CHALMERS
UNIVERSITY OF TECHNOLOGY

Project Matching Application Framework Using Metaheuristic Algorithms

Master's thesis in Computer Science – Algorithms, languages and logic

Jonathan Nilsson

Johannes Magnusson

Department of Signals and systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

Project Matching Application Framework Using Metaheuristic Algorithms

Jonathan Nilsson
&
Johannes Magnusson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and systems
Chalmers University of Technology
Gothenburg, Sweden 2018

Project Matching Application Framework Using Metaheuristic Algorithms
Jonathan Nilsson
Johannes Magnusson

Supervisors: Andreas Fhager, Department of Signals and systems and Peter Lindh, Squeed
Examiner: Andreas Fhager, Department of Signals and systems

Master's Thesis 2018
Department of Signals and systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Project Matching Application Framework Using Metaheuristic Algorithms
Jonathan Nilsson
Johannes Magnusson
Department of Signals and systems
Chalmers University of Technology

Abstract

Timetabling is something that a lot of businesses, public sectors and institutions have to deal with, which is probably why automating the process of creating good timetables is a well-known problem that has been well researched during the recent decades. This thesis introduces a timetabling problem named project matching problem, which is a problem of assigning workers to projects and schedule these projects into a limited time frame, while respecting certain constraints such as the workers preference towards these projects. In order to efficiently solve instances of this problem, an application framework based on the metaheuristic algorithm Iterated local search is developed and implemented using common software techniques and architectural patterns. As the project matching problem has a broad definition, the idea is that the area of use for this application framework comprises timetabling problems outside the domain of project organisation.

The results are based on running these problem instances through the application framework, where the results of these runs are presented in terms of both solution quality and algorithmic efficiency. The solution quality is based on analyses of the actual solutions that is produced while algorithmic efficiency is measured on the execution time of the algorithm, which offers a suggestion of how the solution quality improves as the execution time increases. For one of the problem instances this application framework produced an optimal solution, which offers an indication on the applicability of this framework.

Keywords: Metaheuristic algorithms, Combinatorial optimisation, Iterated local search, Automation framework, C#/.NET.

Acknowledgements

Thanks to Squeed for providing knowledge, office space and supplies. Special thanks to Peter Lindh and Andreas Fhager for being supervisors throughout this project.

Jonathan Nilsson & Johannes Magnusson

Contents

List of Figures

List of Algorithms

List of Tables

List of Codes

1
Introduction

This chapter provides a brief introduction of the problem to be solved and what the benefits of such a solution are. It suggests an application framework and presents the scope of which this thesis aims.

1.1 Background

Timetabling is something that a lot of businesses, public sectors and institutions have to deal with. Creating schedules for restaurant employees, allocating rooms for hospitalised patients or providing offices for project groups are just a few examples of this. Having timetables that are not as efficient as possible could cost money, lead to loss of resources or cause human discomfort. Letting humans create these schedules drains both time and energy that could be used on something else.

Automating the process of creating good timetables is a well-known problem that has been well researched during the recent decades. The variation of each timetabling does further create additional problems and it is therefore hard to develop one general algorithm that solve all these problems. One approach to achieve this generality however, is to create a framework for solving timetabling problems of a given type with a certain complexity, as less complex problems of the same type then could be reduced to this. When such a framework is implemented, it could be expanded to handle problems of other types as well. This framework could then be used for creating various systems to solve specific problems that are similar to the original problem.

1.2 Purpose

This thesis suggests an application framework to efficiently solve the problem of matching workers to projects based on preferences, combined with scheduling these projects into a timetable. The idea is furthermore that this problem is defined wide enough such that the area of use for the application framework comprises timetabling problems outside the domain of project organisation. The problem will be further described and explained in the problem description.

The purpose of this application framework is to facilitate the process of project planning as well as for making workers satisfied with the projects that they are assigned to. Not only could it make these timetables better, it could also save work hours within affected organisations. The advantage of an application framework over a specific system is that it