-human resources: when each activity is started at the earliest point in time at which it is allowed, i.e. all predecessors have been completed, then the demand does not exceed the capacity.

This assumption holds if the non-human resource requirements have been encoded into the precedence constraints. Making this assumption allows us to disregard all non-human resource requirements, simplifying the problem. We can make this assumption because our problem instances are taken from the solutions produced by the algorithm by Van den Broek [5]. The resources are assigned in Subproblem 2.

2.2 Goal

The goal is to find a schedule that is feasible from both the time perspective as from the resource perspective. We say that a schedule is *time feasible* if there exists an assignment of starting times such that all constraints, except resource constraints are met. A schedule is *resource feasible* if there exists an assignment of resources to all tasks such that there are no conflicts. We use a technique known as *constraint posting* to achieve this, see Policella et al. [32]. With a regular schedule, resource conflicts are avoided by assigning fixed starting times to each activity. When using constraint posting, the conflicts are avoided by adding extra precedence constraints, ensuring that every schedule that is time feasible is automatically resource feasible.

As such, our goal is to find a line of work of activities for each member of staff, such that all activities have sufficient members of staff, and that all constraints are satisfied. These lines of work imply a set of extra precedence constraints on the activities. Using this representation allows us to make statements about the flexibility of the found solutions. We define a line of work as follows:

Definition 2.13. A *line of work* for member of staff $k \in \mathcal{R}$ consists of a sequence of activities that are performed by member of staff k in the order in which they are given in the line of work. Lines of work start with activity st_k, followed by br_k and zero or more activities from $\mathcal{A}$ in any order, followed by activity fi_k. This gives the line of work for member of staff k the following layout:

$$\omega_k = [st_k = i_0, i_1, \ldots, br_k, \ldots, i_m, i_{m+1} = fi_k]$$

We denote the i'th activity of ω_k as $(\omega_k)_i$. We denote that an activity $i \in \mathcal{A}^*$ is in a line of work ω_k as $i \in \omega_k$.

In a solution to the problem, an activity can only be on one line of work, as each activity requires at most one member of staff. This allows us to give the following definitions:

Definition 2.14. The *line of work predecessor* $rpre_i \in \mathcal{A}^{st,br}$ of activity $i \in \mathcal{A}^{br,fi}$ is the activity that immediately preceedes i in the line of work ω_k that contains i. If $i = st_k$, $rpre_i = 0$. The *line of work successor* $rsuc_i \in \mathcal{A}^{br,fi}$ of activity $i \in \mathcal{A}^{st,br}$ is the activity that immediately succeeds i in the line of work ω_k that contains i. If $i = fi_k$, $rsuc_i = 0$.

Definition 2.15. Let ω be a line of work. The total amount of time used for walking in the line of work is

$$wd_\omega = \sum_{\substack{i \in \omega \\ i \notin \mathcal{A}^{st} \setminus \mathcal{A}}} dist(loc^f_{rpre_i}, loc^s_i)$$

The total workload, including the walking distances is

$$wl_\omega = \sum_{i \in \omega} p_i + wd_\omega$$

Definition 2.16. The precedence constraints in $\mathcal{P}$ and the lines of work result in a *constraint graph* $G = (V, A)$, which is defined as follows:

$$V = \mathcal{A}^*$$
$$A = \{ (i,j) \mid i \prec j \in \mathcal{P} \} \cup \left\{ (rpre_i, i) \;\middle|\; i \in \mathcal{A}^{br,fi} \right\}$$

Example. Given the example line of work $\omega_k = [st_k, 1, 2, br_k, 3, 4, fi_k]$, we get the following precedence constraints, in addition to the constraints in $\mathcal{P}$.

$$s_1 \geq c_{st_k} + dist(loc^f_{st_k}, loc^s_1) \qquad s_3 \geq c_{br_k} + dist(loc^f_{br_k}, loc^s_3)$$
$$s_2 \geq c_1 + dist(loc^f_1, loc^s_2) \qquad s_4 \geq c_3 + dist(loc^f_3, loc^s_4)$$
$$s_{br_k} \geq c_2 + dist(loc^f_2, loc^s_{br_k}) \qquad s_{fi_k} \geq c_4 + dist(loc^f_4, loc^s_{fi_k})$$

Remark. In the worst case, we have $|A| = O(|V|^2)$. In some cases however, we have $|A| = O(|V|)$. In that case, the constraint graph is *sparse*. In the runtime analyses, we give the runtimes for both cases.

Remark. The constraints in $\mathcal{P}$, and the constraints implied by the selected lines of work, together with the release dates, durations, and deadlines form a scheduling problem. This problem is characterized using the following variables:

$$s_i = \text{the start time of activity } i \in \mathcal{A}^*$$
$$c_i = \text{the finishing time of activity } i \in \mathcal{A}^*$$

The feasibility problem, can them be formulated using the constraints

$$c_j \leq s_i \qquad \forall i \in \mathcal{A}, j \in pre_i \tag{2.1}$$
$$s_i - c_j \geq dist(loc_j^f, loc_i^s) \qquad \forall i \in \mathcal{A}^{br,fi}, j = rpre_i \tag{2.2}$$
$$c_i \geq s_i + p_i \qquad \forall i \in \mathcal{A}^* \tag{2.3}$$
$$s_i \geq r_i \qquad \forall i \in \mathcal{A}^* \tag{2.4}$$
$$c_i \leq d_i \qquad \forall i \in \mathcal{A}^* \tag{2.5}$$
$$s_i \geq 0 \qquad \forall i \in \mathcal{A}^* \tag{2.6}$$

Constraints 2.1 ensure that the precedence constraints in $\mathcal{P}$ are satisfied, and Constraints 2.2 ensure that the precedence constraints implied by the lines of work are satisfied, and that the travel times are respected. Note that this constraint uses $rpre_i$, as given in Definition 2.14. Constraints 2.3 ensure that the tasks take at least their processing times to finish. Constraints 2.4 and 2.5 ensure that each activity is started after the release date of its project, and finished before the deadline of its project respectively. Constraints 2.6 control the domain of the decision variables.

Note that ensuring that all activities are executed within the shift of the assigned member of staff is done through the dummy activities st_k and fi_k and through Constraints 2.2. Therefore, we do not need any constraints using the values rs_k or rf_k.

Definition 2.17. A solution Ω for an instance of the NSSRP consists of a line of work ω_k for each member of staff $k \in \mathcal{R}$:

$$\Omega = \{ \omega_k \mid k \in \mathcal{R} \}$$

A solution Ω is feasible if

1. Every activity $i \in \mathcal{A}$ with $rq_i \neq 0$ is placed in exactly one line of work $\omega_k \in \Omega$, where $rsk_k = rq_i$. Activities where $rq_i = 0$ must not be placed in any line of work.

2. Each line of work has the structure as given in Definition 2.13.

3. The scheduling problem specified through Constraints 2.1 to 2.6, given Ω, has a feasible solution.

2.3 Objective function

The quality of a solution to a problem instance can be characterized by the following features:

- The plan should be *flexible*, meaning that it should still be usable if there are small changes in the problem instance, such as delays in executing a task. This will be covered in more detail in Chapter 3.

- *Fairness* between lines of work: it is preferable that the workload is balanced over the workforce, where wl_k is the total workload (including walking) of $k \in \mathcal{R}$.

- We want to minimize the *walking distance*, as it is prefereable that the members of staff do not have too much overhead.

2.4 Assigning completion times given lines of work

A key part of the NSSRP is that we do not fixate the starting and finishing times of activities. Instead, we only return a set of lines of work, which add extra precedence constraints. This means that checking whether a solution is feasible consists of solving the problem specified through Constraints 2.1 to 2.6. Given that this model is a constraint satisfaction problem on the starting times, checking the existence of a feasible schedule for a problem instance can also be done by modeling the instance as a simple temporal network. This is covered in detail in Chapter 3. However, because the durations are only lower bounds on the durations, and not exact durations, and since the precedence constraints only specify a lower bound and not an upper bound, we can also use a simpler method, which simply assigns the earliest possible starting times.

This method consists of two steps. The first step is to check whether the constraints implied by the lines of work combined with the constraints given in the problem instance do not form any cycles in the constraint graph. If a cycle is formed, assuming the durations of tasks are larger than 0, no solution is possible. If no cycle is found, the second step is to find an actual solution. A solution can be found by choosing the earliest starting times possible for each activity, based on the precedence constraints, its release date, and the constraints implied by the lines of work. If there is any activity that cannot be started in time for it to be finished before the deadline, there is no possible solution. The outline of this algorithm is shown in Algorithm 2.1. The activities are processed in a topological ordering to ensure that all necessary completion times are known when determining the starting time of the activity.

We can combine detecting a cycle in a graph $G = (V, A)$ and finding a topological ordering by using the algorithm by Kahn [23], which runs in $O(|V| + |A|)$ time. The main loop at Line 6 in the algorithm runs $|V|$ times. At Line 7, a total of $O(|A| + |V|)$ values are compared, as each precedence constraint is only looked at once, and each activity is checked again. At line Line 9, another $O(|V|)$ values are compared. This makes the total runtime of this algorithm $O(|A| + |V|) = O(|A|)$. If the constraint graph is dense, this is a runtime of $O(|V|^2) = O(|\mathcal{A}^*|^2)$. If the constraint graph is sparse, we get a runtime of $O(|V|) = O(|\mathcal{A}^*|)$.

ALGORITHM 2.1: Earliest starting times

Input: Constraint graph $G = (V, A)$ based on $\langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$ and Ω, see Definition 2.16
Output: A solution for the problem given by Constraints 2.1 to 2.6 if one exists

$\tau \leftarrow$ GETTOPOLOGICALSORT(G) ▷ *Using the algorithm by Kahn [23]*
if $\tau =$ **nil:** ▷ *G contains a cycle*
  return no solution possible
for $i \in \mathcal{A}^*$**:**
  $s_i \leftarrow \infty, c_i \leftarrow \infty$
for $i \in \tau$**:**
  $s_i \leftarrow \max \left\{ c_j \mid j \in pre_i \right\} \cup \left\{ c_{rpre_i} + dist(loc^f_{rpre_i}, loc^s_i) \right\} \cup \{ r_i \}$
  $c_i \leftarrow s_i + p_i$
  if $s_i + p_i > d_i$**:**
    return no solution possible ▷ *i cannot be started in time for it to be finished before its deadline*
return solution

2.5 MIP FORMULATION

In this section, we give an MIP formulation of the problem. This formulation will be used to check whether instances are easily solvable using a solver, or if solving the instances in an exact way is infeasible. In the MIP formulation, we use the following variables:

$$x^k_{ij} = \begin{cases} 1 & \text{if activity } i \in \mathcal{A}^* \text{ is the predecessor of activity } j \in \mathcal{A}^* \text{ in the line of work for} \\ & \text{member of staff } k \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases}$$

$$y^k_i = \begin{cases} 1 & \text{if activity } i \in \mathcal{A}^* \text{ is executed by member of staff } k \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases}$$

$c_i =$ completion time of activity $i \in \mathcal{A}^*$

We also use some parameters. R_i is the subset of $\mathcal{R}$ containing the members of staff that are allowed to perform activity $i \in \mathcal{A}^*$.

$$R_i = \begin{cases} \mathcal{R}_{rq_i} & \text{if } i \in \mathcal{A} \\ \{ k \} & \text{if } i \in \left\{ st_k, br_k, fi_k \right\} \\ \varnothing & \text{otherwise} \end{cases}$$

Since we are only interested in finding a solution, we do not use the complete objective. To steer the MIP solver in the right direction, we use the walking distances as objective. The problem can then be formulated as follows:

$$\min \sum_{k \in \mathcal{R}} \sum_{i \in \mathcal{A}^*} \sum_{j \in \mathcal{A}^*} dist(loc^f_i, loc^s_j) x^k_{ij} \tag{2.7}$$

subject to

$$\sum_{k \in \mathcal{R}} y_i^k = 1 \qquad \forall i \in \left\{ i \in \mathcal{A}^* \;\middle|\; rq_i \neq 0 \right\} \tag{2.8}$$

$$\sum_{k \in \mathcal{R}} y_i^k = 0 \qquad \forall i \in \left\{ i \in \mathcal{A}^* \;\middle|\; rq_i = 0 \right\} \tag{2.9}$$

$$\sum_{k \in \mathcal{R} \setminus R_i} y_i^k = 0 \qquad \forall i \in \mathcal{A}^* \tag{2.10}$$

$$\sum_{i \in \mathcal{A}^{br,fi}} x_{st_k i}^k = 1 \qquad \forall k \in \mathcal{R} \tag{2.11}$$

$$\sum_{i \in \mathcal{A}^{st,br}} x_{ifi_k}^k = 1 \qquad \forall k \in \mathcal{R} \tag{2.12}$$

$$\sum_{j \in \mathcal{A}^{st,fi}} x_{ji}^k = y_i^k \qquad \forall i \in \mathcal{A}^{br,fi}, k \in \mathcal{R} \tag{2.13}$$

$$\sum_{j \in \mathcal{A}^{st,fi}} x_{jst_k}^k = 0 \qquad \forall k \in \mathcal{R} \tag{2.14}$$

$$\sum_{j \in \mathcal{A}^{br,fi}} x_{ij}^k = y_i^k \qquad \forall i \in \mathcal{A}^{st,br}, k \in \mathcal{R} \tag{2.15}$$

$$\sum_{j \in \mathcal{A}^{br,fi}} x_{fi_k j}^k = 0 \qquad \forall k \in \mathcal{R} \tag{2.16}$$

$$c_i \geq r_i + p_i \qquad \forall i \in \mathcal{A}^* \tag{2.17}$$

$$c_i \leq d_i \qquad \forall i \in \mathcal{A}^* \tag{2.18}$$

$$c_j - c_i \geq p_j \qquad \forall i \prec j \in \mathcal{P} \tag{2.19}$$

$$(c_j - c_i) + M(1 - x_{ij}^k) \geq p_j + dist(loc_i^f, loc_j^s) \qquad \forall i, j \in \mathcal{A}^*, k \in \mathcal{R} \tag{2.20}$$

$$x_{ij}^k \in \{ 0, 1 \} \qquad \forall i, j \in \mathcal{A}^*, k \in \mathcal{R} \tag{2.21}$$

$$y_i^k \in \{ 0, 1 \} \qquad \forall i \in \mathcal{A}^*, k \in \mathcal{R} \tag{2.22}$$

$$c_i \geq 0 \qquad \forall \in \mathcal{A}^* \tag{2.23}$$

Constraints 2.8 ensure that all activities requiring a member of staff are assigned to exactly one member of staff. Constraints 2.9 ensure that activities that do not require staff are not scheduled. Constraints 2.10 ensures that activities are not assigned to members of staff that cannot perform the activity. Constraints 2.10 ensure that each activity is only assigned to members of staff allowed to execute that activity. Constraints 2.11 and 2.12 ensure that each line of work starts with the dummy start activity and ends with the dummy finish activity. Constraints 2.13 and 2.15 ensure that each activity that is assigned to a member of staff is entered and exited exactly once in the corresponding line of work. Constraints 2.14 and 2.16 handle the edge cases with the beginning and end of the line of work. Constraints 2.17 ensure that activities cannot be started prior to the release date. Constraints 2.18 ensure that activities are finished before the deadline. Constraints 2.19 ensure that activities related by a precedence constraint are executed in the correct order. Constraints 2.20

enforces the walking distances. Constraints 2.21 to 2.23 specify the domain of the variables.

2.6 Complexity

In this section, we aim to show that the decision variant of the NSSRP, using only the total walking distance as the objective, is in the complexity class of NP-Complete problems. This variant is defined as follows:

Definition 2.18. The dNSSRP problem is the decision variant of the NSSRP. Given an instance $P = \langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$ and a target cost C, does there exist a solution Ω for P such that the total walking distance is no more than C?

To proof that the dNSSRP is in the complexity class NP-Complete, we first show that it is in the complexity class NP.

Lemma 2.1. *The dNSSRP is in the complexity class NP.*

Proof. A solution $\Omega = \{ \omega_k \mid k \in \mathcal{R} \}$ for the dNSSRP can be verified as follows:

- Constraint 1 can be verified by checking whether each activity is contained in exactly one line of work of the corresponding skill. This can be done in $O(|\mathcal{A}^*|)$ time by going through the lines of work, marking the activities within. The constraint can then be verified by checking if each activity that needs to be scheduled has been marked by a line of work of the correct skill.
- Constraint 2 can be verified in $O(|\mathcal{A}^*|)$ time, by checking whether all lines of work conform to the definition.
- Constraint 3 can be verified by checking whether the selected lines of work still allow for a feasibible schedule, which requires solving the feasibility problem Constraints 2.1 to 2.6. This can be done in $O(|\mathcal{A}^*| + |\mathcal{P}|)$ time by assigning the earliest possible starting times, see Algorithm 2.1.
- Checking the total walking distance can be done in $O(|\mathcal{A}^*|)$ by enumerating the lines of work and adding up the walking distances.

Furthermore, a solution can be represented using the following decision variables:

$$x_{ij}^k = \begin{cases} 1 & \text{if the line of work } \omega_k \text{ for } k \in \mathcal{R} \text{ goes from } i \in \mathcal{A}^{st,br} \text{ to } j \in \mathcal{A}^{br,fi} \\ 0 & \text{otherwise} \end{cases}$$

Since a solution for an instance of the dNSSRP can be verified in polynomial time, and since a solution can be represented using a polynomial number of decision variables, the dNSSRP is in NP. □

To prove that dNSSRP is NP-Hard, we show that the Hamiltonian path problem, which is known to be NP-Hard, see Garey and Johnson [17], is polynomial-time reducible to dNSSRP. The Hamiltonian path problem is defined as follows:

DEFINITION 2.19. Given an undirected, unweighted graph $G = (V, E)$, does there exist a path π such that every vertex is visited exactly once?

LEMMA 2.2. *An instance of the Hamiltonian path problem can be converted in polynomial time to an instance of dNSSRP.*

Proof. We can convert a problem instance $P' = G = (V, E)$ for the Hamiltonian problem to a problem instance $P = \langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$ for the dNSSRP as follows:

- $\mathcal{S} = \{ s \}$.
- $\mathcal{R} = \{ k \}$, with $rsk_k = s$, $rs_k = 0$ and $rf_k = \infty$.
- $\mathcal{P} = \emptyset$.
- For each vertex $i \in V$, add a location l_i to $\mathcal{L}$.
- For each vertex $i \in V$, add an activity i to $\mathcal{A}$ with $p_i = 0$, $r_i = 0$, $d_i = \infty$, $rq_i = s$, and $loc_i^s = loc_i^f = l_i$.
- For each pair of vertices $i, j \in V$ with $(i, j) \in E$, set the distance $dist(l_i, l_j) = 0$. For all pairs of vertices $i, j \in V$ with $(i, j) \notin E$, set $dist(l_i, l_j) = 1$.
- Add a starting location 0 to $\mathcal{L}$, with $dist(0, l) = dist(l, 0) = 0 \quad \forall l \in \mathcal{L}$.
- Set C to 0.

This conversion can be done in $O(|V|^2)$ time, which is polynomial. □

LEMMA 2.3. *Let $P' = G = (V, E)$ be an instance for the Hamiltonian path problem, and let problem instance $P = \langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$ the corresponding reduced instance for dNSSRP. The following holds:*

$$P' \text{ is a YES-instance for the Hamiltonian path problem} \iff P \text{ is a YES-instance for dNSSRP}$$

Proof. This proof consists of two parts:

$\Rightarrow$ A Hamiltonian path π can be converted to a solution $\Omega = \{ \omega_k \}$. The line of work ω_k consists of all activities in $\mathcal{A}$, in the same ordering as the corresponding vertices in the path π. The dummy activities st_k and fi_k are placed at the start and at the end to model the depot. br_k can be placed at an arbitrary location in the line of work. Since there is only one member of staff and all tasks are in ω_k, Constraint 1 hold. Because of the dummy activities Constraint 2 also hold. The feasibility problem can then be solved by setting s_i and c_i for all $i \in \mathcal{A}$ as follows:

$$s_i = c_{rpre_i} + dist(loc^f_{rpre_i}, loc^s_i)$$
$$c_i = s_i + p_i$$

Because there are no precedence constraints, and the release dates and deadlines are all 0 and ∞ respectively, the constraints Constraints 2.1, 2.4 and 2.5 hold automatically. Because of the way s_i and c_i are set, Constraints 2.2 and 2.3 hold respectively. Since all distances to and from 0 are 0, the implied constraints by the dummy activities st_k, fi_k, and br_k also hold.

Since ω_k follows the same ordering as the path π, which of course consists of edges in E, and since we defined $dist(l_i, l_j)$ to have a distance of 0 if $(l_i, l_j) \in E$, we have that each walking distance in ω_k is 0. Furthermore, all walking distances to and from 0 are also defined as 0. Since ω_k is the only line of work, and all walking distances in ω_k are 0, the total walking distance in Ω is 0.

$\Leftarrow$ The solution Ω containing only the line of work ω_k with total walking distance 0 can be converted into a sequence of vertices π by removing the dummy activities and adding the vertices corresponding to the activities in ω_k to π in the same ordering.

Since the total walking distance is 0, and since the walking distances can only be positive, we know that for all $i \in \omega_k$, the walking distance $dist(loc^f_{rpre_i}, loc^s_i)$ is 0. This means that there is an edge containing the corresponding vertices in E. This means that π is a path, and since ω_k contained all activities in $\mathcal{A}$, π contains all vertices in V, meaning that π is a Hamiltonian path. □

Lemma 2.4. *The dNSSRP is in the complexity class of NP-Hard problems.*

Proof. See Lemma 2.2 and Lemma 2.3. □

Theorem 2.1. *The dNSSRP is in the complexity class of NP-Complete problems.*

Proof. As the dNSSRP is both in NP (see Lemma 2.1) and in NP-Hard (see Lemma 2.4), it is in NP-Complete. □

3 MODELING THE PROBLEM AS AN STN

In this chapter, we will give our method for modeling a problem instance and a solution as a simple temporal problem. This allows us to make statements about the flexibility and see how much we can change the starting times of activities. We give some definitions on simple temporal problems in Section 3.1. Section 3.2 contains the way we model the problem instance. In Section 3.3, we will give information on the meaning of flexibility and on methods to calculate the flexibility. Section 3.4 contains some ways to change the model or implementation in order to improve performance.

3.1 Simple Temporal Problem

Dechter et al. [10] define the simple temporal problem (STP) as follows:

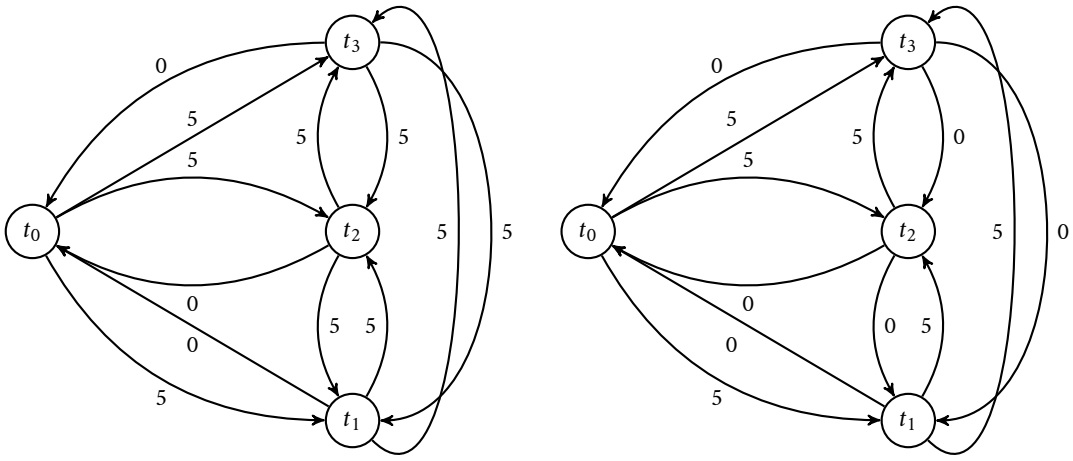
Definition 3.1. Let $S = (T, C)$ be a simple temporal network (STN). Here, T is a set of temporal variables, i.e. each variable $t_i \in T$ represents a specific time point. It also contains a special variable t_0, which is always assigned timepoint 0, providing a fixed start. C is a set of constraints on these timepoints. These constraints are of the form $t_j - t_i \leq l_{ij}$, where $t_i, t_j \in T$.

We will use some other notations for constraints, which can be rewritten as follows:

- $t_i \leq t_j$, which can be rewritten as $t_i - t_j \leq 0$.
- $t_i - t_j \geq l_{ij}$, which can be rewritten as $t_j - t_i \leq -l_{ij}$.
- $t_j - t_i = l_{ij}$, which can be rewritten as $t_j - t_i \leq l_{ij}$ and $t_i - t_j \leq -l_{ij}$.

Note that a variable can be any type of event. In the context of scheduling, we use a variable to represent the starting time of an activity and another variable to represent the completion time of an activity.

Definition 3.2. A solution $\sigma : T \to \mathbb{R}$ for an STN $S = (T, C)$ consists of an assignment of values to all variables $t_i \in T$, such that all constraints in C are satisfied. If such a solution exists, the STN is *consistent*. If no such solution exists, the STN is inconsistent. The value of variable $t_i \in T$ in σ is denoted as $\sigma(t_i)$.



(A) S_1 (B) S_2

FIGURE 3.1: Example STNs S_1 and S_2, specifying a concurrent and sequential execution of the events respectively.

DEFINITION 3.3. An STN $S = (T, C)$ can be modelled as a directed arc-weighted graph $G = (V, A, w)$, known as the distance graph, with

$$\begin{aligned} V &= T \\ A &= \left\{ (t_i, t_j) \;\middle|\; t_j - t_i \leq l_{ij} \in C \right\} \\ w((t_i, t_j)) = l_{ij} &\iff t_j - t_i \leq l_{ij} \in C \end{aligned}$$

EXAMPLE. Let S_1 and S_2 be two STNs, containing three events, t_1, t_2, and t_3. S_1, shown in Figure 3.1a, allows for a concurrent execution of the events, meaning that all three events can be performed at the same time. S_1 contains the following constraints:

$$\begin{aligned} 0 \leq t_1 - t_0 \leq 5 \qquad & -5 \leq t_2 - t_1 \leq 5 \\ 0 \leq t_2 - t_0 \leq 5 \qquad & -5 \leq t_3 - t_1 \leq 5 \\ 0 \leq t_3 - t_0 \leq 5 \qquad & -5 \leq t_3 - t_2 \leq 5 \end{aligned}$$

S_2, shown in Figure 3.1b, requires that the events are executed sequentially in order. Specifically, it contains the precedence constraints $t_1 \prec t_2$ and $t_2 \prec t_3$. These are encoded in the STN as the constraints:

$$\begin{aligned} 0 \leq t_1 - t_0 \leq 5 \qquad & 0 \leq t_2 - t_1 \leq 5 \\ 0 \leq t_2 - t_0 \leq 5 \qquad & 0 \leq t_3 - t_1 \leq 5 \\ 0 \leq t_3 - t_0 \leq 5 \qquad & 0 \leq t_3 - t_2 \leq 5 \end{aligned}$$

In both S_1 and S_2, the interval of possible time values for each individual variable is $[0, 5]$. We can independently select any value in the interval $[0, 5]$ for each of the three events in S_1, where in S_2 we have that $\sigma(t_1) \leq \sigma(t_2) \leq \sigma(t_3)$, in any feasible solution σ.

The consistency of an STN can be checked using the following property:

PROPERTY 3.1. *A given STN S is consistent if and only if the corresponding distance graph G contains no negative weight cycles.*

A negative weight cycle can be detected by applying the algorithm by Bellman [4] and Ford and Fulkerson [15], which has a worst-case runtime of $O(|V|\,|A|)$. Faster algorithms have been developed as well, which operate on the STN instead of on the distance graph. An example is the fully dynamic algorithm by Ramalingam et al. [35], which handles the addition or deletion of a single constraint to the STN in $O(|C| + |T| \log |T|)$ time. For more information on algorithms for checking the consistency of an STN, we refer to the work by Planken [31].

Note that if there are only contains constraints of the form $t_i - t_j \geq l_{ij}$ for $t_j \neq t_0$, and only constraints of the form $t_i - t_0 \leq l_{i0}$, the consistency can be checked by computing the earliest starting times for each variable. If the earliest starting time for any variable is later than its deadline, then the STN is inconsistent. Furthermore, if there are any cycles in the constraints (disregarding the constraints involving t_0), then the STN is also inconsistent. Using this method, the consistency can be checked in $O(|T| + |C|)$ time. A version of this method is shown in Section 2.4.

In the graph representation, a path from t_i to t_j in the graph, consisting of $i_0 = t_i, i_1, \ldots, i_k = t_j$, induces the constraint

$$t_j - t_i \leq \sum_{j=1}^{k} l_{i_{j-1} i_j}$$

This gives rise to another representation for the STN, known as the d-graph. This is a complete directed graph, where each arc (t_i, t_j) is weighted by the length of the shortest path from t_i to t_j in G. This graph can be constructed by using an all-pairs shortest paths algorithm, returning a distance matrix D_S, with $D_S[t_i, t_j]$ the length of the shortest path between t_i and t_j. Using the d-graph, we can obtain the strongest constraints implied by C for any pair of variables.

PROPERTY 3.2. *In a given STN S, with the corresponding distance graph G, the strongest constraints implied by C with respect to the temporal difference between variables $t_i, t_j \in T$ are*

$$t_i - t_j \leq D_S[t_j, t_i]$$
$$t_j - t_i \leq D_S[t_i, t_j]$$

where $D_S[t_i, t_j]$ is the length of the shortest path between t_i and t_j in G.

Using the strongest constraints property, the d-graph representation provides the same domains for the variables as the original set of constraints does. This gives us the following property:

PROPERTY 3.3. *In a given STN S, the set of feasible values for each variable $t_i \in T$ is given by the interval $[et(t_i), lt(t_i)]$, where*

$$et(t_i) = -D_S[t_i, 0]$$
$$lt(t_i) = D_S[0, t_i]$$

3.2 Modeling a problem instance and solution

3.2.1 Modeling the problem instance

In this section, we will describe our method for generating an STN for an instance of the scheduling problem. The STN consists of the following temporal variables:

- Each temporal variable t_i^s represents the starting time of activity $i \in \mathcal{A}^*$, and each temporal variable t_i^f represents the finishing time of an activity $i \in \mathcal{A}^*$.
- Temporal variable t_0 representing the start of the planning period.

Furthermore, we use the following constraints:

- To model the activity processing time of an activity $i \in \mathcal{A}^*$, we add the constraint $t_i^f - t_i^s \geq p_i$ (Equation 2.3). We also add the constraint $t_i^s - t_0 \geq r_i$ to model the release dates (Equation 2.4), and the constraint $t_i^f - t_0 \leq d_i$ to model the deadlines (Equation 2.5). Note that these are also added for the dummy activities st_k, br_k, and fi_k. Because of this, we do not need to model the staff member shifts explicitly in the STN.
- For each precedence constraint $i \prec j \in \mathcal{P}$, we add the constraint $t_i^f \leq t_j^s$ to C. This corresponds to Equation 2.1.

EXAMPLE. The STN shown in Figure 3.2 represents an instance with four activities 1, 2, 3, 4, with 1 and 3 having a release date, 2 and 4 having a deadline, and precedence constraints between 1 and 2 and between 3 and 4. There is one staff member k, with shift $[rs_k, rf_k]$. We now have the following:

- Suppose that $d_2 \geq r_1 + p_1 + p_2$ and $d_4 \geq r_3 + p_3 + p_4$. The graph does not contain any negative weight cycles, so, according to Property 3.1, the instance is feasible.
- Now suppose that $d_2 < r_1 + p_1 + p_2$ and $d_4 \geq r_3 + p_3 + p_4$. The graph now contains a negative weight cycle, $[t_0, t_2^f, t_2^s, t_1^f, t_1^s, t_0]$, with total weight $d_2 - p_2 - p_1 - r_1 < 0$. This means that, according to Property 3.1, the instance is infeasible.
- Using Property 3.3, we find that

$$et(t_1^s) = r_1 \qquad lt(t_1^s) = d_2 - p_1 - p_2$$
$$et(t_1^f) = r_1 + p_1 \qquad lt(t_1^f) = d_2 - p_2$$
$$et(t_2^s) = r_1 + p_1 \qquad lt(t_2^s) = d_2 - p_2$$
$$et(t_2^f) = r_1 + p_1 + p_2 \qquad lt(t_2^f) = d_2$$

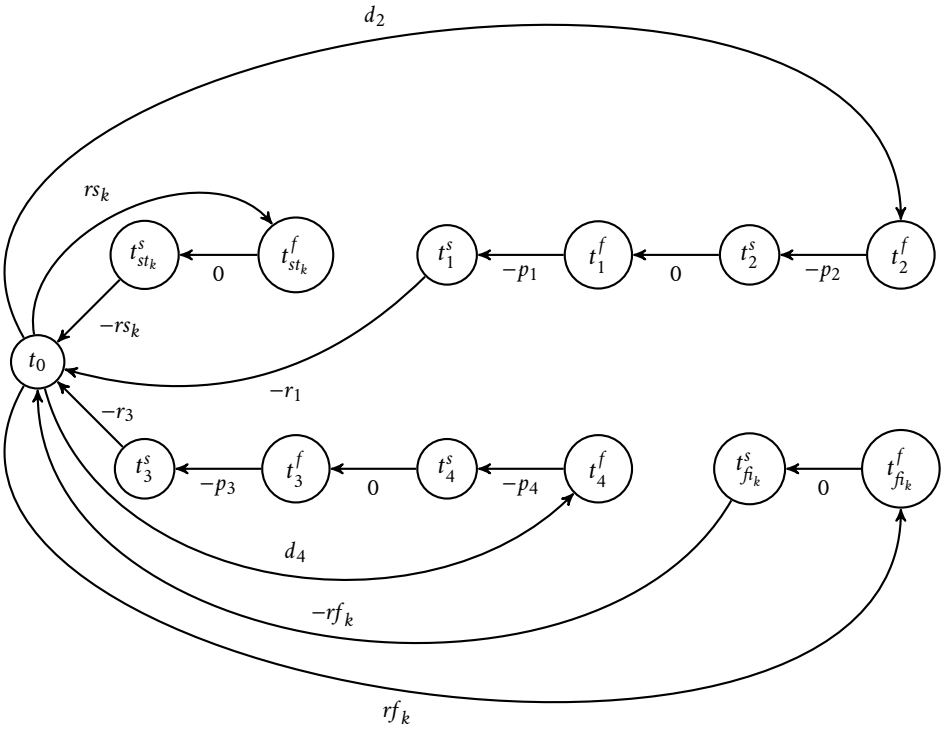


FIGURE 3.2: Example of an instance of the problem modelled as a simple temporal network. This instance has four activities, and one staff member.

3.2.2 Adding lines of work

When a line of work $\omega_k = [st_k = i_0, i_1, \ldots, br_k, \ldots, i_m, i_{m+1} = fi_k]$ is chosen for member of staff $k \in \mathcal{R}$, we add the constraints to C. For each pair of successive activities in the line of work i_j, i_{j+1}, we add the constraint $t^s_{i_{j+1}} - t^f_{i_j} \geq dist(loc^f_{i_j}, loc^s_{i_{j+1}})$, corresponding to Equation 2.2.

Example. Continuing the example from Figure 3.2, an updated network is shown in Figure 3.3, where we have added the line of work 1, 4 for member of staff k. The break is not included in this example, however, it functions the same as any other activity. The constraints added, and the corresponding edges in the graph representation are

$$\begin{aligned} t^s_i - t^f_{st_k} &\geq dist(0, loc^s_1) && (t^s_1, t^f_{st_k}) \\ t^s_4 - t^f_1 &\geq dist(loc^f_1, loc^s_4) && (t^s_4, t^f_1) \\ t^s_{fi_k} - t^f_4 &\geq dist(loc^f_4, 0) && (t^s_{fi}, t^f_4) \end{aligned}$$

With the updated network, the earliest and latest starting time of activity 1 now become the following:

$$\begin{aligned} et(t^s_1) &= \max\{r_1, rs_k + dist(0, loc^s_1)\} \\ lt(t^s_1) &= \min\{(d_2 - p_1 - p_2), \\ &\qquad (d_4 - p_4 - dist(loc^f_1, loc^s_4) - p_1), \\ &\qquad (rf_k - dist(loc^f_4, 0) - p_4 - dist(loc^f_1, loc^s_4) - p_1)\} \end{aligned}$$

3.2.3 Complexity

The complete STN for a problem instance $\langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$ and a solution Ω is $S = (T, C)$, where

$$\begin{aligned} T &= \left\{ t^s_i \;\middle|\; i \in \mathcal{A}^* \right\} \cup \left\{ t^f_i \;\middle|\; i \in \mathcal{A}^* \right\} \cup \{ t_0 \} \\ C &= \left\{ t^f_i - t^s_i \geq p_i \;\middle|\; i \in \mathcal{A}^* \right\} \cup \left\{ t^s_i - t_0 \geq r_i \;\middle|\; i \in \mathcal{A}^* \right\} \\ &\quad \cup \left\{ t^f_i - t_0 \leq d_i \;\middle|\; i \in \mathcal{A}^* \right\} \cup \left\{ t^s_j - t^f_i \geq 0 \;\middle|\; i \prec j \in \mathcal{P} \right\} \\ &\quad \cup \left\{ t^s_i - t^f_{rpre_i} \geq dist(loc^f_{rpre_i}, loc^s_i) \;\middle|\; i \in \mathcal{A}^{br,fi} \right\} \end{aligned}$$

Since $\mathcal{A}^*$ consists of the activities in $\mathcal{A}$ and three dummy activities for each member of staff in $\mathcal{R}$, the number of temporal variables is $|T| = 2|\mathcal{A}| + 3|\mathcal{R}| + 1$. Because we assume that the number of staff members is much smaller than the number of activities (Assumption 2.3), the number of temporal variables is $|T| = O(|\mathcal{A}|)$.

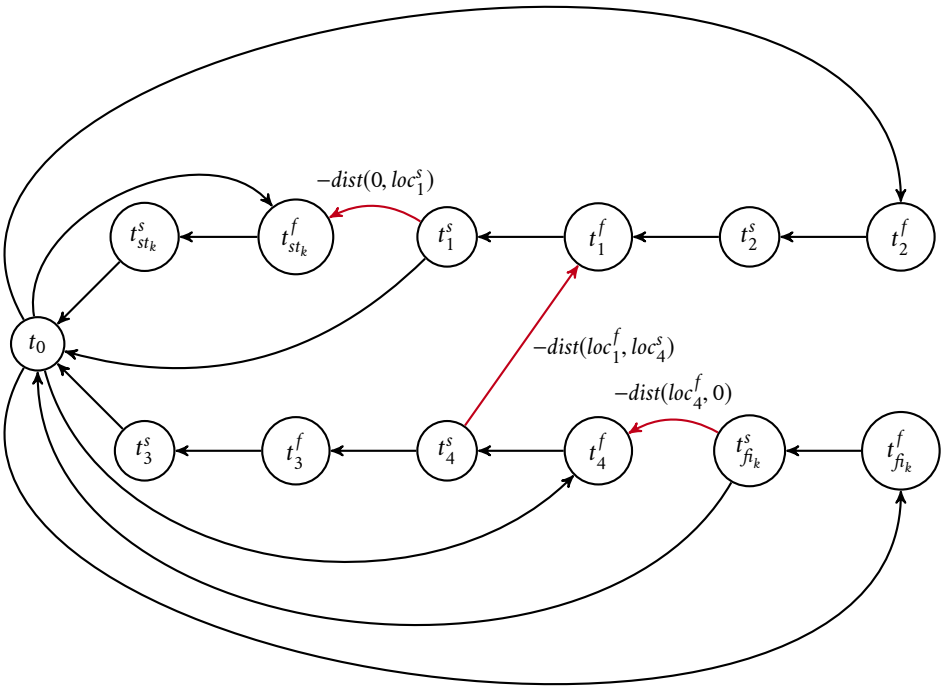


FIGURE 3.3: The same example as in Figure 3.2. Line of work $\omega_k = [st_k, 1, 4, fi_k]$ selected for member of staff k. New arcs are in shown in red.

The set of constraints C consists of constraints for the precedence constraints in $\mathcal{P}$, resulting in $|\mathcal{P}|$ constraints. C also contains constraints representing the release dates, deadlines, and durations, resulting in at most $3\,|\mathcal{A}^*|$ constraints. Furthermore, C contains the line of work constraints indicating the predecessors of each activity, resulting in $\mathcal{A}^* - \mathcal{R}$ constraints, since each activity, except for the st_k activities has a predecessor. In total this is $|C| = |\mathcal{P}| + |\mathcal{A}^*| - |\mathcal{R}| + 3\,|\mathcal{A}^*|$. Once again, we assume that the number of staff members is much smaller than the number of activities (Assumption 2.3). In the worst case, we have a quadratic number of precedence constraints, giving $|C| = O(|\mathcal{A}|^2)$ constraints in the STN. In the case of a sparse graph, we have $O(|\mathcal{A}|)$ constraints.

Note that our model is almost a directed acyclic graph. In the figures, all arcs are directed towards t_0, except for the edges that model deadlines. This enables us to simply calculate the earliest starting times for each variable, using an algorithm similar to the algorithm discussed in Section 2.4, instead of having to use an algorithm designed for more complex STNs. A formulation of the STN as a directed acyclic graph is given in Section 3.4.1.

3.3 Flexibility

The flexibility of an STN is a measure indicating the amount of freedom present in assigning time values to the variables in T, while keeping the constraints in C satisfied. This section is based on and paraphrased from the paper by Wilson et al. [37]. The examples used are the STNs S_1 and S_2 from Figures 3.1a and 3.1b.

3.3.1 Naive flexibility

A simple measure is to use the sizes of the intervals $[et(t_i), lt(t_i)]$:

$$flex_N(S) = \sum_{t_i \in T} (lt(t_i) - et(t_i))$$

While this measure is exact in the flexibility of each individual variable $t \in T$, simply using the sum does not give meaningful results, as it misses dependencies that might exist between the times of events. Because of this, it often overstates the flexibility of STNs. The authors give an example using the STNs S_1 and S_2:

Example. Using $flex_N$, the flexibility of S_1 (see Figure 3.1a) is

$$\begin{aligned} flex_N(S_1) &= lt(t_1) - et(t_1) + lt(t_2) - et(t_2) + lt(t_3) - et(t_3) \\ &= 5 - 0 + 5 - 0 + 5 - 0 = 15 \end{aligned}$$

Likewise, the flexibility of S_2 (see Figure 3.1b) is also 15. However, intuitively, the flexibility of S_2 should be lower than the flexibility of S_1, as in S_2 the choice in values for the events may influence the choice in value for the other events. More precisely, in S_2 we have

$$0 \leq lt(t_1) \leq lt(t_2) \leq lt(t_3) \leq 5$$

As such, intuitively, the flexibility of S_2 should be 5.

3.3.2 Hunsberger's measure of flexibility

To better deal with dependencies between variables, Hunsberger [21] defined a flexibility measure using not only the individual flexibility $lt(t_i) - et(t_i)$ of each $t_i \in T$, but the flexibility between each distinct pair of variables $t_i, t_j \in T$ as well. The flexibility between a pair of variables t_i and t_j is given by

$$flex_H(t_i, t_j) = D_S[t_i, t_j] + D_S[t_j, t_i]$$

The values in D indicate the strongest constraints implied by the constraint set C with respect to the temporal difference between each pair of variables, as shown in Property 3.2. The flexibility of the entire STN is then defined as the naive flexibility summed with the flexibility between each distinct pair of variables. In the notation, τ is any ordering over the variables.

$$flex_H(S) = flex_N(S) + \sum_{t_i \in T} \sum_{\substack{t_j \in T \\ \tau(t_j) > \tau(t_i)}} flex_H(t_i, t_j)$$

While $flex_H$ is an improvement over $flex_N$, it still has shortcomings, as it is still not able to capture the dependencies between the flexibilities of variables in a satisfactory way.

Example. Using $flex_H$, the flexibility of S_1 (see Figure 3.1a) is

$$\begin{aligned} flex_H(S_1) &= flex_N(S_1) + D_{S_1}[t_1, t_2] + D_{S_1}[t_2, t_1] + D_{S_1}[t_1, t_3] \\ &\quad + D_{S_1}[t_3, t_1] + D_{S_1}[t_2, t_3] + D_{S_1}[t_3, t_2] \\ &= 15 + 5 + 5 + 5 + 5 + 5 + 5 = 45 \end{aligned}$$

Likewise, the flexibility of S_2 (see Figure 3.1b) is 30.

3.3.3 Concurrent flexibility

Wilson et al. [37] propose a measure that does capture the dependencies between events known as concurrent flexibility. The method is based on a concept they call *interval schedules*. Where a regular schedule simply assigns a time value to each variable, an interval schedule assigns an interval of time values to each variable. A schedule can then be obtained by selecting any value from the corresponding interval for each variable.

Definition 3.4. Let $S = (T, C)$ be an STN. An *interval schedule* $\iota : T \to \mathbb{R} \times \mathbb{R}$ is an assignment of intervals to all variables $t_i \in T$, such that any selection of values from these intervals yields a feasible schedule σ for S. The interval in ι for variable t_i is denoted as $\iota(t_i) = [l_{t_i}, u_{t_i}]$.

The difference between using an interval schedule and using the earliest and latest starting times from Property 3.3 is that the intervals in the interval schedule cannot change because of dependencies. When forming a schedule σ from the earliest and latest starting times, selecting a value $\sigma(t_i)$ can change the intervals of the other variables. When forming a schedule σ based on an interval schedule ι, this cannot happen.

Example. In the STN S_2 from Figure 3.1b, the intervals obtained when using Property 3.3 are $[0, 5]$ for t_1, t_2, and t_3. If we form a schedule where $\sigma(t_1) = 5$, the intervals for the variables t_2 and t_3 change to $[5, 5]$. As such, it is not possible to arbitrarily pick any value from each of the three intervals to obtain a schedule. An example of an interval schedule for S_2 would be $\iota(t_1) = [0, 2]$, $\iota(t_2) = [2, 3]$, and $\iota(t_3) = [3, 5]$.

The concurrent flexibility is similar to the naive flexibility, however, instead of using the sizes of the intervals $[et, lt]$, it uses the sizes of an interval schedule ι, which is selected in such a way that the sum of the sizes is maximal:

$$\mathit{flex}(S) = \sum_{t_i \in T} |\iota(t_i)|$$

Example. Using *flex*, the flexibility of S_1 (see Figure 3.1a) is 15, with the following interval schedule:

$$\iota(t_1) = [0, 5] \quad \iota(t_2) = [0, 5] \quad \iota(t_3) = [0, 5]$$

The flexibility of the sequential STN S_2 (see Figure 3.1b) is now 5, with the following interval schedule, among other options:

$$\iota(t_1) = [0, 5] \quad \iota(t_2) = [5, 5] \quad \iota(t_3) = [5, 5]$$

Obtaining a maximal interval schedule using an LP

To find an interval schedule for STN $S = (T, C)$, Wilson et al. [37] use a special STN $S' = (T', C')$, which is derived from S. This is called the double STN. Here, each variable $t_i \in T$ is split into two variables, t_i^- and t_i^+, representing the earliest and latest times at which t_i can be executed respectively, with $t_i^+ \geq t_i^-$. Each constraint $t_j - t_i \leq l_{ij} \in C$ is converted to ensure that the earliest execution time of t_i is at most l_{ij} time units away from the latest execution time of t_j. Like any other STN, S' also contains a dummy variable t_0'.

Definition 3.5. The *double STN* $S' = (T', C')$ for the STN $S = (T, C)$ is derived as follows:

$$\begin{aligned} T' &= \left\{ t_i^+, t_i^- \;\middle|\; t_i \in T \right\} \cup \left\{ t_0' \right\} \\ C' &= \left\{ t_j^+ - t_i^- \leq l_{ij} \;\middle|\; t_j - t_i \leq l_{ij} \in C \right\} \\ &\quad \cup \left\{ t_i^- - t_i^+ \leq 0 \;\middle|\; t_i \in T \right\} \\ &\quad \cup \left\{ t_0' - t_0^- \leq 0, t_0^+ - t_0' \leq 0 \right\} \end{aligned}$$

A solution $\sigma' : T' \to \mathbb{R}$ corresponds to an interval schedule ι for S, where the interval for variable $\iota(t_i)$ is $[\sigma'(t_i^-), \sigma'(t_i^+)]$. As said before, the sizes need to be maximized. This can be done by solving the following linear program:

$$\max \mathit{flex}(S) = \sum_{t_i \in T} (t_i^+ - t_i^-)$$

subject to

$$
\begin{aligned}
& t_i^- \leq t_i^+ && \forall t_i \in T \\
& t_j^+ - t_i^- \leq l_{ij} && \forall (t_j - t_i \leq l_{ij}) \in C \\
& t_0^+ = 0 \\
& t_0^- = 0
\end{aligned}
$$

Obtaining a maximal interval schedule using a matching algorithm

The concurrent flexibility of an STN can be computed by finding a perfect minimum weight matching on a bipartite graph, as shown by Mountakis et al. [29]. The matching algorithm works by computing a matching on a complete bipartite graph specified by the augmented distance matrix D_S^*. This is defined as follows:

DEFINITION 3.6. Let $G_S^m = (T^+ \cup T^-, E, w)$ be the matching graph for STN $S = (T, C)$. G_S^m has

$$
\begin{aligned}
T^+ &= \left\{ t_i^+ \;\middle|\; t_i \in T \right\} \\
T^- &= \left\{ t_i^- \;\middle|\; t_i \in T \right\} \\
E &= \left\{ (t_i^+, t_j^-) \;\middle|\; t_i, t_j \in T \right\} \\
w((t_i^+, t_j^-)) &= D_S^*[t_j, t_i]
\end{aligned}
$$

where D_S^* is the augmented distance matrix, defined as

$$
D_S^*[t_i, t_j] = \begin{cases} D_S[t_0, t_i] + D_S[t_i, t_0] & \text{if } t_i = t_j \\ D_S[t_i, t_j] & \text{otherwise} \end{cases}
$$

The cost of a minimum weight perfect matching in graph G_S^m now corresponds to the concurrent flexibility of S. The computation cost of this algorithm is $O(|T|^3)$, since we first need to compute the distance matrix, which takes $O(|T|^3)$ time, for example by using the algorithm by Floyd [14] and Warshall [36]. We then need to calculate the matching, which takes $O(|T|^3)$ time, by using the algorithm by Kuhn [27].

3.4 IMPROVING PERFORMANCE

3.4.1 Finding the distance matrix

If the STN $S = (T, C)$ only contains constraints of the form $t_i - t_j \geq l_{ij}$ for $t_j \neq t_0$, and only constraints of the form $t_i - t_0 \leq l_{i0}$, then the process of finding the distance matrix can be sped up. This is the case when modeling activities with release dates, deadlines, precedence constraints with a lower bound, and processing times given as lower bounds. In this situation, then the corresponding distance graph $G_S = (V, A)$ contains only edges (t_i, t_j) with either $w((t_i, t_j)) \leq 0$ and $t_i \neq t_0$, or with $w((t_i, t_j)) \geq 0$ and $t_i = t_0$. This means that we can convert G into a directed acyclic graph by

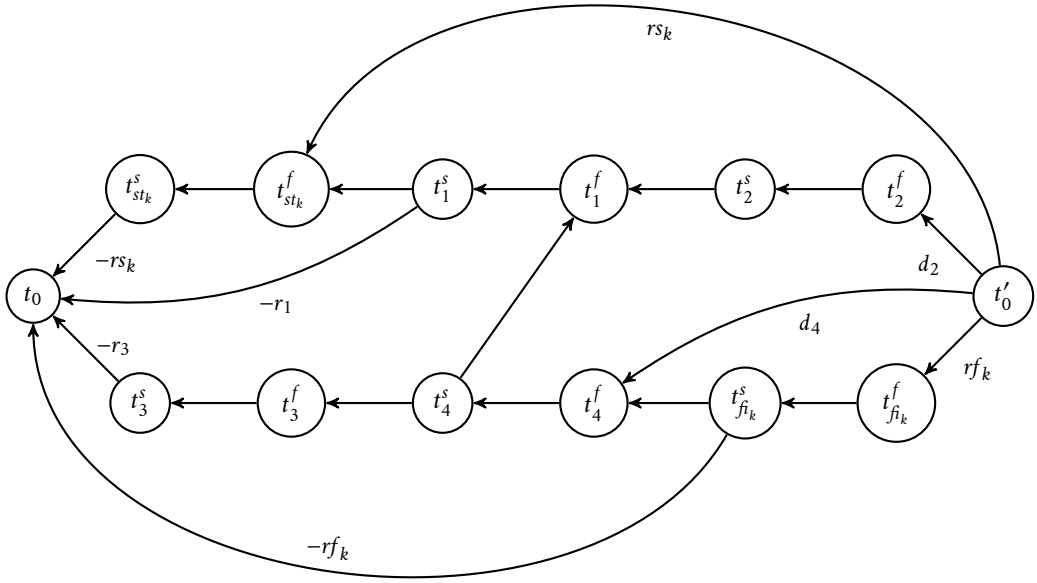


FIGURE 3.4: Directed acyclic graph resulting from converting the distance graph shown in Figure 3.3. Not all edge weights are shown, the missing edge weights remain equal to their value in Figure 3.3.

changing each edge (t_0, t_i) to have a new dummy source t'_0. This gives us the following directed acyclic graph $G'_S = (V', A')$, with

$$\begin{aligned} V' =& V \cup \{ t'_0 \} \\ A' =& \{ (t_i, t_j) \mid (t_i, t_j) \in A \wedge t_i \neq t_0 \} \cup \{ (t'_0, t_i) \mid (t_0, t_i) \in A \} \end{aligned}$$

EXAMPLE. The directed acyclic graphs resulting from the conversion of the distance graph shown in Figure 3.3 is shown in Figure 3.4.

The length of the shortest path between two vertices t_i and t_j in G_S is now given by the minimum of the length of shortest path between t_i and t_j in G'_S and the length of the shortest path between t_i and t_0 together with the length of the shortest path between t'_0 and t_j. This means we can compute the distance matrix D_S as follows:

$$D_S[t_i, t_j] = \min \{ D'_S[t_i, t_j], D'_S[t_i, t_0] + D'_S[t'_0, t_j] \}$$

We can find the lengths of the shortest paths between a vertex $v \in V'$ and each other vertex in V' in a directed acyclic graph in $O(|V'|+|A'|)$ time, see Cormen et al. [8]. If we execute the single-source algorithm for each vertex, we get the complete distance matrix. This takes $O(|V'|^2 + |V'|\,|A'|)$ time. If the network is dense, this is in $O(|V|^3)$, however, if the network is sparse, the runtime is in $O(|V|^2)$. In the case of a sparse graph, this algorithm is an improvement on using the algorithm

by Floyd [14] and Warshall [36], which runs in $O(|V|^3)$, and an improvement on the algorithm by Johnson [22], which runs in $O(|V|^2 \log |V| + |V|\,|A|)$ time.

Note that since this optimization only affects the graph representation of the STN, but not the contents of the STN, the flexibilities are not affected in any way. Only the runtime of finding the shortest path matrix is affected.

3.4.2 Reducing the number of variables

In our model, we model the starting and finishing times of activities as two separate variables. However, this is not completely necessary, as we do not use exact processing times. Instead of saying that the processing time of an activity i is the lower and upper bound on the time that can be spent on an activity, we simply say that it is only a lower bound. Furthermore, the only relation between the starting time and finishing time variable of a particular activity is the constraint representing the processing time. This means that we can contract these two variables into one variable. Each variable t_i, except t_0, now represents the completion time of activity $i \in \mathcal{A}^*$. The processing times are modeled using constraints in which t_i is the later activity. For example, the precedence constraint $i \prec j$, which normally would result in $t_j^s - t_i^f \geq 0$ now results in $t_j - t_i \geq p_j$. Using this approach gives us the STN $S^f = (T^f, C^f)$, where

$$
\begin{aligned}
T^f &= \{\, t_i \mid i \in \mathcal{A}^* \,\} \cup \{\, t_0 \,\} \\
C^f &= \{\, t_i - t_0 \geq r_i + p_i \mid i \in \mathcal{A}^* \,\} \\
&\quad \cup \{\, t_i - t_0 \leq d_i \,\} \cup \left\{\, t_j - t_i \geq p_i \;\middle|\; i \prec j \in \mathcal{P} \,\right\} \\
&\quad \cup \left\{\, t_i - t^f_{rpre_i} \geq dist(loc^f_{rpre_i}, loc^s_i) + p_i \;\middle|\; i \in \mathcal{A}^{br,fi} \,\right\}
\end{aligned}
$$

Note that this change does not affect the asymptotic runtimes of any algorithms, as the number of variables is still $O(|\mathcal{A}|)$, and the number of constraints is still $O(|\mathcal{A}|^2)$ or $O(|\mathcal{A}|)$ in a dense or sparse network respectively, which is the same as in the model given in Section 3.2. Of course, halving the number of variables still leads to a two times improvement in linear-time algorithms, four times in quadratic-time algorithms and eight times in cubic-time algorithms.

Example. The smaller STN resulting from the removal of starting time variables of the STN shown in Figure 3.3 is shown in Figure 3.5.

Reducing the number of variables does affect the flexibility measures: since the naive flexibility, and the method by Hunsberger [21] work by summing up the flexibilities per variable or per pair of variables, reducing the number of variables in the STN also reduces the flexibility value found. The flexibility measure by Wilson et al. [37] is not impacted by the reduction in the number of variables. Since this method uses interval schedules, the intervals found for the starting and completion time variables t_i^s and t_i^f in S can be combined to form the interval of the single variable t_i in S^f. As such, the sum of the sizes of the intervals does not change.

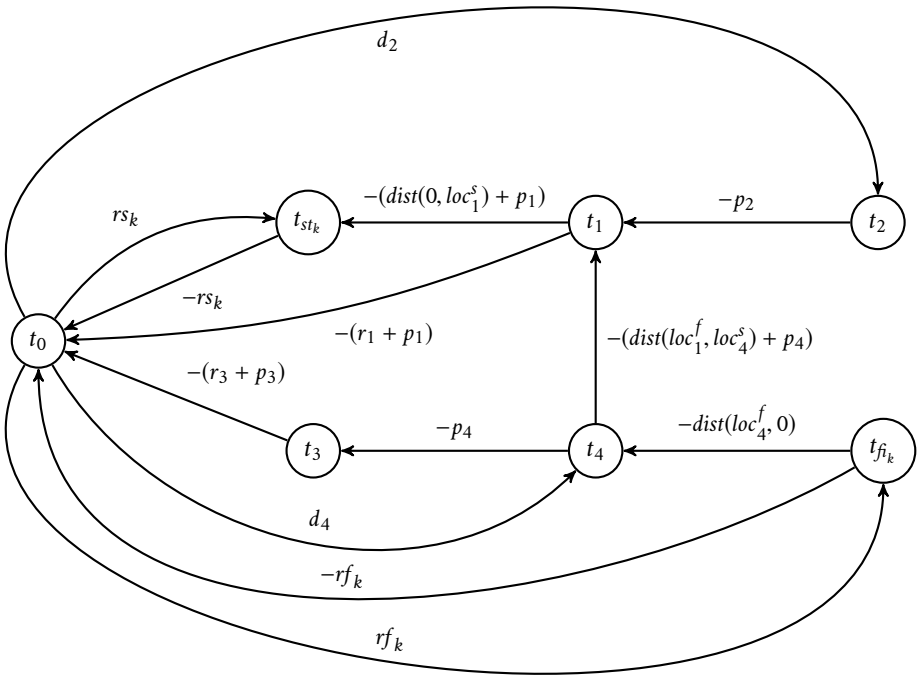


FIGURE 3.5: Smaller STN resulting from converting the distance graph shown in Figure 3.3.

(A) S_a (B) S_b

FIGURE 3.6: Two example STNs to illustrate the effect of removing variables on the flexibility.

EXAMPLE. In Figure 3.6a, an STN with one activity j is shown. We have $r_j \neq 0$, $p_j \neq 0$, and $d_j \neq 0$. We denote this STN S_a. In Figure 3.6b, the same STN is shown, but converted to only contain completion time variables. This STN is S_b. The naive flexibilities of S_a and S_b are

$$\begin{aligned} \mathit{flex}_N(S_a) &= lt(t_j^s) - et(t_j^s) + lt(t_j^f) - et(t_j^f) = (d_j - p_j) - r_j + d_j - (r_j - p_j) = 2d_j - 2p_j - 2r_j \\ \mathit{flex}_N(S_b) &= lt(t_j) - et(t_j) = d_j - r_j - p_j \end{aligned}$$

As can be seen, in this case, the difference between the converted STN and the original STN is that the flexibility is halved. The flexibilities when measured using the method by Hunsberger [21] of S_a and S_b are

$$\begin{aligned} \mathit{flex}_H(S_a) &= \mathit{flex}_N(S_a) + D_{S_a}[t_j^s, t_j^f] + D_{S_a}[t_j^f, t_j^s] = (2d_j - 2p_j - 2r_j) + (-r_j) + (d_j - p_j) \\ &= 3d_j - 3p_j - 3r_j \\ \mathit{flex}_H(S_b) &= \mathit{flex}_N(S_b) = d_j - r_j - p_j \end{aligned}$$

Like with the naive flexibility, there is a difference between the flexibilities. In this case, the difference is even greater than with the naive flexibility. The concurrent flexibility, which is calculated using interval schedules, is the same in both cases. In S_a, a maximal interval schedule is $\iota(t_j^s) = [r_j, r_j + \frac{1}{2}p_j]$ and $\iota(t_j^f) = [r_j + \frac{1}{2}p_j, d_j]$. In S_b, there is only one variable, so the interval schedule has $\iota(t_j) = [r_j + p_j, d_j]$. In both cases, the concurrent flexibility is equal to $d - r_j - p_j$.

4 HEURISTIC APPROACH

In this chapter, we will give our method for finding solutions for the problem described in Chapter 2. First, we will show the objective functions used in Section 4.1. Our method to find a solution consists of two steps. First, we generate an initial solution using a greedy heuristic. This is shown in Section 4.2. If this heuristic is unable to find a feasible solution, we try to give the time window that contains too many activities for the given members of staff. This is shown in Section 4.5. Then, we use local search to improve this solution, which we show in Section 4.3. We make use of incremental solutions in both the greedy heuristic and the local search improvement step. This is detailed in Section 4.4.

4.1 Objective functions

We use several objectives to measure the quality of a solution. The different objective values are added together using weights to obtain one objective value. The weights will be determined using experimentation. This sum will be maximized.

Flexibility We have incorporated flexibility in the objective by using the concurrent flexibility metric, see Section 3.3.3. As we want to maximize flexibility, we want the weight $w_{flexibility}$ to be positive.

Fairness Fairness is incorporated in the objective function as the average standard deviation of the workloads of each skill. Let σ_s be the standard deviation of workloads in skill $s \in \mathcal{S}$:

$$\sigma_s = \sqrt{\frac{1}{|\mathcal{R}_s|} \sum_{k \in \mathcal{R}_s} (wl_{\omega_k} - wl_{avg,s})^2}$$

where $wl_{avg,s} = \frac{1}{|\mathcal{R}_s|} \sum_{k \in \mathcal{R}_s} wl_{\omega_k}$. We then try to minimize the maximum of the standard deviations:

$$\min \max_{s \in \mathcal{S}} \sigma_s$$

As we try to minimize the standard deviation, we want the weight $w_{fairness}$ to be negative, resulting in a penalty that becomes smaller the closer the standard deviation is to zero.

Walking distances We try to minimize the walking distances for the members of staff:

$$\min \sum_{k \in \mathcal{R}} wd_{\omega_k}$$

Since we are minimizing the walking distances, we want the weight $w_{walking}$ to be negative.

Number of unscheduled activities During our search, we allow activities to be unscheduled. Of course, this should be penalized in the objective function, to prevent an empty solution from becoming the best solution. In other words, we want to minimize the following:

$$\min \left| \left\{ i \in \mathcal{A}_S \;\middle|\; \nexists k \in \mathcal{R} : i \in \omega_k \right\} \right|$$

Given that a solution with an unscheduled objective is not feasible, we want the weight $w_{unscheduled}$ to be negative, and to be sufficiently negative to ensure that all activities are scheduled.

4.2 Heuristic for initial solution

To obtain an initial feasible solution, we use a greedy approach. We add all activities in a topological ordering, to ensure that no activity is scheduled before its predecessors. For each activity, we examine all possible insertion points, meaning that we check across all lines of work of the correct skill, and all possible indices in each line of work. The activity is then inserted at the position such that the value of the combined objective function is highest and the resulting STN is still consistent. If there is no position at which the activity can be inserted without making the STN inconsistent, we leave the activity unscheduled. This algorithm uses two parameters, B and R.

For each activity $i \in \mathcal{A}_S$ that is inserted, we keep track of a value Δ_i which represents the impact that inserting i at the best possible position had on the objective value of the solution. If i could not be inserted at any position, we have $\Delta_i = -\infty$. This is checked by using Algorithm 2.1. It can be expected that adding a new activity into the schedule will reduce the flexibility of the resulting STN and increase the walking distances. Since we are maximizing, and since we are using the flexibility as main objective and walking distances as penalty, we can expect that adding an activity will result in a value $\Delta_i \leq 0$.

If, when all activities are examined, none are left unscheduled, the solution is returned as the initial feasible solution. If any activities are left unscheduled, we take a ruin-and-recreate approach. During each round, we keep a queue Q', which contains the activities that could not be scheduled, and will be scheduled in the next round. If an activity could not be scheduled, it is added to Q' at Line 19. Then, we add the $B - |Q'|$ activities that have had the highest impact on the solution (i.e. the activities with the lowest Δ_i values) and remove them from the schedule, unless removing them would make the STN inconsistent*. In that case, we leave the activity in the schedule. These activities are also added to Q'. The activities in Q' are then readded to the solution in the next round. Note that all unscheduled activities are added to Q', so it may occur that $|Q'| > B$. In that case, no activities are added that have been scheduled.

* This is possible if the removed activity i has $dist(loc^f_{rpre_i}, loc^s_i) + p_i + dist(loc^f_i, loc^s_{rsuc_i}) < dist(loc^f_{rpre_i}, loc^s_{rsuc_i})$. This is most likely to happen if i is an activity involving movement of a train unit, since these have differing start and end locations, and since the processing time is likely to be less than the walking distances.

This process is repeated until we reach a point where all activities are scheduled, or until a certain number of rounds have passed. We denote the maximum number of rounds in this algorithm R. If no solution can be found, we use the local search algorithm with the neighbourhood operators shown in Sections 4.3.2 and 4.3.3 to find a feasible insertion point for the remaining activities.

The pseudocode for this routine is shown in Algorithm 4.1. The functions INSERT and REMOVE are sets of changes to be applied to a solution Ω. These can be applied and reverted to obtain the new candidate solution and to reject the new candidate solution respectively. This is done to avoid copying the entire solution, and to avoid large computational costs in calculating the objective function. This is detailed further in Section 4.4.

4.2.1 Complexity

The number of positions (and corresponding solutions) that we need to examine differs per activity. Suppose that each activity belongs to the same skill. In the case of the first activity i_1 to be scheduled, each line of work contains only the three dummy activities, so each line of work has only two possible insertion points, either before the break or after the break. This means that we need to examine $2|\mathcal{R}|$ positions. Then for the second activity, one line of work may have an activity in it, meaning that it has three positions to check. The total number of positions to check is then $2|\mathcal{R}| + 1$. Generalizing this means that when inserting the kth activity we need to check $2|\mathcal{R}| + k - 1$ solutions. Thus, the total number of insertion points that we need to check for the first round is

$$\sum_{k=1}^{n}(2m + k - 1) = 2nm + 1 + 2 + \cdots + n - n = 2nm + T_{n-1} = 2nm + \frac{n(n-1)}{2}$$

where $n = |\mathcal{A}_{\mathcal{S}}|$, $m = |\mathcal{R}|$, and T_n is the nth triangular number. In subsequent rounds, we do not have to check each activity, we only need to examine the $B \leq n$ activities that have had the most impact. This means that the number of solutions we need to check in each round is $Bm + T_B$. Since we only allow R rounds, and each insertion point results in a partial solution, this gives us the total number of solutions to check of

$$nm + T_{n-1} + R(nm + T_B)$$

which is in $O(n^2 + nm)$, since both R and T_B are constant factors, and since $T_{n-1} = \frac{n(n-1)}{2}$. If the constraint graph is sparse, we have to check $O(n^2)$ solutions.

4.3 IMPROVEMENT USING LOCAL SEARCH

4.3.1 Algorithm

In our improvement step, we make use of the simulated annealing algorithm, introduced by Kirkpatrick et al. [24]. The main outline of this algorithm is shown in Algorithm 4.2. The consistency of candidate solutions is checked using Algorithm 2.1. The initial solution is Ω_{init}.

Unlike standard hill climbers, which only accept new solutions if they are an improvement on the current solution, simulated annealing also allows a decrease in objective value. This is done by keeping a temparature, which determines the probability that a solution that is worse than the current solution is accepted. This is shown in Line 11. At the start of the process, this probability

ALGORITHM 4.1: Outline of the greedy and ruin-and-recreate algorithm to generate a feasible solution.

Input: Problem instance $\langle \mathcal{A}, \mathcal{L}, \mathcal{P}, \mathcal{R}, \mathcal{S}, dist \rangle$, B, R
Output: Solution Ω if one was found, else **nil**

1 $Q \leftarrow \mathcal{A}_{\mathcal{S}}$ ordered in topological ordering
2 $round \leftarrow 1$
3 $\Omega \leftarrow$ empty solution
4 $Q' \leftarrow \emptyset$ ▷ *Q' contains the activities that will be retried in the next round*
5 **while** $Q = \emptyset$**:**
6 $i \leftarrow$ DEQUEUE(Q) ▷ *Scheduling activity i*
7 $\delta_{best} \leftarrow$ **nil** ▷ *Best solution found with i added*
8 $\delta_{best}.objective \leftarrow -\infty$
9 **for** $k \in$ SHUFFLE($\mathcal{R}_{rq_i}$)**:**
10 **for** $1 \leq index < |\omega_k| - 1$**:** ▷ *Indices are chosen such that i is not placed before st_k or after fi_k*
11 $\delta \leftarrow$ INSERT($i, \omega_k, index$)
12 $\Omega \leftarrow \delta$ applied to Ω
13 **if** Ω is consistent and $\Omega.objective \geq \delta_{best}.objective$**:**
14 $\delta_{best} \leftarrow \delta$
15 $\Omega \leftarrow \delta$ rolled back from Ω
16 **if** $\delta_{best} \neq$ **nil:** ▷ *A feasible insertion point for i has been found*
17 $\Delta_i \leftarrow \delta_{best}.objective - \Omega.objective$
18 $\Omega \leftarrow \delta$ applied to Ω
19 **else:**
20 ENQUEUE(Q', i)
21 **if** $Q' = \emptyset$**:** ▷ *Feasible solution found*
22 **return** Ω
23 **if** $round \leq R$**:**
24 **for** $|Q'| - B$ times**:** ▷ *Removing the activities with the highest effect*
25 $i \leftarrow \text{argmin}_{j \in A'} \Delta_i$
26 $\omega_k \leftarrow$ the line of work containing i
27 $\delta \leftarrow$ REMOVE(i, ω_k)
28 $\Omega \leftarrow \delta$applied to Ω
29 **if** Ω is consistent**:**
30 $\Delta_i \leftarrow \infty$
31 ENQUEUE(Q, i)
32 **else:** ▷ *Removing i makes the STN inconsistent*
33 $\Omega \leftarrow \delta$ rolled back from Ω
34 $Q \leftarrow Q'$
35 $round \leftarrow round + 1$
36 **go to** Line 5 ▷ *Retry adding the removed activities*
37 **return nil** ▷ *No feasible solution was found*

should be relatively high, and it should decrease gradually as the algorithm runs. If there are activities that remain unscheduled in the initial solution Ω_{init}, we increase the temperature with the amount of unscheduled activities multiplied by a factor $T_{unscheduled}$, to improve diversification. The base value for the initial temperature T_{init}. We use a cooling scheme where the temperature is multiplied by a factor $\beta_c < 1$ every t_c iterations. This happens at Line 19. We use a stopping condition where the algorithm stops if the best solution found has not been updated in $it_{\max}$ iterations

ALGORITHM 4.2: The simulated annealing algorithm by Kirkpatrick et al. [24]

Input: Parameters $T_{unscheduled}, T_{init}, t_c, \beta_c, it_{\max}, \Omega_{init}$
Output: Solution Ω

1 $\Omega_{best} \leftarrow \Omega_{init}$
2 $\Omega \leftarrow \Omega_{init}$
3 $T \leftarrow T_{init} + T_{unscheduled} \cdot \left|\left\{ i \in \mathcal{A}_S \;\middle|\; \nexists \omega \in \Omega_{init} : i \in \omega \right\}\right|$
4 $it \leftarrow 0$
5 **while** stop condition not met**:**
6 $\quad \delta \leftarrow$ changes to get successor of Ω obtained from neighbourhood
7 $\quad objective_\Omega \leftarrow \Omega.objective$ ▷ *Storing old objective for comparisons*
8 $\quad \Omega \leftarrow \delta$ applied to Ω
9 $\quad$ **if** $\Omega.objective \geq objective_\Omega$**:**
10 $\quad\quad$ **pass** ▷ *Accept the changes from δ*
11 $\quad$ **else if** $\exp\left(\frac{\Omega.objective - objective_\Omega}{T}\right) >$ rand**:**
12 $\quad\quad$ **pass** ▷ *Accept the changes from δ*
13 $\quad$ **else:**
14 $\quad\quad \Omega \leftarrow \delta$ rolled back from Ω ▷ *Reject*
15 $\quad$ **if** $\Omega.objective > \Omega_{best}.objective$**:**
16 $\quad\quad \Omega_{best} \leftarrow \Omega$ ▷ *New best solution found*
17 $\quad it \leftarrow it + 1$
18 $\quad$ **if** $it \mod t_c = 0$**:**
19 $\quad\quad T \leftarrow T \cdot \beta_c$ ▷ *Cooling down*
20 **return** Ω_{best}

4.3.2 Neighbourhood operators for feasible solutions

We use the following neighbourhood operators in the local search improvement step to improve feasible solutions. The operators can only place activities in lines of work belonging to a member of staff with the correct skill, to ensure that an engineer does not get assigned a cleaning activity for example. The operators also take care to not move the dummy activities in $\mathcal{A}^* \setminus \mathcal{A}$ between lines of works. The operators return a set of changes that need to be made to the current solution. The details of these changes are given in Section 4.4.

Bring related together This method aims to improve flexibility by placing activities that are related to each other by way of the precedence constraints in the same line of work. This is done

by selecting a random line of work $\omega_k \in \Omega$, and picking a random activity $i \in \omega_k \cap \mathcal{A}$. We then select a related activity $j \in apre_i \cup asuc_i$ with $rq_i = rq_j$. The line of work in which this activity is currently scheduled is ω_m. We then pick an unrelated activity $j' \in unrl_i \cap \omega_k \cap \mathcal{A}$ to replace with j in ω_k. The related activity j is then swapped with the unrelated activity j'. When this is done, both i and j are in ω_k. This operator returns Swap(ω_k, j', ω_m, j).

Place immediate predecessor This operator is a stronger version of the *Bring related together* operator. It aims to reduce the number of constraints added by the lines of work compared to the existing precedence constraints by placing the activities of an existing precedence constraint directly following in the same line of work. The rationale behind this is that the extra constraints reduce the flexibility of the schedule. We first select a random precedence constraint $i \prec j \in \mathcal{P}$, with $rq_i = rq_j$ and $loc_i^f = loc_j^s$. The line of work containing i is denoted as ω_k and the line of work containing j is denoted as ω_m. We then place i directly in front of j in ω_m. Now, ω_m contains the subsequence $[\ldots, i, j, \ldots]$. The operator returns Move(ω_k, i, ω_m, j).

Relieve largest line of work This operator removes a random activity from the line of work with the largest workload and places it at a random position in the line of work with the smallest workload. This is done to improve the fairness of the schedule. We first select a random skill $s \in \mathcal{S}$. We then pick the largest line of work $\omega_k = \text{argmax}_{k \in \mathcal{R}_s} wl_{\omega_k}$ and the smallest line of work $\omega_m = \text{argmin}_{m \in \mathcal{R}_s} wl_{\omega_m}$. We then select random activities $i \in \omega_k \cap \mathcal{A}$ and $j \in \omega_m \cap \mathcal{A}$. We then place i directly in front of j in ω_m. The operator returns Move(ω_k, i, ω_m, j).

Swap within line of work This operator is included for some diversification. It selects a random line of work $\omega_k \in \Omega$ and selects two activities $i, j \in \omega_k \cap \mathcal{A}^{br}$. These are then swapped in position. We also allow the break activity to be swapped in this operator. The operator returns Swap(ω_k, i, ω_k, j).

4.3.3 Neighbourhood operators for infeasible solutions

We also use the local search algorithm to find a feasible solution, if the greedy heuristic shown in Section 4.2 is not able to do so. In addition to the neighbourhood operators from Section 4.3.2, we now use the following neighbourhood operators. These are all able to deal with unscheduled activities. These operators are also used for diversification in the case of a feasible solution.

Insert random This operator takes a random unscheduled activity $i \in \mathcal{A}$ and inserts it in the line of work of a member of staff k of the correct type ($rq_i = rsk_k$), at the position currently occupied by $j \in \omega_k \setminus \{ st_k \}$. j and any subsequent activities shift to a later point in time. Returns Insert(ω_k, i, j). This operator is only selected if there is at least one unscheduled activity.

Delete random This operator takes a random line of work $\omega_k \in \Omega$ and removes a random activity $i \in \mathcal{A} \cap \omega_k$ from the line of work. Returns Remove(ω_k, i).

Replace random This operator is a combination of the previous two operators: it takes a random line of work $\omega_k \in \Omega$ and removes a random activity $i \in \mathcal{A} \cap \omega_k$ from the line of work. Then, an unscheduled activity $j \in \mathcal{A}_{rq_i}$ is picked randomly. j is then inserted in ω_k at the position previously occupied by i. Returns Replace(ω_k, i, j). This operator is only selected if there is at least one unscheduled activity.

4.4 Incremental solutions

To improve on the speed of the local search algorithm, we make use of sets of changes that we can apply to and revert from solutions. This means that each neighbourhood operator does not return a complete new candidate solution Ω', but instead a list of changes δ that need to be applied to the current solution to get the new solution $\Omega + \delta$. These are then applied to the current solution to check the feasibility and objective of the resulting solution. If accepted, the changes are kept (Line 8), if rejected the changes are rolled back (Line 14). This improves performance since we do not need to copy the entire candidate solution and associated data structures each iteration. Instead, we only need to change the relevant part of the candidate solution. As such, we can use the same object and data structures during the entire local search process.

The contents of a changeset are dependent on the type of change: for example, in the case of a swap, it simply tracks the indices and lines of work that need to be swapped. However, all types of changesets keep track of the edges that need to be deleted from the STN and the edges that need to be added to the STN to keep the STN representation consistent with the contents of the candidate solution.

Using incremental solutions is especially helpful in our case, if we use an LP-solver to find the value of the flexibility objective. Normally, we would have to solve the entire LP from scratch in each iteration, but using incremental solutions allows us to reuse the solution of the LP for the current solution Ω to calculate the flexibility of the solution with the changes applied $\Omega + \delta$. With the walking distance objective, we can obtain the objective for $\Omega + \delta$ by subtracting the walking distances associated with the edges that are removed from the STN and adding the walking distances associated with the added edges. The fairness objective is calculated in full each time.

The neighbourhood operators can return two different types of changes. The greedy algorithm uses two different types of changes. Here, we will give the notation we use in Sections 4.2 and 4.3.2 on neighbourhood operators. Edges added have weights corresponding to the walking distances. For example, if edge (t_i^s, t_j^f) is added, its weight will be $-dist(loc_j^f, loc_i^s)$.

Insert The operation Insert(ω_k, i, j) inserts the activity i in line of work ω_k at the index currently occupied by j. Applying this operation removes one constraint from the STN and adds two. The exact changes are shown in Table 4.1.

Remove The operation Remove(ω_k, i) removes the activity i from line of work ω_k. Applying this operation removes two constraints and adds one constraint to the STN. The exact changes are shown in Table 4.1.

Replace The operation Replace(ω_k, i, j) removes the activity i from line of work ω_k and then places activity j in ω_k at the position previously occupied by i. This operation is equivalent to executing Insert(ω_k, j, i) and Remove(ω_k, i) in order. Applying this operation always adds and removes two constraints from the STN. The exact changes are shown in Table 4.1.

Move The operation Move(ω_k, i, ω_m, j) removes activity i from line of work ω_k and places it directly in front of activity j in line of work ω_m. This operation is equal to executing Remove(ω_k, i) and Insert(ω_m, i, j) in order. Applying this operation always adds and removes three constraints from the STN. The exact changes are shown in Table 4.1.

Insert(ω_k, i, j)		Remove(ω_k, i)		Replace(ω_k, i, j)		Move(ω_k, i, ω_m, j)	
Delete	Add	Delete	Add	Delete	Add	Delete	Add
$(t^s_j, t^f_{rpre_j})$			$(t^s_{rsuc_i}, t^f_{rpre_i})$			$(t^s_j, t^f_{rpre_j})$	$(t^s_{rsuc_i}, t^f_{rpre_i})$
	$(t^s_i, t^f_{rpre_j})$	$(t^s_i, t^f_{rpre_i})$		$(t^s_i, t^f_{rpre_i})$	$(t^s_j, t^f_{rpre_i})$	$(t^s_i, t^f_{rpre_i})$	$(t^s_i, t^f_{rpre_j})$
	(t^s_j, t^f_i)	$(t^s_{rsuc_i}, t^f_i)$		$(t^s_{rsuc_i}, t^f_i)$	$(t^s_{rsuc_i}, t^f_j)$	$(t^s_{rsuc_i}, t^f_i)$	(t^s_j, t^f_i)

Table 4.1: The arcs added and removed from the STN when Insert(ω_k, i, j), Remove(ω_k, i), Replace(ω_k, i, j) or Move(ω_k, i, ω_m, j) is executed.

Swap(ω_k, i, ω_m, j) $k = m \wedge i = j + 1$		Swap(ω_k, i, ω_m, j) $k = m \wedge j = i + 1$		Swap(ω_k, i, ω_m, j) Other	
Delete	Add	Delete	Add	Delete	Add
		$(t^s_i, t^f_{rpre_i})$	$(t^s_j, t^f_{rpre_i})$	$(t^s_i, t^f_{rpre_i})$	$(t^s_j, t^f_{rpre_i})$
$(t^s_j, t^f_{rpre_j})$	$(t^s_i, t^f_{rpre_j})$			$(t^s_j, t^f_{rpre_j})$	$(t^s_i, t^f_{rpre_j})$
(t^s_i, t^f_j)	(t^s_j, t^f_i)	(t^s_j, t^f_i)	(t^s_i, t^f_j)		
$(t^s_{rsuc_i}, t^f_i)$	$(t^s_{rsuc_i}, t^f_j)$			$(t^s_{rsuc_i}, t^f_i)$	$(t^s_{rsuc_i}, t^f_j)$
		$(t^s_{rsuc_j}, t^f_j)$	$(t^s_{rsuc_j}, t^f_i)$	$(t^s_{rsuc_j}, t^f_j)$	$(t^s_{rsuc_j}, t^f_i)$

Table 4.2: The arcs added and removed from the STN when Swap(ω_k, i, ω_m, j) is executed.

Swap The operation Swap(ω_k, i, ω_m, j) swaps the activity i in line of work ω_k with the activity j in line of work ω_m. If $k = m$ and $i = j + 1$ or $j = i + 1$, this operation adds and removes three constraints from the STN. Otherwise, this operation adds and removes four constraints from the STN. The exact changes are shown in Table 4.2.

4.5 Handling infeasible solutions

If the algorithm does not find a feasible solution, we will aim to provide some information on the bottleneck using the original schedule σ^{orig}, obtained by converting the solutions from the algorithm by Van den Broek [5]. This gives us a starting and completion time s_i^{orig} and c_i^{orig} for each activity $i \in \mathcal{A}$. We do this by solving an assignment problem, where we assign a predecessor to every activity, while minimizing the walking distances. If we are unable to find an assignment, the bottleneck might lie in that specific interval. This is done per distinct skill and shift period. For example, if we have two staff members with intervals $[rs_1, rf_1]$ and $[rs_2, rf_2]$, with $rs_2 < rf_1$, we solve the assignment problem for the intervals $[rs_1, rs_2]$, $[rs_2, rf_1]$, and $[rf_1, rf_2]$.

The assignment problem corresponds to finding a minimum weight perfect matching in a complete bipartite graph. This can be done by using the algorithm by Kuhn [27]. The formulation given here is loosely based on the assignment formulations for the single-depot vehicle-scheduling problem by Freling et al. [16]. The complete weighted bipartite graph $G = (X \cup Y, E, w)$ for skill

$s \in \mathcal{S}$ and interval $[rs, rf]$ contains two vertices $i \in X$ and $i' \in Y$ for each activity $i \in \mathcal{A}_s$ for which the original execution overlaps with the interval. This subset of activities will be denoted as $\mathcal{A}'$. X and Y also contain the dummy activities st_k and fi_k for all $k \in \left\{ k \in \mathcal{R}_s \;\middle|\; rf_k \geq rs \wedge rs_k \leq rf \right\}$ respectively.

An edge is added between each pair of activities that can be done subsequently. Edges are also added between each st_k and each $i' \in Y$ and between each $i \in X$ and each fi_k. All edges are weighed using the walking distances. To ensure that the graph is complete, dummy edges are added between all pairs of vertices without an edge. The dummy edges have a sufficiently large weight N. More formally:

$$\mathcal{A}' = \left\{ i \;\middle|\; i \in \mathcal{A} \wedge rq_i = s \wedge c_i^{orig} > rs \wedge s_i^{orig} < rf \right\}$$
$$X = \{ i \mid i \in \mathcal{A}' \} \cup \{ st_k \mid k \in \mathcal{R}_s \}$$
$$Y = \{ i' \mid i \in \mathcal{A}' \} \cup \left\{ fi_k \;\middle|\; k \in \mathcal{R}_s \right\}$$
$$E = \{ (x, y) \mid x \in X, y \in Y \}$$
$$w((i, j')) = \begin{cases} 0 & \text{if } i = st_k \vee j' = fi_k \\ dist_{loc_i^f, loc_j^s} & \text{if } c_i + dist_{loc_i^f, loc_j^s} \leq s_j \wedge i \notin asuc_j \\ N & \text{otherwise} \end{cases}$$

A matching $M \subset E$ now gives an assignment of all activities to a predecessor. If we have an edge (i, j') in the matching, that means that activity j is performed directly after activity i. Using these assignments, a set of lines of work is obtained.

Example. In Figure 4.1, an example of the assignment graph is shown. In this example, $\mathcal{A}' = \{ 1, 2, 3, 4, 5, 6 \}$, and $\mathcal{R}_s = \{ 1, 2 \}$. The activities have the following original start and finish times:

$$s_1^{orig} = 15 \quad c_1^{orig} = 18 \quad s_3^{orig} = 10 \quad c_3^{orig} = 16 \quad s_5^{orig} = 25 \quad c_5^{orig} = 32$$
$$s_2^{orig} = 35 \quad c_2^{orig} = 42 \quad s_4^{orig} = 30 \quad c_4^{orig} = 34 \quad s_6^{orig} = 20 \quad c_6^{orig} = 22$$

An example assignment is shown in Figure 4.1 in red. This assignment gives the lines of work $\omega_1 = [st_1, 1, 4, 2, fi_1]$ and $\omega_2 = [st_2, 3, 6, 5, fi_2]$.

If the matching contains an edge with weight N, i.e. there is at least one activity that could not be assigned a predecessor using the non-dummy edges, then the bottleneck might lie in this interval. The imported instances only contain one movement at a time, meaning that it should be possible to execute all activities using only one member of staff of each skill. This is not possible because of the walking distances. The imported instances can contain activities where $s_j^s - c_i^f \leq dist(loc_i^f, loc_j^s)$, meaning that the activities i and j cannot be executed in order because of the walking distances. This means that we need more members of staff.

Note that this method is a simplification of the NSSRP. We only consider the walking distance objective, and we disregard the flexibility and fairness objectives. Furthermore, tasks that fall in two different intervals are not necessarily assigned to the same member of staff in both intervals. It

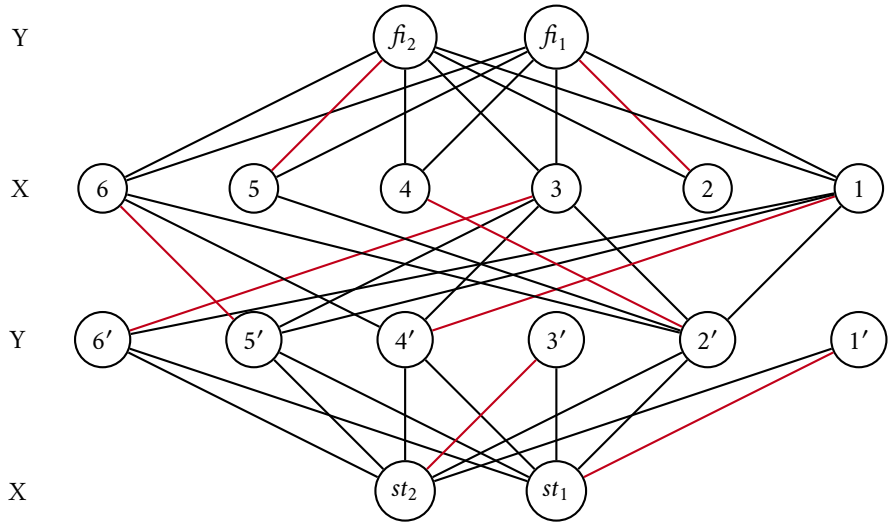


FIGURE 4.1: Example of the bipartite graph used for the assignment problem. An example assignment is shown in red.

also does not take breaks, and walking distances to and from the depot into account. Finally, in the NSSRP, the starting times and finishing times are not fixed, while they are in the assignment problem. If the original plan contains an unfortunate choice in starting times, then a feasible assignment might not be possible, even though executing the activities in series might very well be feasible. For example, suppose that there are two tasks i and j, with $rq_i = rq_j$, $c_i^{orig} = s_j^{orig}$ and $dist(loc_i^c, loc_j^s) \neq 0$, meaning that these two activities cannot be performed in order by a single member of staff. If there is only one member of staff with $rsk_k = rq_i$, then the matching algorithm will always report that the problem lies in the interval and skill containing i and j, even though the reason the greedy heuristic or the local search method were unable to find a solution is with another interval or skill.

5 COLUMN GENERATION APPROACH

In this chapter, we give an alternative formulation for the problem of minimizing the walking distances subject to Constraints 2.8 to 2.23, which enables us to use column generation to find a solution. Like in the other MIP formulation, we do not use the flexibility as the objective function. Instead, we use the walking distances, since these are more suited to a column generation approach. The formulation used is the method to solve parallel machine scheduling with release dates, deadlines, and precedence constraints by Van den Akker et al. [1], with the walking distances as the objective function, instead of the number of selected columns. They use a local search approach to generate new columns. In this chapter, Section 5.1 contains the formulation of the master problem, and Section 5.2 the formulation of the pricing problem.

5.1 Master problem

The master problem consists of minimizing the walking distances by selecting a line of work for each member of staff, such that every activity that needs to be scheduled is in exactly one line of work of the correct type, and such that precedence constraints are respected. The formulation used is based on Let Ω^* be the set containing the candidate lines of work. Each line of work $\omega \in \Omega^*$ represents a day of work for a specific member of staff. Lines of work are specified through the following parameters:

$$a_i^\omega = \begin{cases} 1 & \text{if the line of work } \omega \text{ contains activity } i \in \mathcal{A}^* \\ 0 & \text{otherwise} \end{cases}$$

$$b_k^\omega = \begin{cases} 1 & \text{if the line of work } \omega \text{ is meant for member of staff } k \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases}$$

$$d_{ij}^\omega = \begin{cases} 1 & \text{if the line of work } \omega \text{ goes from activity } i \in \mathcal{A}^* \text{ to activity } j \in \mathcal{A}^* \\ 0 & \text{otherwise} \end{cases}$$

$$c_i^\omega = \text{the completion time of activity } i \in \mathcal{A}^* \text{ in line of work } \omega$$

5.1.1 Mixed integer programming formulation

The master problem consists of making a selection of lines of work out of the set of candidate lines of work Ω^*. This is done while minimizing the walking distances resulting from the choice of lines

of work. To do this, we use the following decision variables:

$$x_\omega = \begin{cases} 1 & \text{if the line of work } \omega \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

We use the objective of minimizing the walking distances, leading to the following integer linear programming formulation:

$$\min \sum_{\omega \in \Omega^*} wd_\omega x_\omega = \sum_{\omega \in \Omega^*} \sum_{i \in \mathcal{A}^*} \sum_{j \in \mathcal{A}^*} dist(loc_i^f, loc_j^s) d_{ij}^\omega x_\omega \tag{5.1}$$

subject to

$$\sum_{\omega \in \Omega^*} a_i^\omega x_\omega = 1 \qquad \forall i \in \left\{ j \in \mathcal{A}^* \;\middle|\; rq_k \neq 0 \right\} \tag{5.2}$$

$$\sum_{\omega \in \Omega^*} a_i^\omega x_\omega = 0 \qquad \forall i \in \left\{ j \in \mathcal{A}^* \;\middle|\; rq_k = 0 \right\} \tag{5.3}$$

$$\sum_{\omega \in \Omega^*} b_k^\omega x_\omega = 1 \qquad \forall k \in \mathcal{R} \tag{5.4}$$

$$\sum_{\omega \in \Omega^*} \left(c_j^\omega x_\omega - c_i^\omega x_\omega \right) \geq p_j \qquad \forall i \prec j \in \mathcal{P} \tag{5.5}$$

$$x_\omega \in \{ 0, 1 \} \qquad \forall \omega \in \Omega^* \tag{5.6}$$

Here, Constraints 5.2 ensures that each activity that needs a member of staff is assigned to exactly one member of staff. Constraints 5.3 ensures that activities that do not need staff members are not assigned. Constraints 5.4 ensures that members of staff are given at most one line of work. Constraints 5.5 ensures that tasks with a precedence relation are executed in the correct order, and that processing times are respected. Constraints 5.6 specifies the domain of the variables.

5.1.2 LP relaxation and dummy variables

We start the process without any meaningful lines of work in Ω^*, only adding the trivial line of work $[st_k, br_k, fi_k]$ for all members of staff $k \in \mathcal{R}$. In order to ensure that we can always find a feasible solution for the master problem, we add a dummy variable for each activity: these variables can then all be set to one to ensure a feasible solution. Setting these variables to a non-zero value is penalized in the objective function. These variables are

$$U_i = \begin{cases} 1 & \text{if activity } i \in \mathcal{A}^* \text{ is not assigned} \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, since we use a column generation approach, we need the LP-relaxation of the master problem. The objective of the LP-relaxation with the dummy variables is to

$$\min \sum_{\omega \in \Omega^*} \sum_{i \in \mathcal{A}^*} \sum_{j \in \mathcal{A}^*} dist(loc_i^f, loc_j^s) d_{ij}^{\omega} x_{\omega} + \sum_{i \in \mathcal{A}^*} LU_i \tag{5.7}$$

subject to

$$\sum_{\omega \in \Omega^*} a_i^{\omega} x_{\omega} + U_i = 1 \qquad \forall i \in \left\{ j \in \mathcal{A}^* \;\middle|\; rq_k \neq 0 \right\} \tag{5.8}$$

$$\sum_{\omega \in \Omega^*} a_i^{\omega} x_{\omega} + U_i = 0 \qquad \forall i \in \left\{ j \in \mathcal{A}^* \;\middle|\; rq_k = 0 \right\} \tag{5.9}$$

$$\sum_{\omega \in \Omega^*} b_k^{\omega} x_{\omega} = 1 \qquad \forall k \in \mathcal{R} \tag{5.10}$$

$$\sum_{\omega \in \Omega^*} \left(c_j^{\omega} x_{\omega} - c_i^{\omega} x_{\omega} \right) + LU_j \geq p_j \qquad \forall i \prec j \in \mathcal{P} \tag{5.11}$$

$$x_{\omega} \geq 0 \qquad \forall \omega \in \Omega^* \tag{5.12}$$

Terms for the dummy variables are added to the objective, where selecting a dummy variable results in a penalty. Here, L is a sufficiently large number.

5.2 Pricing problem

The pricing problem is solved every iteration of the column generation method to obtain new columns. Only columns that have a possibility to yield an improvement are selected. Whether a new column will yield an improvement is determined by calculating the reduced cost of the new column.

5.2.1 Reduced cost

Adding a new column ω with variable value $x_{\omega} = \epsilon$ results in the following changes in the objective function:

$$\epsilon \left[\sum_{i \in \mathcal{A}^*} \sum_{j \in \mathcal{A}^*} dist(loc_i^f, loc_j^s) d_{ij}^{\omega} \right]$$

The effect on the constraints is as follows, with π_i the shadow price for the constraint in Constraints 5.8 and Constraints 5.9 for activity i, and λ_k the shadow price for the constraint in Constraints 5.10 for member of staff k.

$$\text{Constraints 5.8 and 5.9: } -\epsilon \left[\sum_{i \in \mathcal{A}^*} a_i^{\omega} \pi_i \right]$$

$$\text{Constraints 5.10: } -\epsilon \left[\sum_{k \in \mathcal{R}} b_k^{\omega} \lambda_k \right]$$

$$\text{Constraints 5.11: } -\epsilon\left[\sum_{i<j\in\mathcal{P}}\delta_{ij}\left(c_j^\omega - c_i^\omega\right)\right] = -\epsilon\left[\sum_{i\in\mathcal{A}^*}\left[\sum_{j\in pre_i}\delta_{ij}c_i^\omega - \sum_{j\in suc_i}\delta_{ji}c_i^\omega\right]\right]$$
$$= -\epsilon\left[\sum_{i\in\mathcal{A}^*}c_i^\omega\left[\sum_{j\in pre_i}\delta_{ij} - \sum_{j\in suc_i}\delta_{ji}\right]\right]$$

Together, this gives the following reduced cost per unit:

$$\sum_{i\in\mathcal{A}^*}\sum_{j\in\mathcal{A}^*} dist(loc_i^f, loc_j^s)d_{ij}^\omega - \sum_{i\in\mathcal{A}^*}a_i^\omega\pi_i - \sum_{k\in\mathcal{R}}b_k^\omega\lambda_k - \sum_{i\in\mathcal{A}^*}c_i^\omega\left[\sum_{j\in pre_i}\delta_{ij} - \sum_{j\in suc_i}\delta_{ji}\right] \tag{5.13}$$

5.2.2 Complete formulation

Since a line of work is meant for exactly one member of staff, say $k \in \mathcal{R}$, we have $b_k = 1$ and $b_{k'} = 0$ for all $k' \neq k$. This means that the term $\sum_{k\in\mathcal{R}} b_k^\omega\lambda_k$ is constant, and can be removed from the summation. Let $k \in \mathcal{R}$ be the member of staff for which the column ω is meant, i.e. $b_k^\omega = 1$. The objective of the pricing problem is to

$$\min \sum_{i\in\mathcal{A}^*}\left[\sum_{j\in\mathcal{A}^*} dist(loc_i^f, loc_j^s)d_{ij}^\omega - a_i^\omega\pi_i\right] \tag{5.14}$$

subject to the constraints

1. Activity $i \in \omega$ is started after the release date and finished before the deadline: $r_i + p_i \leq c_i^\omega \leq d_i$.
2. ω has the structure $[st_k = i_0, i_1, \ldots, br_k, \ldots, i_m, i_{m+1} = fi_k]$ as given in Definition 2.13.
3. ω contains only activities i where $rq_i = rsk_k$.
4. The precedence constraints are satisfied: if $i \prec j \in \mathcal{P}$, and $i, j \in \omega$, then we require that $c_j^\omega \geq c_i^\omega + p_j$.
5. If d_{ji}^ω, then $c_i^\omega \geq c_j^\omega + dist(loc_j^f, loc_i^s) + p_i$.
6. The completion times are positive: $c_i^\omega \geq 0$, for all $i \in \mathcal{A}^*$.

6 EXPERIMENTAL SETUP

In this chapter, we give information on the case study we performed for NS. In Section 6.1, we give information on the case and the location. We show the scenarios used for testing, and the preprocessing steps in Section 6.2.

6.1 Kleine Binckhorst

We have examined the performance of the algorithm using instances for the facility Kleine Binckhorst, near The Hague Central Station. In this section, we will give some details on the location and on the tasks that are performed there.

6.1.1 Location

A schematic map of the location is shown in Figure 6.2. This map is a simplified view. The black thin lines are the tracks. The H indicates the location of the depot, where the members of staff start and end their shifts and where the breaks happen, i.e. location $0 \in \mathcal{L}$.

This location has eight tracks that can be used for inspections and repairs, and for parking. These are the tracks 52–59. Track 64 has extra facilities for special types of repairs. The tracks 61 and 62 have a special platform to facilitate cleaning the inside of the train. Track 63 contains the installation for exterior cleaning. Track 60 can only be used for parking.

The positions of the track numbers in the map are the coordinates used when computing the walking distances. If the track number is in a hexagon, this means that the track can be used by a member of staff from both sides. If the number is in a trapezium, the track can only be used from the slanted side, i.e. track 53 can be used from both sides, while track 62 can only be used from the south side. This is because tracks 61 and 62 require the member of staff to enter from the special platform, which is located inbetween these two tracks.

Track 906a is a special track, as it is the location where trains arrive and leave the facility from the national railway network. In the context of staffing, the national railway network is part of the Centraal Bediend Gebied (CBG), while the facility is part of Niet Centraal Bediend Gebied (NCBG). For Kleine Binckhorst, this means that tracks 51–64 are part of the NCBG and track 906a is part of the CBG. The boundary between CBG and NCBG has consequences for the types of staff that are able to make specific trips. This boundary is shown in Figure 6.1.

Because of safety regulations, members of staff are only permitted to cross the tracks at designated paths. This means that a walk between two parallel tracks may take much longer than expected.

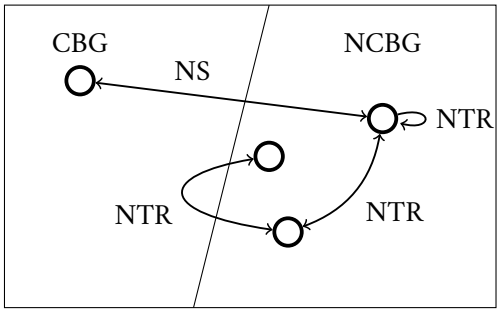


FIGURE 6.1: Diagram to illustrate the difference between Centraal Bediend Gebied and Niet Centraal Bediend Gebied. The arrows indicate trips between the two areas. NS and NTR indicate which type of engineer needed for each trip, see Section 6.1.2.

These paths are shown as the thicker red lines.

The numbers in the rectangles indicate the distance along the track. Note that this is a simplified diagram and that the distances are only valid for the track number indicators (i.e. the coordinates of the tracks) and for the paths. Furthermore, we assume that all paths are straight*. The distances between the tracks vary per track, though they are all in the range of 4 m to 6 m. In our model, we assume that the distances between each track are exactly 5 m. We also assume that the distances between parallel paths follows the same pattern, e.g. that the distance between the path between track 58 and 59 and the path between track 57 and 58 is also exactly 5 m. The complete distance matrix used in our model is shown in Table A.1. While the facility is used throughout the day, we only consider the planning period from 17 h 30 min to 7 h 40 min the following day. This is because of the high availability of testing scenarios for this time interval, as this time interval has been used in experimentation by NS in the past. Furthermore, this time interval is the busiest of the day, since almost no EMUs are in service during the night.

6.1.2 Staff

In our model, we model three disjoint types of staff:

Engineers Engineers are responsible for driving the EMUs across the facility. Note that there are two types of engineers: NS engineers and NTR engineers. NTR engineers are responsible for driving trips that start and end within the NCBG, while NS engineers are responsible for driving trips that have at least one end in the CBG, as shown in Figure 6.1. The only tasks that travel from CBG to NCBG are the tasks where EMUs arrive or depart. In our model, we do not consider the NS engineers, since the engineers generally leave the facility once the arrive task is complete, and enter the facility when the depart task will take place.

Technicians A maintenance technician is responsible for performing the inspections, as well as small repair jobs. In our model, we assume that all technicians can perform all tasks on all types of EMUs.

* The path from track 59 to track 906a is not used in our model.

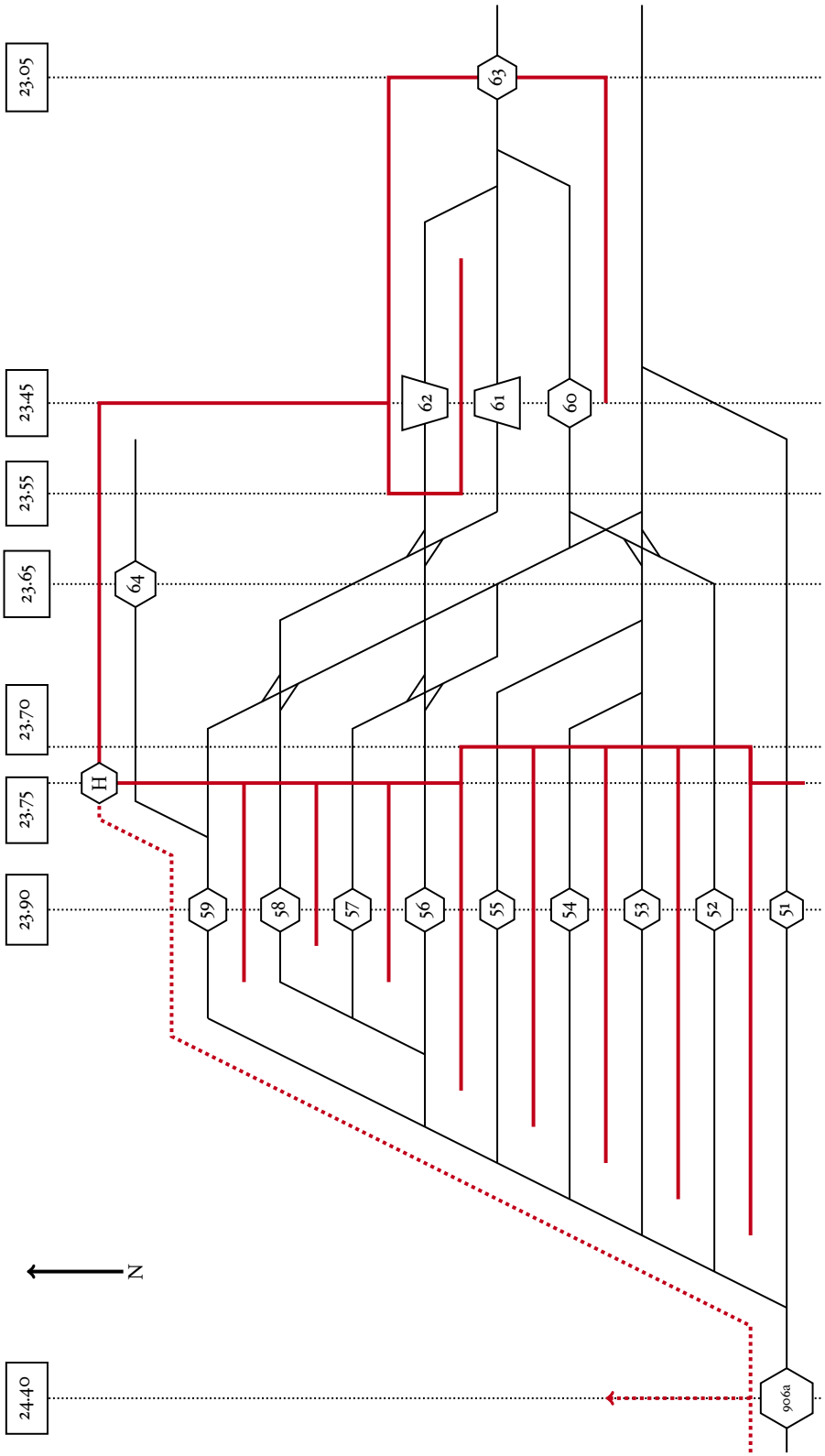


FIGURE 6.2: Simplified schematic diagram of Kleine Binckhorst used in our model. Black lines are tracks, red lines are walking paths. The values in the hexagons and trapezia are the track numbers, the numbers in the rectangles are the position in km along the track. The diagram is not drawn to scale.

Name	Cleaners			Engineers			Technicians		
	Early	Middle	Late	Early	Middle	Late	Early	Middle	Late
2|2|2	1	0	1	1	0	1	1	0	1
2|3|2	1	0	1	1	1	1	1	0	1
2|4m|2	1	0	1	1	2	1	1	0	1
2|4e1|2	1	0	1	2	0	2	1	0	1
3|3|3	1	1	1	1	1	1	1	1	1
3|4m|3	1	1	1	1	2	1	1	1	1
3|4e1|3	1	1	1	2	0	2	1	1	1

TABLE 6.1: Number of staff used of each skill and shift per configuration.

Cleaning team A cleaning team is responsible for cleaning interiors, and consists of several cleaners. As the time it takes the team to clean a single EMU is dependent on the number of team members, we assume that all cleaning teams contain the same number of team members.

We assume that all members of staff have a constant walking speed of 4 km/h, which is the norm used by NS.

The members of staff work in shifts of six or eight hours. We only consider the eight hour shifts. Specifically, we use the shifts *Early* from 17 h 30 min to 1 h 30 min, *Middle* from 20 h 35 min to 4 h 35 min, and *Late* from 23 h 40 min to 7 h 40 min. We chose these times because they spread evenly over the planning period.

Breaks take 30 minutes. The window of time in which a break for member of staff k can occur is set at $[(rs_k + rf_k)/2 - 45; (rs_k + rf_k)/2 + 45]$. With the shifts given before, the windows are 20 h 45 min to 22 h 15 min, 23 h 50 min to 1 h 20 min, and 2 h 55 min to 4 h 25 min respectively.

We use several configurations in our testing. The numbers of staff available of skill and shift per configurations are shown in Table 6.1.

6.1.3 Types of tasks

The tasks performed at Kleine Binckhorst can be divided into several categories:

Arrive and Exit These tasks represent the arrival or departure of one or more EMUs at the facility. Arrive tasks always have track 906a as the finishing location. Exit tasks always have track 906a as the starting location. We assume that this is done by an NS engineer, and that it does not require any of the staff modeled. We also assume that the subsequent Move operation, which moves the EMU from/to track 906a from/to a location on the facility is done by the NS engineer. These actions must be performed at exact times, as such, the activities have a release date that is equal to the deadline, ensuring that there is no possibility of executing these tasks at a different time. All other tasks types in this list do not need release dates and deadlines, since they are implied through the precedence constraints.

Move These tasks represent the shunting of one or more EMUs within the facility. These require an *engineer*.

Wait These tasks represent the periods during which the EMU is parked on a track in the facility. These require no staff.

Split and Combine These tasks represent the splitting or combining of train pairs. These require an *engineer*.

Inspections These tasks represent the inspections that are performed regularly on the EMUs. A *maintenance technician* is required for this task.

Interior cleaning These tasks represent the daily cleaning that is performed on the inside of the EMU. This task requires a *cleaning team*.

Exterior cleaning These tasks represent the cleaning of the exterior of the EMU by driving through a washing machine. This task requires an *engineer*.

6.2 Test scenarios

To obtain test scenarios, we will use the plans generated by the algorithm for routing the trains and scheduling the tasks by Van den Broek [5]. These plans are based on scenarios that are computer generated, with the use of real world parameters. Each scenario has been generated to schedule a different number of EMUs in the planning period.

A solution contains a set of tasks $\mathcal{T}$. For each task $t \in \mathcal{T}$, we know

- The starting and finishing time of the task in the plan, st_t and fi_t respectively;
- The starting and finishing location of the task;
- The resources (such as tracks, signals, switches) required by the task. This is important in the case of a "Move" activity, as these need exclusivity on the entire route.

We transform a plan into a precedence graph, such that we can use it as an instance for our problem. This is done as follows. In this description, λ is five minutes.

- Add all the tasks as activities to the problem instance. Each task $t \in \mathcal{T}$ is converted into an activity i and gets a minimum duration according to the duration it has in the plan, i.e. $p_i = fi_t - st_t$, except for the Wait activities, which get a minimum duration of $p_i = 0$. This is done because the waiting is simply a consequence of the scheduling of the other tasks, i.e. trains do not have to use the complete waiting time. We also set the original starting and finishing times, i.e. $s_i^{orig} = st_t$ and $c_i^{orig} = fi_t$. The start and end locations are also set accordingly.
- The Arrive and Exit tasks get their starting time as release date and finishing time as deadline. Since these tasks are partly external, it is important that they are executed at the specified time. These are the only tasks with release dates and deadlines, besides the dummy activities representing the start and end of member of staff shifts. Other tasks do not require release dates and deadlines, since they are implied through the precedence constraints and the release dates and deadlines on the Arrive and Exit tasks.

- For each train, select all the tasks that concern this train, either by itself or as part of a larger shunting unit. Order these by start time and add precedence constraints between each task. These are then added to $\mathcal{P}$.

- For each resource (tracks, switches, etc.), select all the tasks that use the resource. Order these by start time. Since some resources, such as tracks, may have a capacity greater than one, we partition the tasks into subsets, such that each subset represents a schedule for that particular "unit" of the resource. For each subset, ordered by start time, we add a precedence constraint to $\mathcal{P}$. This satisfies Assumption 2.4.

Note that Assumption 2.1 is now satisfied, as the precedence constraints are obtained from a solution that is known to be feasible.

The database we used to obtain instances contained over 20000 solutions. Out of these solutions, we picked twenty. The IDs of the selected solutions are shown in Table A.2, together with the number of EMUs contained in the solved scenarios, and the number of tasks and movements. This does not include parking tasks. The number of EMUs ranges from 11 to 21. The total number of tasks and movements per scenario ranges from 91 to 176. Furthermore, Table A.2 shows the number of activities and precedence constraints after importing each scenario. We also give the average flexibility per activity in the scenario without any lines of work, which we define as the concurrent flexibility divided by the number of activities. The statistics after preprocessing, as well as the number of EMUs that enter and leave the facility in each scenario are shown in Table A.5. The number of activities per skill after preprocessing in each scenario is shown in Table 6.2.

6.2.1 Preprocessing

Removing wait activities As we have said in Section 6.1.3, the Wait tasks do not need any member of staff to be executed. As such, we do not actually need to plan them. Furthermore, in the plans we import, the Wait tasks always have one incoming and one outgoing precedence constraint. These are Move tasks, to bring the train to the parking location. Because we do not need to schedule these tasks, we can remove them from the scenario.

This means that we remove each Wait task i, and the Move activities given by the precedence constraints $j \prec i$ and $i \prec j'$. We then add a precedence constraint $j \prec j'$, since the Move activities still need to be performed in the given order. The result of this operation is a reduction of around 33 % in the number of activities, and around 23 % in the number of precedence constraints. Detailed results are shown in Table A.3.

Removing Arrive/exit activities As said in Section 6.1.3, an Arrive task i places the train at track 906a. However, the train is always moved to a location in the facility by the same NS engineer performing the Arrive task. The plan contains this as a Move task j with $i \prec j$.

Since i and j are performed by an outside engineer, we do not need to schedule any of these two tasks. As such, we remove them from the plan, and combine them into a new Arrive task j', with $r_{j'} = r_i$, $d_{j'} = d_i + p_j$,$p_{j'} = p_j$, and $rq_{j'} = 0$. In order to keep existing precedence constraints intact, we say $pre_{j'} = pre_i \cup pre_j$ and $suc_{j'} = suc_i \cup suc_j$.

Id	# EMUs	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{A}_{cleaners}\rvert$	$\lvert\mathcal{A}_{engineers}\rvert$	$\lvert\mathcal{A}_{technicians}\rvert$	$\lvert\mathcal{A}_0\rvert$	$\mathcal{P}$
93240	11	84	11	51	8	14	107
93127	14	79	14	36	12	17	105
94882	14	88	14	46	11	17	121
93366	13	91	13	52	9	17	129
95171	14	98	14	57	10	17	130
112085	14	100	14	58	11	17	140
112487	15	107	15	60	12	20	147
113639	14	108	14	67	10	17	145
102809	15	117	15	70	12	20	163
102861	15	120	15	73	12	20	179
102816	15	128	15	83	10	20	181
90321	16	131	16	79	15	21	182
110562	15	129	15	81	13	20	173
102058	15	139	15	93	11	20	189
93384	15	144	15	97	12	20	204
93960	15	152	15	105	12	20	213
100322	15	151	15	105	11	20	206
111658	20	155	20	91	18	26	216
101062	15	162	15	116	11	20	224
93215	21	164	21	102	14	27	237

TABLE 6.2: Statistics on each scenario after the Wait, Arrive, and Exit tasks are removed, and the transitive reduction algorithm has been applied, with the activity counts split per skill.

Applying this operation results in a decrease in the number of activities of around 14 % when compared to just removing the Wait activities. The reduction in the number of precedence constraints is around 8 %. The detailed results of this operation combined with removing the Wait activities is shown in Table A.4.

Removing redundant constraints In the process, the precedence graph may contain many redundant precedence constraints. In order to satisfy Assumption 2.2, we compute the transitive reduction of the precedence graph.

DEFINITION 6.1. Let $G' = (V, A')$ be the transitive reduction of a weighted graph $G = (V, A)$. G' satisfies the following properties:

1. There exists a path from u to v in $G \iff$ there exists a path from u to v in G', $\forall u, v \in V$.
2. A' is a minimal subset of A such that the previous property holds.

To find the transitive reduction, we use the algorithm by Hsu [20], outlined in Algorithm 6.1. This algorithm starts by finding all the pairs of nodes between which a path in G exists. This is done using the algorithm by Floyd [14] and Warshall [36], which runs in $O(n^3)$ time. If the Floyd-Warshall algorithm does not find a path between two vertices, the length is set to ∞. The path matrix will then be used as the initial adjacency matrix for G'. The algorithm then iterates over the nodes,

continuously removing arcs (i, k) from G' if there are still arcs (i, j) and (j, k) in G'. This algorithm also runs in $O(n^3)$ time.

ALGORITHM 6.1: Algorithm by Hsu [20] for finding a transitive reduction of a graph

Input: Graph $G = (V, A)$
Output: Graph $G' = (V, A')$, a transitive reduction of G, where A' is the adjacency matrix.

```
D ← output from Floyd-Warshall(G_w = (V, A, 0.0))   ▷ Floyd [14] and Warshall [36]
A' ← adjacency matrix for G'
for i ∈ V:
    for j ∈ V:
        if D[i, j] ≠ ∞:
            A'[i, j] ← true
for i ∈ V:
    for j ∈ V:
        if A'[i, j] = true:
            for k ∈ V:
                if A'[j, k] = true:
                    A'[i, k] ← false
return G' ← (V, A')
```

Applying this operation does not reduce the activity count. The number of precedence constraint is reduced by around 45 % when compared to the scenarios with the Wait, Arrive, and Exit tasks removed. The detailed results of this operation on the testing scenarios combined with the removal of the Wait, Arrive, and Exit activities is shown in Table A.5.

6.2.2 Complexity

In the testing scenarios, there are two types of precedence constraints in $\mathcal{P}$: constraints that arise from the ordering of activities from the shunting unit perspective, and constraints that arise from the ordering of activities from the resource perspective. Most activities have only one predecessor through the shunting unit constraints, except for the Combine activities, which have two predecessors. Furthermore, each activity only has one predecessor from the resource perspective, seeing as we have ordered all activities per resource with precedence constraints. This means that the number of precedence constraints is linear in the number of activities, and as such, the constraint graph is sparse.

7 RESULTS

We have tested the algorithm in several configurations: using just the greedy heuristic from Section 4.2, and using the greedy heuristic and the local search improvement algorithm from Section 4.3. We have also tested the algorithm using each component of the objective function shown in Section 4.1. individually. The parameters used are shown in Table 7.1. We have tested each scenario in Table 6.2 with each staff combination in Table 6.1, resulting in a set of 140 problem instances. Each problem instance and configuration combination has been tested eight times, with upto four runs executing concurrently, resulting in a total of 1120 runs per configuration. In Section 7.1, the ability of the algorithm to find solutions is evaluated. Section 7.2 contains information on the quality of the solutions found. Finally, Section 7.3 contains the results of other tests, to indicate the performance of the algorithm.

7.1 Ability to find solutions

In this section, we try to measure the ability of the algorithm to find solutions. This is done by checking the performance of the greedy heuristic by itself, and the performance of the greedy heuristic followed by the local search improvement algorithm. Both are checked using several objective functions. Furthermore, we have put each instance through a MIP solver, to be able to make statements about the existence of feasible solutions. Table A.6 shows the results found by the MIP solver for each instance. We have set a time limit of 10 min. The MIP solver used is Gurobi by Gurobi Optimization, Inc. [18]. Note that if the MIP solver was unable to find a feasible solution within the time limit, it does not mean that such a solution does not exist.

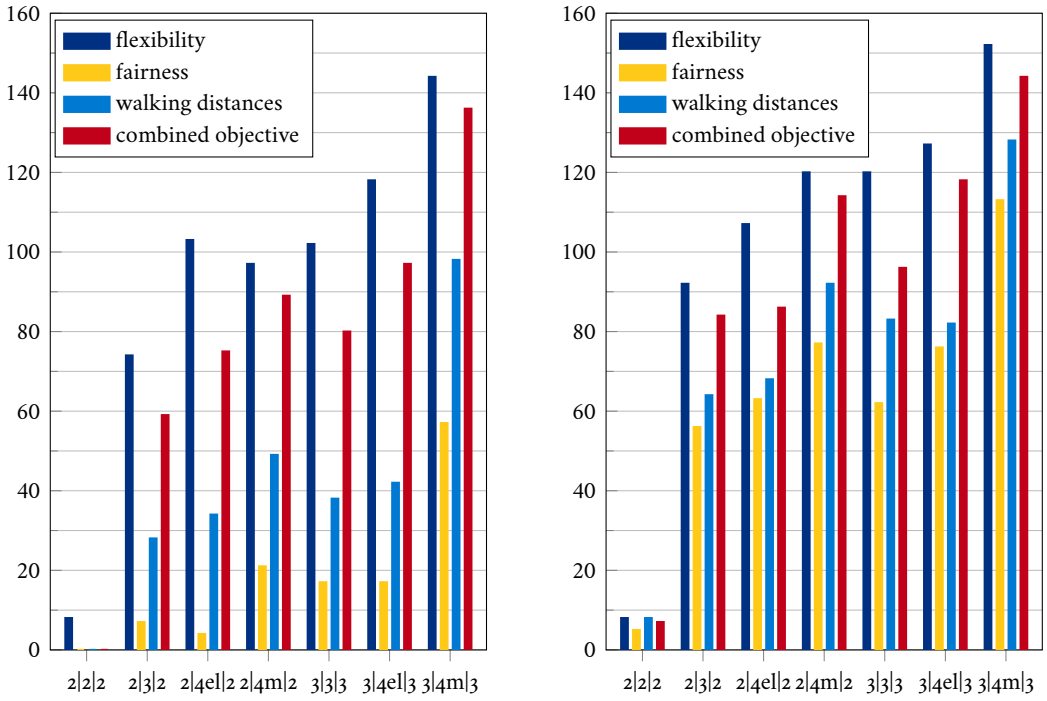
7.1.1 Only greedy heuristic

In Figure 7.1, the number of solutions found is shown per staff combination. For each staff combination, the total number of solutions found is shown, across all scenarios. This means that the maximum is 160. The plots contain bars for each version of the objective function tested: one with only the flexibility component, one with only the walking distance component, one with only the fairness component, and one with the components combined using the weights shown in Table 7.1. The unscheduled activities component of the objective function is only used in the local search algorithm, and is used in all versions tested, using the weight shown in Table 7.1. It is not used in the greedy heuristic.

In Figure 7.1a, the results found when only using the greedy heuristic are shown. The detailed

Notation	Description	Section	Value
$w_{flexibility}$	Weight of the flexibility in the combined objective	4.1	1
$w_{fairness}$	Weight of the fairness in the combined objective	4.1	−5
$w_{walking}$	Weight of the walking distances in the combined objective	4.1	−0.5
$w_{unscheduled}$	Weight of the unscheduled activities in the combined objective	4.1	−1000
B	Number of activities removed each round in greedy algorithm	4.2	25
R	Number of rounds in greedy algorithm	4.2	5
$T_{unscheduled}$	Weight of unscheduled activities in the temperature	4.3	1
T_{init}	Initial temperature in simulated annealing algorithm	4.3	50
t_c	Temperature update interval in simulated annealing algorithm	4.3	1000
β_c	Temperature update factor in simulated annealing algorithm	4.3	0.98
$it_{\max}$	Maximum allowed number of iterations without improvement	4.3	25 000

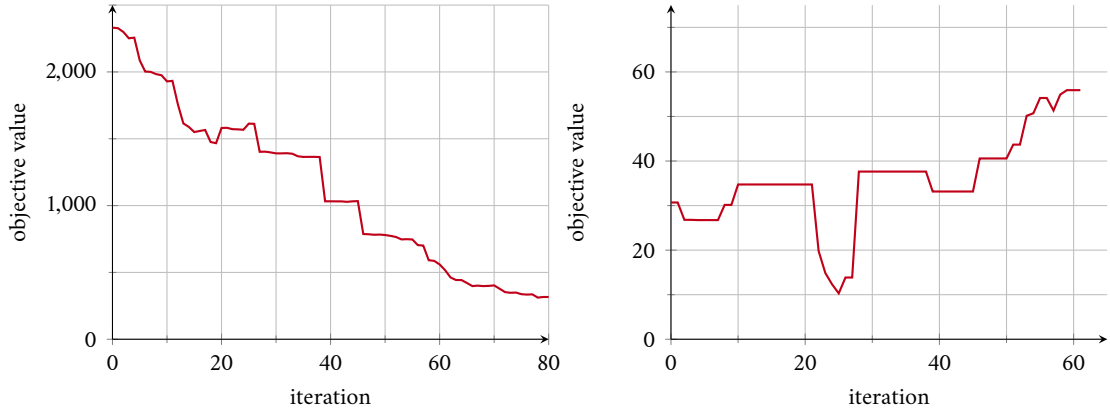
TABLE 7.1: Parameter values used in the experimentation.



(A) Only greedy heuristic.

(B) Greedy heuristic and local search improvement.

FIGURE 7.1: Number of solutions found (out of a possible 160), using different objective functions.



(A) Concurrent flexibility (higher is better)

(B) Fairness (lower is better)

FIGURE 7.2: Progression of the objective function during the greedy heuristic step, using different objective functions. Only one round is shown.

results, giving the numbers per scenario are shown in Tables A.7 to A.10. From these results, it becomes apparent that simply using the flexibility as the objective gives the best results. The combined objective shows a slightly weaker performance. Using just the fairness component of using just the walking distance component both give results that are much worse. We expect that this is due to the mostly decreasing nature of the flexibility: adding an activity to a line of work almost always results in a decrease of the flexibility. An example of this is shown in Figure 7.2a. The walking distance objective does not necessarily increase when adding an activity to a line of work, it can also decrease, e.g. when adding a Move activity. This problem is even more pronounced with the fairness component: adding an activity may increase or decrease the fairness, depending on the contents of the lines of work. Furthermore, since we use the maximum of the standard deviations per skill, the objective may often not change at all. An example is shown in Figure 7.2b. Because of this, the greedy method is often unable to find a solution when using the fairness objective.

When compared to the results found by the MIP solver, see Table A.6, the greedy heuristic optimizing the flexibility or the combined objective performs better with larger instances. Starting from scenario 112487, the MIP solver is unable to find solutions for several staff configurations, while the heuristics do not have that problem. However, the greedy heuristics do not perform as well on scenario 93366, where the MIP solver found optimal solutions for each staff configuration except 2|2|2, while the greedy heuristics are unable to find a solution every time.

7.1.2 Greedy heuristic combined with local search improvement

In Figure 7.1b, the results are shown that were found when using the greedy heuristic to obtain an initial solution, together with the local search improvement. The detailed results are shown in Tables A.11 to A.14. Like with the previous results, using only the flexibility component yields the most solutions, followed by using the combined objective. Like before, the walking distance and fairness components do not perform as well. This is likely due to the same reasons shown earlier. Of course, the local search method combined with the greedy heuristic is able to find more solutions

than the method using only the greedy heuristic. Like the greedy heuristic, it performs much better than the MIP solver on large instances.

The difference in ability to find feasible solutions between the objective functions may be due to a lack of soft constraints in our model. In the local search improvement step, we have a single soft constraints, seeing as solutions are also accepted if there are activities that are unscheduled, however, this is not the case with the greedy heuristic. Furthermore, we do not allow any deadline violations, or any situation that would result in an inconsistent STN. If we would have incorporated more soft constraints, both methods might have been able to better explore the search space.

In Figures 7.1a and 7.1b, we can see that the algorithm finds the most solutions for the staff combinations 3|4m|3 and 3|4el|3. Of course, this makes sense, since these combinations contain the largest number of members of staff, meaning that it is easier to divide the tasks over the members of staff. Furthermore, the combination 3|4m|3 outperforms the combination 3|4el|3. This is likely because the number of jobs during the Middle shift is larger than the number of jobs during the Early or Late shifts. The Early shifts generally contain the Arrive tasks, and as such, not all EMUs are on the facility during the Early shift, resulting in a smaller number of jobs for the members of staff. The same holds for the Late shifts, which contain the Exit tasks. As such, having more engineers during the Middle shifts gives better results, when compared to having more engineers during the Early and Late shifts.

Tables A.7 to A.14 show that the greedy method was able to find solutions for all scenarios using the staff combinations from Table 6.1, except for 113639 and 93960. The greedy method combined with the local search improvement step is able to find solutions for all scenarios except for 113639. Both methods were unable to reliably find solutions for scenario 101062, except when using the flexibility objective.

We have used the output of the matching algorithm (see Section 4.5) to find a staff configuration for which the algorithm was able to solve scenario 113639. Starting from staff configuration 2|2|2, and adding new staff based on the output of the matching algorithm, we obtained a staff set 2|5m|3e, which contains two cleaners, (one early, and one late), five engineers, (one early, three middle, one late), and three technicians, (two early, one middle). Using this staff configuration, the algorithm is consistently able to find solutions. The main issue with the staff combinations used in our experimentation is the placement of the technicians: we only consider configurations with one technician in the early shift, one in the late shift, and in some configurations one in the middle shift. As it turns out, in scenario 113639, almost all of the technician activities are scheduled early in the planning period. As such, no technician is needed in the late shift.

7.2 Quality of the solutions

In this section, we measure the quality of the solutions found by the algorithm. Only runs where a feasible solution was found have been used in the calculations in this section.

7.2.1 Components of the objective function

Figure 7.3a shows the average flexibility of the solutions found per staff configuration, for each version of the objective function. It is apparent that using only the flexibility component as the objective function yields the most flexible solutions. The combined objective gives slightly worse

results, however, the flexibilities are still relatively high. Much worse results are obtained when optimizing the walking distance component or the fairness component. The average flexibilities found per scenario and technician set are shown in Table A.15. The values shown are the flexibilities divided by the number of activities in the instance, i.e. $flex/(|\mathcal{A}| + 3\,|\mathcal{R}|)$.

In Figure 7.3b, the fairness of the solutions is shown, which is measured as the maximum of the standard deviation of the workloads per skill. Of course, in most cases optimizing using only the fairness gives the best result, with the combined objective closely following. However, in the case of 2|4m|2, optimizing the combined objective yields solutions with a better fairness than simply optimizing the fairness. This is likely due to the fact that optimizing the combined objective gave more solutions than optimizing the fairness, see Figure 7.1b. With more solutions, the chances of finding solutions with a better fairness are also higher, thus lowering the average. Optimizing the flexibility provides worse results, and optimizing the walking distances provides the worst results.

The walking distances in the solutions found are shown in Figure 7.3c. The solutions with the least walking were found by optimizing only the walking distances, as expected. Optimizing the combined objective finds solutions with larger walking distances. Even larger walking distances are found when using the flexibility. The largest walking distances were found when optimizing the fairness. The walking distances are still far away from the optimum. For example, the average total walking distances across all solutions found for scenario 93127 by the greedy heuristic with local search improvement, optimizing the walking distances is 254.7, where the average of the optimal solutions is 160.5.

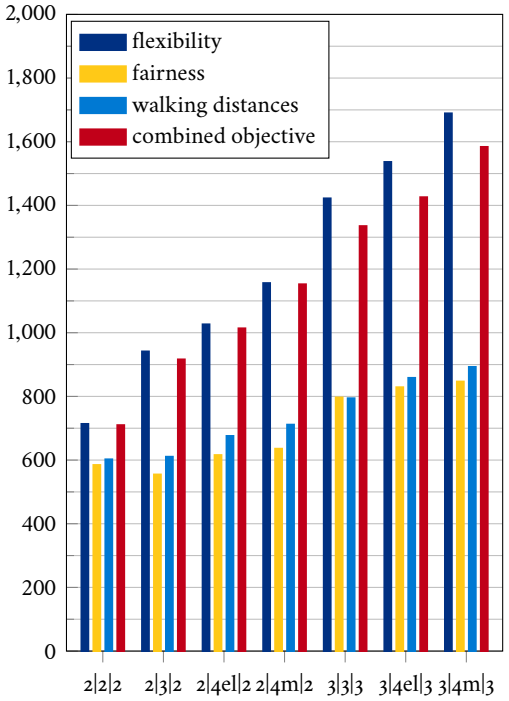
7.2.2 Extending activity durations

We have also measured the robustness of solutions by checking whether a solution is still feasible if the runtimes are extended. For example, runtimes can be extended because of disruptions, or because of slowness. For each solution, we calculate the maximum factor with which the runtimes can be multiplied while still keeping the STN consistent. A low factor, such as 1.01 indicates that the processing times can only be extended by 1 %, indicating a tight schedule.

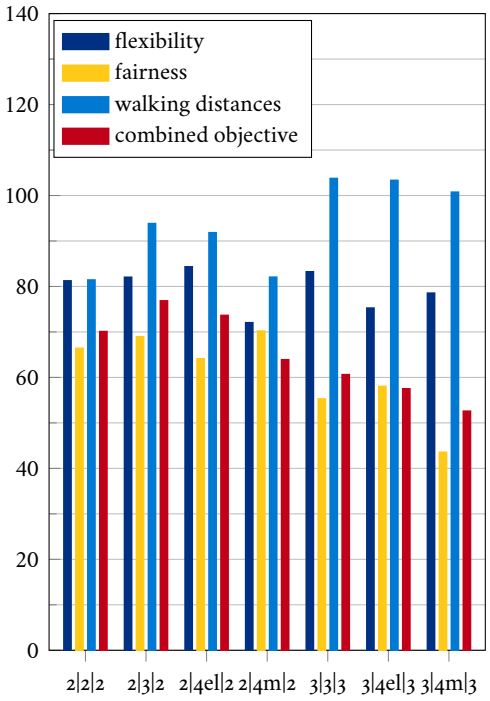
This is done using a method similar to binary search: from an initial lower and upper bound of 1.0 and 2.0 respectively, we continually take the average of the bounds, and check if the STN obtained by applying this average to the durations is still consistent. If it is, the lower bound is set to the average. If the STN is inconsistent, the upper bound is set to the average. The process is stopped after ten iterations.

For example, if we have the lower and upper bounds of 1.0 and 2.0 respectively, we get an average of 1.5. Suppose the STN corresponding to the problem instance with each p_i multiplied by 1.5 is consistent. We then set the lower bound to 1.5 to get a new average of 1.75. This is then repeated ten times.

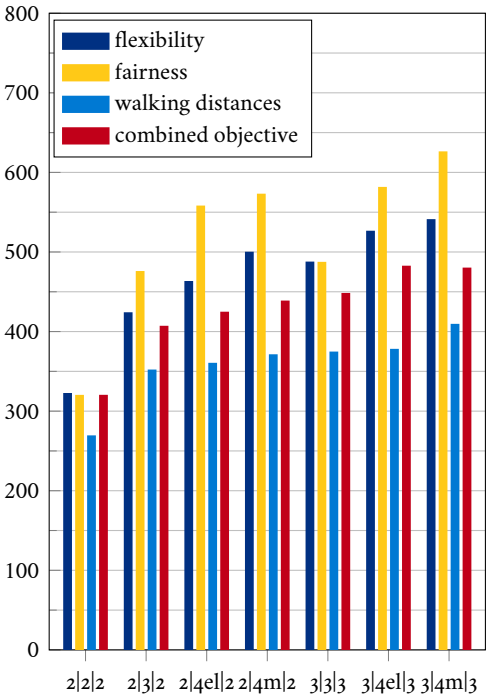
The maximum percentages for the solutions found when using the greedy heuristic and the local search improvement with the combined objective are shown in Table A.16. The values shown are averages over the maximum percentages of each solution found, see Table A.14. The percentages found range from 0.2 % to 21.9 %, with a complete average of 5 %. We can see that the percentages are highest for the technician set 3|4m|3, which is expected, as that is the set with the highest number of technicians. 3|4m|3 also performs better than 3|4el|3, which is also expected, as there are generally more activities during the middle shift.



(A) Concurrent flexibility (higher is better)



(B) Fairness (lower is better)



(C) Walking distances (lower is better)

FIGURE 7.3: Average values of each component of the objective function over all solutions found when using each version of the objective function.

The values found when only extending the processing times of activities requiring a specific skill are higher: when only extending the cleaning tasks, we get an average percentage of 16.3 %, with a maximum of 64.7 %. With the engineer activities, the average is 13.5 %, and the maximum is 69.9 %. With the technician activities, the average is 19.9 %, and the maximum is 75.3 %.

While a low factor indicates a tight schedule, very high factors are not expected. The measure used is very pessimistic, as in reality, it is unlikely that all activities are taking longer than planned. It is more likely that some individual activities are delayed.

There seems to be a slight relation between the flexibility of a solution and the maximum increase in duration. Comparing Table A.15 and Table A.16 shows that sometimes a higher flexibility also means a higher percentage, see for example the column for technician set 3|4m|3. However, in some cases, such as with scenario 93240, the flexibility is rather high, and the percentages are low.

7.3 Performance

In this section, we give some indicators for the performance of the algorithm. We do this by looking at the runtime, memory usage, and the acceptance rate of the neighbourhood operators. The runtime tests were performed on a computer containing an Intel Core i5-4440 quad-core CPU. The method used to check the feasibility of a solution is the algorithm shown in Section 2.4. The flexibility is computed using the algorithm by Mountakis et al. [29] shown in Section 3.3.3, with the optimizations from Sections 3.4.1 and 3.4.2 applied. At most four runs were performed in parallel, to ensure that the performance was not hindered by a lack of threads on the processor. The application has been written in C#, using the .NET Framework, and using Entity Framework for communications with the database. Database connections are only made before and after the call to the algorithm.

7.3.1 Runtime

Table A.17 shows the performance of the algorithm, measured as the number of iterations per second in the local search algorithm. The runs used are the runs from Table A.14. Depending on the scenario, the algorithm is able to process between 350 and 2 000 iterations per second. The number of iterations per second decreases when scenarios with more activities are examined, and when problem instances with more members of staff are examined. Of course, this is as expected. The main bottleneck for the runtime performance is calculating the flexibility. To calculate the flexibility, we first solve an all-pairs shortest path problem on the simple temporal network, and we then find a matching in a complete bipartite graph with two vertices for each variable in the simple temporal network. This amounts to about 70 % of the runtime of the application.

In Table A.18, the average number of iterations needed before termination is shown for each scenario and staff configuration. The runs used are the runs from Table A.14. Across all runs, the algorithm terminated in 75 038 iterations on average. When only considering runs that resulted in a feasible solution, the algorithm needed an average of 57 244 iterations. When only considering infeasible runs, 99 555 iterations were needed. Furthermore, if we only consider runs where a feasible solution was found using the greedy heuristic, only 45 959 iterations were needed. Runs that found a feasible solution through the local search improvement step, where no solution was found using the greedy heuristic, needed 114 414 iterations on average.

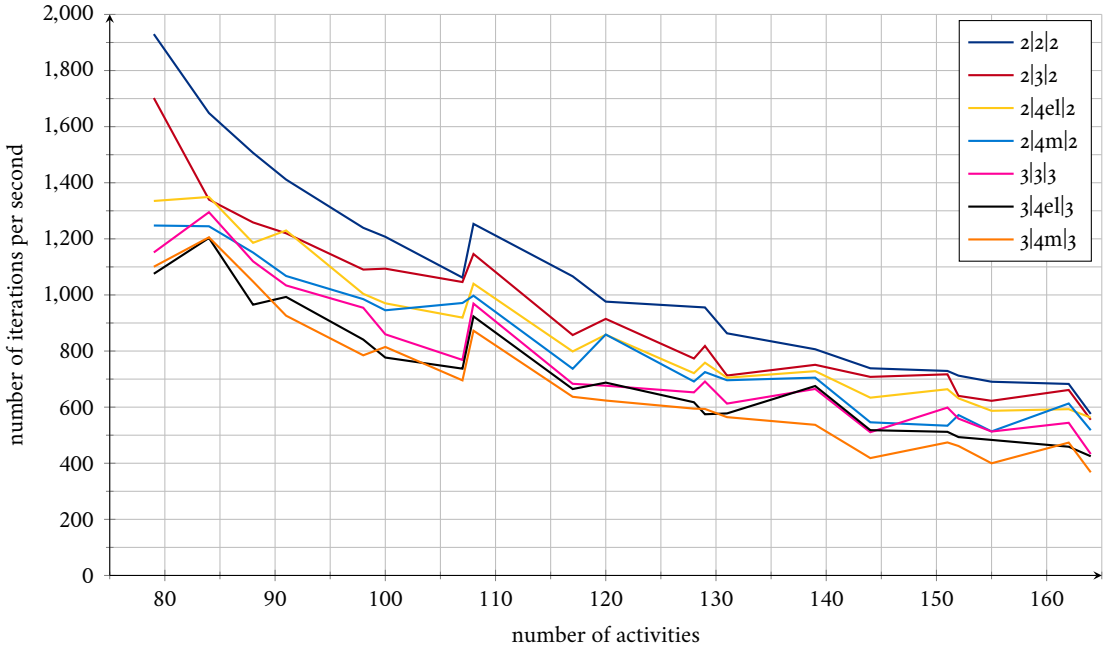


FIGURE 7.4: Number of iterations per second for each staff configuration by number of activities. Each data point is an average over eight runs.

From this we conclude that the algorithm terminates not long after discovering a feasible solution. The local search method is stopped when no improvement has been made in 25 000 iterations. From this, we can conclude that the local search algorithm is still able to improve at the start of the algorithm, though the ability to improve quickly diminishes, causing the stop condition to be met.

7.3.2 Memory usage

As is expected from a local search algorithm, the memory usage of the algorithm is constant during the algorithm. A chart detailing the memory usage during a single run of the algorithm is shown in Figure 7.5. We can see that the memory usage during the algorithm is around 55 MiB when solving scenario 93217 with staff configuration 2|3|2, and around 60 MiB when solving scenario 93215 with staff configuration 3|4m|3.

As can be seen, the memory usage does not change much when solving more difficult instances, with more activities and more members of staff. As such, we believe that the bulk of the memory usage is overhead due to our usage of the .NET Framework. The peaks at the beginning and end are likely caused by database communications using Entity Framework and converting database objects to the objects used in the algorithm.

7.3.3 Operator acceptance rate

To measure the acceptance rate of the neighbourhood operators in the local search algorithm, we use the following two measures. The first, shown in Equation 7.1 shows the average acceptance

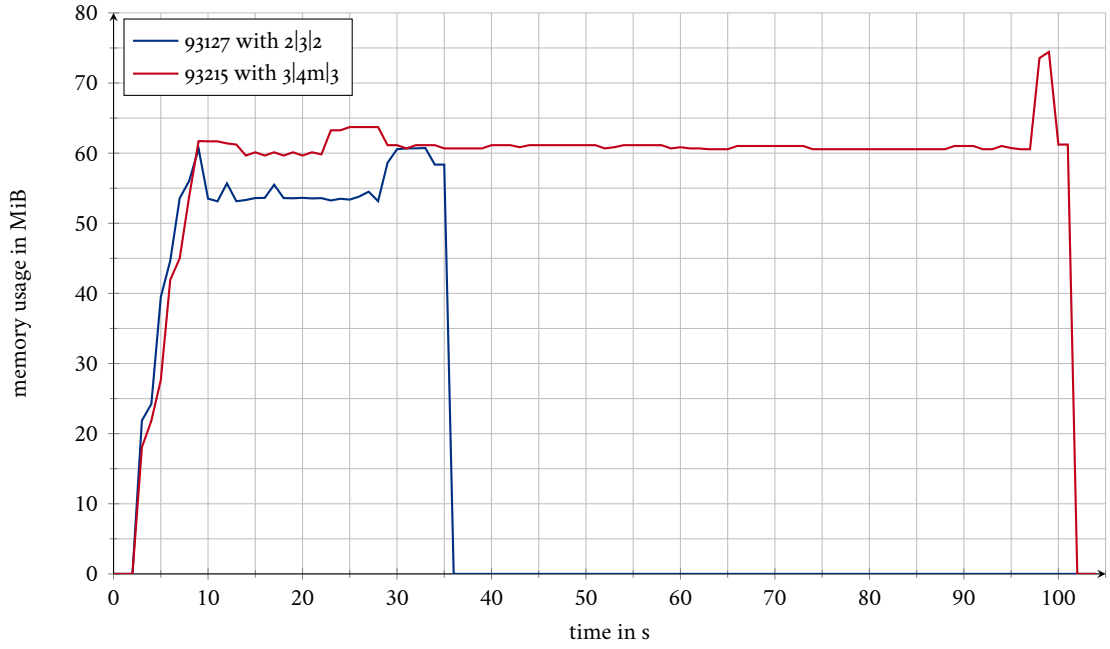


FIGURE 7.5: Memory usage in MiB during two runs of the algorithm. Both the greedy heuristic and local search improvement have been executed, using the combined objective.

rate of the operator across the set of results. The second, shown in Equation 7.2 shows the total acceptance rate across the set of results. The results used are the results that were previously used for the values in Tables A.11 to A.14.

$$acceptance_{average} = \frac{1}{\#\text{ results}} \sum \left[\frac{\#\text{ accepted}}{\#\text{ accepted} + \#\text{ rejected}} \right] \tag{7.1}$$

$$acceptance_{consolidated} = \frac{\sum \#\text{ accepted}}{\sum \#\text{ accepted} + \#\text{ rejected}} \tag{7.2}$$

The results are shown in Table A.19. We can see that the *Place immmediate predecessor*, *Swap within line of work*, and *Replace random* operators are the most accepted across both measures, with the *Place immediate predecessor* being the most accepted when considering the first measure, and the *Swap within line of work* being the most accepted when considering the second measure. The other operators have very low acceptance rates, most notably the *Delete random* operator. We believe that this occured because of the large penalty incurred when removing an activity from the schedule: we have set the weight $w_{unscheduled}$ to -1000. The temperature settings used do not seem to allow a solution with such a large decrease in objective function to be accepted.

The *Place immediate predecessor* operator was accepted exceptionally well in two scenarios: 93127 and 112487. In these two scenarios, the acceptance rate of the operator is around 55 %. The operator has an acceptance rate of around 28 % with scenario 112085, and an acceptance rate between 1 % and 15 % with the other scenarios.

8 CONCLUSION

We give some final remarks in this chapter. Section 8.1 contains the conclusion, and Section 8.2 contains some ways to improve the model.

8.1 Conclusion

In this thesis, we have presented an efficient metaheuristic for solving the problem of finding lines of work for members of staff, given a set of activities with release dates, deadlines, and precedence constraints, while keeping walking distances, flexibility, and fairness into account. The method, which is based on starting with an initial solution obtained from a greedy heuristic, and enhancing the solution using simulated annealing, is able to find solutions consistently for at least one of the considered staff configurations for nineteen out of the twenty scenarios considered.

Furthermore, since we model an instance and solution as a simple temporal network, we are able to disregard the fixed starting and completion times. Using the STN allows us to easily check whether executing an activity earlier or later than originally planned is allowed. While this is also possible by using the earliest starting times, using the STN allows for model enhancements, such as maximum time lags between activities, i.e. constraints specifying that an activity must be executed within a certain amount of time compared to another activity. It also allows us to use existing measures for estimating the flexibility of a solution, such as the concurrent flexibility.

Our objective consists of three components: the walking distances, the flexibility, and the maximum of the standard deviation per type of staff. The best results for each component individually were found when optimizing that specific component, however, optimizing over a weighted sum of the three components gave the second best results for each component.

The algorithm is able to find solutions efficiently, with upto 2 000 iterations per second on easier scenarios, and around 350 iterations per second on the most difficult scenario considered. In comparison, the MIP solver was unable to find solutions within the time limit for around half of the scenarios considered. The main bottleneck is the flexibility algorithm used, seeing as calculating the flexibility amounts to about 70 % of the runtime of the application.

8.2 Model enhancements

Other flexibility measure The measures for the flexibility of an STN mentioned in Section 3.3 only deal with the degree of freedom in moving individual activities in time. It might also be interesting to look at the degree of freedom in moving a set of related activities in time, e.g. all activities

concerning a single EMU, or all activities concerning a single member of staff. To do this, we can make use of the paths in the STN. We can calculate the degree of freedom in choosing the placement of the entire path by comparing the earliest execution time of the earliest activity in the path, and the latest execution time of the latest activity in the path. However, some tasks are fixed in time, such as the Arrive/Exit tasks, and the dummy tasks for the start and end of the shifts. Furthermore, the dummy activities for the breaks are also restricted by release dates and deadlines. Because of these restrictions, we only consider paths where each activity is completely free to move around in time. Note that this method is only suited for STNs that can be converted to a directed acyclic graph, as shown in Section 3.4.1. The following description assumes the model from Section 3.2 using the optimizations in Sections 3.4.1 and 3.4.2, i.e. the STN containing only a variable for the completion time of each variable, and two dummy variables t_0 and t_0'.

Let $S = (T, C)$ be an STN, and $G_S' = (V, A)$ be the associated distance graph, as a directed acyclic graph. Let Π_S be the set containing all paths in G_S', where a path consists of activities $[\pi_0, \pi_1, \ldots, \pi_m]$, where π_0 is the final activity in time, and π_m the first. Let the set $\Pi_S' \subseteq \Pi_S$ be the set of paths in S where every activity is free to move in time, i.e. $\Pi_S' = \{ \pi \in \Pi \mid \forall i \in \Pi : r_i = 0 \wedge d_i = \infty \}$. We then define the flexibility as the minimum of the slack over all paths:

$$flex = \min_{\pi \in \Pi_S'} lt(\pi_0) - et(\pi_m)$$

where $et(t_i) = -D_S[t_i, 0]$ and $lt(t_i) = D_S[0, t_i]$, and D_S is the distance matrix of the graph G_S', see Property 3.3. In our model, the set Π_S' contains paths corresponding to

- The "line of work" corresponding to each EMU, without the corresponding Arrive and Exit activities.
- The "line of work" corresponding to each non-human resource, such as tracks without any Arrive and Exit activities.
- The partial line of work for each member of staff starting with the first non-dummy activity in the shift and ending with the last activity before the break.
- The partial line of work for each member of staff starting with the first activity past the break, and ending with the last non-dummy activity in the shift.

Plan stability A real-life scenario is subject to disruptions, such as tasks taking longer than planned, or starting later. We have tried to keep disruptions into account, by optimizing the flexibility of the solution, measured as the concurrent flexibility of the corresponding simple temporal network. If there are any disruptions, the solution should be able to handle these. However, if the disruptions become too big, the solution may not be able to absord them. In that case, a new solution must be generated.

Since we are dealing with personnel, and not with machines, it is desired that there is a form of plan stability: the new plan should be similar to the old plan, with small modifications to ensure feasibility under the changed circumstances. This is because the members of staff prefer some form of certainty when handed their roster for the day: it is not appreciated when the roster changes completely during their shift.

Plan stability could be handled in the algorithm by restricting the choice of neighbourhood, for example by disallowing the *Delete random* and *Swap within line of work* operators. However, since all operators move activities around, this is likely to be insufficient.

Another method to handle plan stability would be to use a different objective function, which represents the difference between the previous, now infeasible plan, and the newly generated plan. An example of such an objective function would be the number of activities that have a different predecessor in the new solution when compared to the old solution.

Integrating the staff scheduling with the rest of the problem The staff scheduling problem presented in this thesis is a part of a larger planning problem, which also contains planning the activities themselves, and finding shunting routes, among other things. The contents of the larger problem are explained in more detail in Section 1.2. Because of this, an instance of our algorithm is created from a solution to the other subproblems. However, since the method used to solve the staff scheduling problem is similar to the method used by Van den Broek [5], and since our method consists of adding extra constraints to a solution found by the algorithm by Van den Broek [5], it should also be possible to integrate our method for finding lines of work in the algorithm by Van den Broek [5], resulting in an algorithm which solves the complete planning problem.

BIBLIOGRAPHY

[1] J. M. van den Akker, J. A. Hoogeveen and J. W. van Kempen. Using column generation to solve parallel machine scheduling problems with minmax objective functions. In: *Journal of Scheduling* **15**:6 (2012), 801–810. DOI: 10.1007/s10951-010-0191-z (cit. on p. 47).

[2] M. A. Aloulou and M.-C. Portmann. An Efficient Proactive Reactive Scheduling Approach to Hedge against Shop Floor Disturbances. In: *In Proceedings of the 1 st Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2003.* 337–362 (cit. on p. 7).

[3] C. Artigues and F. Roubellat. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. In: *European Journal of Operational Research* **127**:2 (2000), 297–316. DOI: 10.1016/S0377-2217(99)00496-8 (cit. on p. 5).

[4] R. Bellman. On a routing problem. In: *Quarterly of applied mathematics* **16**:1 (1958), 87–90. DOI: 10.1090/qam/102435 (cit. on p. 23).

[5] R. W. van den Broek. *Train Shunting and Service Scheduling: an integrated local search approach*. MSc thesis. Universiteit Utrecht, 2016. URL: https://dspace.library.uu.nl/handle/1874/338269 (cit. on pp. 1, 4, 5, 9, 12, 44, 55, 71).

[6] A. Cesta, A. Oddi and S. F. Smith. Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In: *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998*. Ed. by R. G. Simmons, M. M. Veloso and S. F. Smith. AAAI, 1998, 214–223. ISBN: 1-57735-052-9. URL: http://www.aaai.org/Library/AIPS/1998/aips98-026.php (cit. on p. 7).

[7] J. Cordeau, G. Laporte, F. Pasin and S. Ropke. Scheduling technicians and tasks in a telecommunications company. In: *Journal of Scheduling* **13**:4 (2010), 393–409. DOI: 10.1007/s10951-010-0188-7 (cit. on p. 6).

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8 (cit. on p. 32).

[9] F. Deblaere, E. Demeulemeester, W. Herroelen and S. van de Vonder. Robust Resource Allocation Decisions in Resource-Constrained Projects. In: *Decision Sciences* **38**:1 (2007), 5–37. DOI: 10.1111/j.1540-5915.2007.00147.x (cit. on pp. 5, 7).

[10] R. Dechter, I. Meiri and J. Pearl. Temporal Constraint Networks. In: *Artificial Intelligence* **49**:1-3 (1991), 61–95. DOI: 10.1016/0004-3702(91)90006-6 (cit. on pp. 7, 21).

[11] M. van Dommelen. *Scheduling train service activities to determine capacity*. BSc thesis. Vrije Universiteit Amsterdam, 2015. URL: www.ubvu.vu.nl/pub/index_oclc.cfm?SearchObjectId=8&objectid=109&ordering=1&openitem=177349 (cit. on p. 4).

[12] P.-F. Dutot, A. Laugier and A.-M. Bustos. Technicians and interventions scheduling for telecommunications. In: *France Telecom R&D* (2006). URL: http://challenge.roadef.org/2007/files/sujet2.en.pdf (cit. on p. 6).

[13] M. Firat and C. A. J. Hurkens. An improved MIP-based approach for a multi-skill workforce scheduling problem. In: *Journal of Scheduling* **15**:3 (2012), 363–380. DOI: 10.1007/s10951-011-0245-x (cit. on p. 6).

[14] R. W. Floyd. Algorithm 97: Shortest path. In: *Communications of the ACM* **5**:6 (1962), 345. DOI: 10.1145/367766.368168 (cit. on pp. 31, 33, 57, 58).

[15] L. Ford and D. Fulkerson. *Flows in networks*. Princeton U. Press, Princeton, NJ, 1962 (cit. on p. 23).

[16] R. Freling, A. P. M. Wagelmans and J. M. P. Paixão. Models and Algorithms for Single-Depot Vehicle Scheduling. In: *Transportation Science* **35**:2 (2001), 165–180. DOI: 10.1287/trsc.35.2.165.10135 (cit. on p. 44).

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7 (cit. on p. 18).

[18] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*. 2016. URL: http://www.gurobi.com (cit. on p. 59).

[19] D. van den Heuvel. *Decomposing and Interactively Solving a High Dimensional Problem: Scheduling Trains on Shunting Yards*. MSc thesis. TU Delft, 2017. URL: http://resolver.tudelft.nl/uuid:3a4b2767-6172-4eeb-a6b4-1ddae7b531c5 (cit. on p. 4).

[20] H. T. Hsu. An Algorithm for Finding a Minimal Equivalent Graph of a Digraph. In: *Journal of the ACM* **22**:1 (1975), 11–16. DOI: 10.1145/321864.321866 (cit. on pp. 57, 58).

[21] L. Hunsberger. Algorithms for a Temporal Decoupling Problem in Multi-Agent Planning. In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.* Ed. by R. Dechter and R. S. Sutton. AAAI Press / The MIT Press, 2002, 468–475. URL: http://www.aaai.org/Library/AAAI/2002/aaai02-071.php (cit. on pp. 7, 29, 33, 35).

[22] D. B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. In: *Journal of the ACM* **24**:1 (Jan. 1977), 1–13. DOI: 10.1145/321992.321993 (cit. on p. 33).

[23] A. B. Kahn. Topological sorting of large networks. In: *Commununications of the ACM* **5**:11 (1962), 558–562. DOI: 10.1145/368996.369025 (cit. on pp. 15, 16).

[24] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by Simulated Annealing. In: *Science* **220**:4598 (1983), 671–680. DOI: 10.1126/science.220.4598.671 (cit. on pp. 39, 41).

[25] A. A. Kovacs, S. N. Parragh, K. F. Doerner and R. F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. In: *Journal of Scheduling* **15**:5 (2012), 579–600. DOI: 10.1007/s10951-011-0246-9 (cit. on p. 6).

[26] L. G. Kroon, R. M. Lentink and A. Schrijver. Shunting of Passenger Train Units: An Integrated Approach. In: *Transportation Science* **42**:4 (2008), 436–449. DOI: 10.1287/trsc.1080.0243 (cit. on p. 4).

[27] H. W. Kuhn. The Hungarian method for the assignment problem. In: *Naval Research Logistics Quarterly* **2**:1-2 (1955), 83–97. DOI: 10.1002/nav.3800020109 (cit. on pp. 31, 44).

[28] R. Leus and W. Herroelen. Stability and resource allocation in project planning. In: *IIE Transactions* **36**:7 (2004), 667–682. DOI: 10.1080/07408170490447348 (cit. on p. 5).

[29] S. Mountakis, T. Klos and C. Witteveen. Temporal Flexibility Revisited: Maximizing Flexibility by Computing Bipartite Matchings. In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*. Ed. by R. I. Brafman, C. Domshlak, P. Haslum and S. Zilberstein. AAAI Press, 2015, 174–178. ISBN: 978-1-57735-731-5. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10610 (cit. on pp. 31, 65).

[30] V. Pillac, C. Guéret and A. L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. In: *Optimization Letters* **7**:7 (2013), 1525–1535. DOI: 10.1007/s11590-012-0567-4 (cit. on p. 6).

[31] L. R. Planken. *Algorithms for Simple Temporal Reasoning*. PhD thesis. TU Delft, 2013. DOI: 10.4233/uuid:2e41be6a-5220-421c-bef8-bbb8f91c128f (cit. on p. 23).

[32] N. Policella, A. Cesta, A. Oddi and S. F. Smith. From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling. In: *AI Communications* **20**:3 (2007), 163–180. URL: http://content.iospress.com/articles/ai-communications/aic403 (cit. on p. 12).

[33] N. Policella, A. Oddi, S. F. Smith and A. Cesta. Generating Robust Partial Order Schedules. In: *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*. Ed. by M. Wallace. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, 496–511. ISBN: 3-540-23241-9. DOI: 10.1007/978-3-540-30201-8_37 (cit. on p. 5).

[34] N. Policella, S. F. Smith, A. Cesta and A. Oddi. Generating Robust Schedules through Temporal Flexibility. In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*. Ed. by S. Zilberstein, J. Koehler and S. Koenig. AAAI, 2004, 209–218. ISBN: 1-57735-200-9. URL: http://www.aaai.org/Library/ICAPS/2004/icaps04-026.php (cit. on p. 7).

[35] G. Ramalingam, J. Song, L. Joskowicz and R. E. Miller. Solving Systems of Difference Constraints Incrementally. In: *Algorithmica* **23**:3 (1999), 261–275. DOI: 10.1007/PL00009261 (cit. on p. 23).

[36] S. Warshall. A Theorem on Boolean Matrices. In: *Journal of the ACM* **9**:1 (1962), 11–12. DOI: 10.1145/321105.321107 (cit. on pp. 31, 33, 57, 58).

[37] M. Wilson, T. Klos, C. Witteveen and B. Huisman. Flexibility and decoupling in Simple Temporal Networks. In: *Artificial Intelligence* **214**: (2014), 26–44. DOI: 10.1016/j.artint.2014.05.003 (cit. on pp. 7, 28–30, 33).

[38] F. E. Wolfhagen. *The Train Unit Shunting Problem with Reallocation*. MSc thesis. Erasmus Universiteit Rotterdam, 2017. URL: http://hdl.handle.net/2105/37696 (cit. on p. 4).

A

TABLES

	H	52	53	54	55	56	57	58	59	60	61	62	63	64
H	0													
52	290	0												
53	285	5	0											
54	280	405	5	0										
55	175	410	405	5	0									
56	170	415	410	405	5	0								
57	165	420	415	410	405	5	0							
58	160	425	420	415	410	305	5	0						
59	155	430	425	420	415	310	305	5	0					
60	1130	1420	1415	1410	1405	1300	1295	1290	1285	0				
61	525	815	810	805	800	695	690	685	680	1015	0			
62	320	815	810	805	800	695	690	685	680	1015	5	0		
63	725	1015	1010	1005	1000	895	890	885	880	405	610	610	0	
64	100	390	385	380	375	270	265	260	255	1030	425	220	625	0

Table A.1: Distance matrix for Kleine Binckhorst. Distances are in m.

Id	# EMUs	# Movements	# Tasks	# Movements + # Tasks	$\|\mathcal{A}\|$	$\|\mathcal{P}\|$	$flex/\|\mathcal{A}\|$
93240	11	71	20	91	140	270	25.3
93127	14	65	26	91	130	253	25.3
94882	14	73	27	100	148	292	29.89
93366	13	75	25	100	156	308	15.19
95171	14	82	28	110	164	334	23.47
112085	14	84	26	110	176	350	17.38
112487	15	93	27	120	190	372	25.89
113639	14	91	29	120	185	363	19.97
102809	15	97	30	127	204	398	23.85
102861	15	100	30	130	212	419	21.18
102816	15	106	30	136	223	460	20.44
90321	16	110	31	141	233	468	22.67
110562	15	110	30	140	226	471	21.1
102058	15	124	28	152	252	533	18.97
93384	15	122	30	152	260	548	20.15
93960	15	130	32	162	274	556	19.8
100322	15	133	28	161	277	562	18.68
111658	20	131	40	171	278	552	21.51
101062	15	142	30	172	296	627	20.8
93215	21	136	40	176	290	596	24.13

TABLE A.2: The selected scenarios and the amount of tasks in the original solution, and the statistics without preprocessing.

Id	$\lvert\mathcal{A}\rvert$	%	$\lvert\mathcal{P}\rvert$	%	$flex/\lvert\mathcal{A}\rvert$
93240	98	−30.00 %	212	−21.48 %	25.77
93127	96	−26.15 %	208	−17.79 %	26.06
94882	105	−29.05 %	234	−19.86 %	26
93366	108	−30.77 %	246	−20.13 %	14.99
95171	115	−29.88 %	265	−20.66 %	17.72
112085	117	−33.52 %	267	−23.71 %	19.76
112487	127	−33.16 %	284	−23.66 %	29.23
113639	125	−32.43 %	280	−22.87 %	14.5
102809	137	−32.84 %	308	−22.61 %	31.78
102861	140	−33.96 %	325	−22.43 %	22.66
102816	148	−33.63 %	360	−21.74 %	17.32
90321	152	−34.76 %	352	−24.79 %	24.32
110562	149	−34.07 %	361	−23.35 %	17.41
102058	159	−36.90 %	396	−25.70 %	20
93384	164	−36.92 %	415	−24.27 %	25.32
93960	172	−37.23 %	416	−25.18 %	18.25
100322	171	−38.27 %	412	−26.69 %	17.98
111658	181	−34.89 %	419	−24.09 %	19.69
101062	182	−38.51 %	462	−26.32 %	19.08
93215	191	−34.14 %	466	−21.81 %	25.16
Average		−33.56 %		−22.96 %	

TABLE A.3: Statistics on each scenario after the Wait tasks are removed.

Id	$\lvert\mathcal{A}\rvert$	%	$\lvert\mathcal{P}\rvert$	%	$flex/\lvert\mathcal{A}\rvert$
93240	84	−14.29 %	195	−8.02 %	28.67
93127	79	−17.71 %	181	−12.98 %	31.59
94882	88	−16.19 %	203	−13.25 %	28.55
93366	91	−15.74 %	224	−8.94 %	16.93
95171	98	−14.78 %	231	−12.83 %	18.52
112085	100	−14.53 %	240	−10.11 %	22.62
112487	107	−15.75 %	250	−11.97 %	34.04
113639	108	−13.60 %	247	−11.79 %	14.02
102809	117	−14.60 %	282	−8.44 %	37.01
102861	120	−14.29 %	301	−7.38 %	26.15
102816	128	−13.51 %	328	−8.89 %	18.62
90321	131	−13.82 %	319	−9.38 %	27.67
110562	129	−13.42 %	340	−5.82 %	19.59
102058	139	−12.58 %	360	−9.09 %	21.94
93384	144	−12.20 %	396	−4.58 %	28.13
93960	152	−11.63 %	405	−2.64 %	19.38
100322	151	−11.70 %	384	−6.80 %	19.84
111658	155	−14.36 %	397	−5.25 %	22.67
101062	162	−10.99 %	432	−6.49 %	19.79
93215	164	−14.14 %	436	−6.44 %	28.73
Average		−13.99 %		−8.55 %	

TABLE A.4: Statistics on each scenario after the Wait, Arrive, and Exit tasks are removed.

Id	$\|\mathcal{A}\|$	%	$\|\mathcal{P}\|$	%	$flex/\|\mathcal{A}\|$
93240	84	0 %	107	−45.13 %	28.67
93127	79	0 %	105	−41.99 %	31.59
94882	88	0 %	121	−40.39 %	28.55
93366	91	0 %	129	−42.41 %	16.93
95171	98	0 %	130	−43.72 %	18.52
112085	100	0 %	140	−41.67 %	22.62
112487	107	0 %	147	−41.20 %	34.04
113639	108	0 %	145	−41.30 %	14.02
102809	117	0 %	163	−42.20 %	37.01
102861	120	0 %	179	−40.53 %	26.15
102816	128	0 %	181	−44.82 %	18.62
90321	131	0 %	182	−42.95 %	27.67
110562	129	0 %	173	−49.12 %	19.59
102058	139	0 %	189	−47.50 %	21.94
93384	144	0 %	204	−48.48 %	28.13
93960	152	0 %	213	−47.41 %	19.38
100322	151	0 %	206	−46.35 %	19.84
111658	155	0 %	216	−45.59 %	22.67
101062	162	0 %	224	−48.15 %	19.79
93215	164	0 %	237	−45.64 %	28.73
Average		0 %		−44.33 %	

TABLE A.5: Statistics on each scenario after the Wait, Arrive, and Exit tasks are removed, and the transitive reduction algorithm has been applied.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	N	O	O	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
2|3|2	T	O	O	O	O	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N
2|4el|2	T	O	O	O	O	T	T	N	N	N	T	N	T	N	N	N	N	N	N	N
2|4m|2	T	O	T	O	O	T	T	N	T	T	T	N	T	N	N	N	N	N	N	N
3|3|3	T	O	O	O	O	T	N	N	N	N	T	N	N	N	N	N	N	N	N	N
3|4el|3	T	O	O	O	O	T	N	N	N	N	N	N	N	N	N	N	N	N	N	N
3|4m|3	O	O	T	O	O	T	T	N	N	N	T	N	N	N	N	N	N	N	N	N

TABLE A.6: Status of the MIP solver for each scenario. *O* indicates that the problem has been solved to optimality, *T* indicates that a solution has been found, but that optimality could not be proven within the time limit, *N* indicates that no solution was found within the time limit.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	7	8	8	4	7	8	7	0	8	0	8	0	3	0	1	0	5	0	0	0
2|4el|2	8	8	8	3	8	8	8	0	8	0	8	8	8	0	6	0	8	0	6	0
2|4m|2	5	8	8	8	8	8	8	0	8	2	8	0	8	0	8	0	8	0	1	1
3|3|3	8	8	8	4	8	8	8	0	8	5	8	8	3	0	8	0	6	4	0	0
3|4el|3	8	8	8	0	8	8	8	0	8	8	8	8	8	7	2	0	7	6	8	0
3|4m|3	8	8	8	8	8	8	8	0	8	8	8	8	8	8	8	0	8	8	8	8

TABLE A.7: Number of solutions found (out of 8) per scenario and staff combination, using only the greedy heuristic with the flexibility objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|4el|2	0	0	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2|4m|2	2	0	1	7	0	8	0	0	2	0	0	0	1	0	0	0	0	0	0	0
3|3|3	1	8	2	0	0	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3|4el|3	0	7	0	0	5	4	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3|4m|3	4	8	7	7	1	8	5	0	4	0	7	0	5	1	0	0	0	0	0	0

TABLE A.8: Number of solutions found (out of 8) per scenario and staff combination, using only the greedy heuristic with the fairness objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	6	8	0	2	0	5	1	0	0	0	0	0	6	0	0	0	0	0	0	0
2|4el|2	5	7	1	0	6	3	1	0	0	0	0	6	5	0	0	0	0	0	0	0
2|4m|2	7	8	7	6	3	8	2	0	0	0	0	1	6	0	1	0	0	0	0	0
3|3|3	7	8	2	3	2	3	4	0	0	1	5	2	1	0	0	0	0	0	0	0
3|4el|3	6	8	2	0	7	2	1	0	0	2	1	7	3	0	0	0	0	3	0	0
3|4m|3	8	8	8	6	8	8	7	0	6	3	8	7	8	7	3	0	1	2	0	0

TABLE A.9: Number of solutions found (out of 8) per scenario and staff combination, using only the greedy heuristic with the walking distance objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	8	8	8	5	1	8	0	0	8	0	8	2	3	0	0	0	0	0	0	0
2|4el|2	8	8	8	0	8	8	0	0	8	0	8	8	8	0	0	0	0	0	3	0
2|4m|2	8	8	8	8	8	8	1	0	8	0	8	8	0	0	8	0	8	0	0	0
3|3|3	8	8	8	7	1	8	8	0	8	8	8	8	0	0	0	0	0	0	0	0
3|4el|3	8	8	8	6	8	8	5	0	8	8	8	8	6	0	0	0	5	2	1	0
3|4m|3	8	8	8	8	8	8	8	0	8	8	8	8	8	8	8	0	8	8	0	8

TABLE A.10: Number of solutions found (out of 8) per scenario and staff combination, using only the greedy heuristic with the combined objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	8	8	8	5	5	8	8	0	8	0	8	6	8	0	6	0	6	0	0	0
2|4el|2	8	8	8	5	8	8	8	0	8	0	8	8	8	0	8	0	8	0	6	0
2|4m|2	8	8	8	8	8	8	8	0	8	3	8	8	8	0	8	8	8	0	2	3
3|3|3	8	8	8	6	8	8	8	0	8	7	8	8	7	6	8	0	6	8	0	0
3|4el|3	8	8	8	1	8	8	8	0	8	8	8	8	8	8	7	0	7	8	8	0
3|4m|3	8	8	8	8	8	8	8	0	8	8	8	8	8	8	8	8	8	8	8	8

TABLE A.11: Number of solutions found (out of 8) per scenario and staff combination, using the greedy heuristic and local search improvement with the flexibility objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	5	7	5	3	0	8	5	0	8	0	3	6	6	0	0	0	0	0	0	0
2|4el|2	7	8	7	0	6	7	1	0	8	0	4	7	7	0	1	0	0	0	0	0
2|4m|2	7	8	8	8	1	8	7	0	8	0	7	5	8	0	2	0	0	0	0	0
3|3|3	3	8	8	2	1	8	8	0	7	0	6	4	6	0	1	0	0	0	0	0
3|4el|3	6	8	6	0	8	4	7	0	5	1	7	7	7	3	0	0	0	7	0	0
3|4m|3	7	8	8	8	4	8	8	0	8	7	8	6	8	8	3	0	5	6	0	3

TABLE A.12: Number of solutions found (out of 8) per scenario and staff combination, using the greedy heuristic and local search improvement with the fairness objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	7	8	5	6	3	8	3	0	8	0	5	4	6	0	1	0	0	0	0	0
2|4el|2	8	8	3	3	8	7	6	0	8	0	1	8	7	0	1	0	0	0	0	0
2|4m|2	8	8	8	8	6	8	6	0	8	0	8	7	8	0	4	0	5	0	0	0
3|3|3	8	8	8	6	4	7	8	0	7	3	7	5	8	1	0	0	0	3	0	0
3|4el|3	8	8	5	5	8	5	5	0	8	2	5	8	7	1	0	0	0	7	0	0
3|4m|3	8	8	8	8	8	8	8	0	7	6	8	8	8	8	7	1	6	6	0	7

TABLE A.13: Number of solutions found (out of 8) per scenario and staff combination, using the greedy heuristic and local search improvement with the walking distance objective.

	93240	93127	94882	93366	95171	112085	112487	113639	102809	102861	102816	90321	110562	102058	93384	93960	100322	111658	101062	93215
2|2|2	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2|3|2	8	8	7	8	6	8	6	0	8	0	8	8	8	0	1	0	0	0	0	0
2|4el|2	8	8	8	0	8	8	6	0	8	0	8	8	8	0	6	0	0	0	2	0
2|4m|2	8	8	8	8	8	8	6	0	8	0	8	8	7	0	8	8	8	0	0	5
3|3|3	8	8	8	8	3	8	8	0	8	8	8	8	7	2	0	0	0	4	0	0
3|4el|3	8	8	8	8	8	8	8	0	8	8	8	8	7	3	8	0	7	4	1	0
3|4m|3	8	8	8	8	8	8	8	0	8	8	8	8	8	8	8	8	8	8	0	8

TABLE A.14: Number of solutions found (out of 8) per scenario and staff combination, using the greedy heuristic and local search improvement with the combined objective.

	2|2|2	2|3|2	2|4el|2	2|4m|2	3|3|3	3|4el|3	3|4m|3
93240	-	9.98	11.66	11.56	12.32	13.69	11.84
93127	7.31	10.46	11.32	12.41	15.61	16.68	17.52
94882	-	10.15	9.99	12.48	11.48	12.61	14.12
93366	-	7.23	-	8.40	10.28	10.38	11.90
95171	-	7.82	8.40	9.16	10.36	11.80	12.89
112085	-	7.83	8.12	10.11	11.91	12.17	14.01
112487	-	8.02	8.56	9.85	12.04	12.34	13.23
113639	-	-	-	-	-	-	-
102809	-	5.84	6.18	7.63	9.41	9.38	10.89
102861	-	-	-	-	7.78	7.73	10.31
102816	-	4.95	5.83	7.79	7.01	7.40	8.49
90321	-	6.55	7.20	8.23	8.69	11.17	10.35
110562	-	4.99	5.88	6.30	7.35	8.28	8.79
102058	-	-	-	-	7.71	6.70	9.17
93384	-	4.76	5.42	7.11	-	7.66	10.51
93960	-	-	-	5.57	-	-	7.86
100322	-	-	-	6.91	-	7.08	8.57
111658	-	-	-	-	7.27	8.20	8.90
101062	-	-	4.19	-	-	6.44	-
93215	-	-	-	4.48	-	-	6.65

TABLE A.15: The average of the concurrent flexibility per activity of the found solutions when using the greedy heuristic with the local search improvement step and the combined objective (see Table A.14). A hyphen indicates that no solutions were found.

	2|2|2	2|3|2	2|4el|2	2|4m|2	3|3|3	3|4el|3	3|4m|3
93240	-	4.4 %	4.1 %	1.1 %	3.0 %	2.1 %	1.0 %
93127	9.4 %	2.1 %	2.3 %	3.4 %	13.8 %	10.7 %	12.6 %
94882	-	4.5 %	8.3 %	8.5 %	10.1 %	11.0 %	17.7 %
93366	-	2.6 %	-	5.5 %	5.9 %	0.9 %	6.4 %
95171	-	2.4 %	11.5 %	1.8 %	2.2 %	16.8 %	16.4 %
112085	-	10.5 %	12.3 %	10.0 %	21.9 %	16.0 %	13.4 %
112487	-	2.6 %	4.6 %	3.6 %	1.6 %	3.6 %	3.5 %
113639	-	-	-	-	-	-	-
102809	-	6.2 %	2.5 %	2.6 %	9.3 %	6.1 %	8.7 %
102861	-	-	-	-	6.2 %	6.5 %	1.7 %
102816	-	2.8 %	6.4 %	11.0 %	3.1 %	1.1 %	11.2 %
90321	-	0.5 %	2.0 %	0.2 %	5.1 %	6.5 %	4.9 %
110562	-	2.2 %	2.9 %	3.5 %	3.1 %	2.3 %	2.0 %
102058	-	-	-	-	1.9 %	0.6 %	5.2 %
93384	-	6.9 %	4.2 %	11.8 %	-	3.3 %	13.0 %
93960	-	-	-	2.3 %	-	-	3.3 %
100322	-	-	-	2.5 %	-	0.8 %	5.1 %
111658	-	-	-	-	3.7 %	1.9 %	1.5 %
101062	-	-	1.0 %	-	-	1.5 %	-
93215	-	-	-	1.1 %	-	-	1.0 %

TABLE A.16: The maximum percentage with which all activity durations can increase, while keeping the solutions found consistent. The numbers shown are averages over the solutions found in Table A.14. A hyphen indicates that no solutions were found.

	2|2|2	2|3|2	2|4e1|2	2|4m|2	3|3|3	3|4e1|3	3|4m|3
93240	1648	1340	1349	1245	1295	1203	1205
93127	1930	1702	1335	1248	1152	1075	1100
94882	1507	1259	1186	1151	1120	965	1048
93366	1411	1220	1230	1068	1034	993	926
95171	1239	1090	1004	985	954	841	785
112085	1207	1094	970	945	860	777	815
112487	1062	1046	919	971	768	737	696
113639	1253	1146	1040	997	969	923	873
102809	1066	857	799	737	683	665	637
102861	976	915	858	859	676	687	624
102816	958	773	721	691	653	617	595
90321	863	713	705	696	613	578	564
110562	955	818	758	725	691	575	593
102058	806	751	728	705	665	675	537
93384	739	708	634	546	511	518	418
93960	712	640	631	572	559	493	462
100322	729	717	664	534	598	512	474
111658	691	623	587	514	513	483	400
101062	683	661	593	613	544	459	474
93215	576	555	563	518	433	425	368

TABLE A.17: Performance of the local search method measured as the number of iterations per second.

	2|2|2	2|3|2	2|4el|2	2|4m|2	3|3|3	3|4el|3	3|4m|3
93240	60941	41829	40677	41152	43646	41702	43221
93127	63204	45245	43510	41487	54578	46371	47740
94882	46266	42596	42899	42949	45668	43801	47140
93366	83008	79943	48907	42563	48833	52022	43974
95171	98983	118026	66392	43709	66822	63562	45684
112085	154427	43775	45055	43313	45526	47997	46329
112487	69162	136283	134428	99314	59718	85334	47825
113639	101323	101153	108052	48225	65093	62865	47590
102809	84385	45533	38086	39294	42160	38350	40684
102861	153861	178026	69029	126403	36719	51102	39400
102816	123497	62882	42118	43186	44850	41456	43897
90321	103346	38803	54764	65716	45449	42166	41453
110562	110389	61372	73490	66731	141579	73022	44577
102058	119465	117010	60991	60560	86830	89126	41073
93384	171019	88259	127203	36623	70052	103442	39477
93960	169111	124702	98493	59330	139531	39757	52854
100322	179883	87972	78469	38959	110164	55083	40569
111658	104967	111244	144927	41665	145399	64686	41108
101062	122066	134964	52384	154260	110296	79847	87076
93215	206959	182147	82448	90028	134414	74393	44451

TABLE A.18: Performance of the local search method measured as the total number of iterations before termination.

Operator	$acceptance_{average}$	$acceptance_{consolidated}$
Bring related together	1.127 %	0.806 %
Place immediate predecessor	13.143 %	2.724 %
Relieve largest line of work	0.595 %	0.463 %
Swap within line of work	4.274 %	3.922 %
Insert random	0.335 %	0.089 %
Delete random	0.017 %	0.009 %
Replace random	2.093 %	1.602 %

TABLE A.19: Acceptance percentages.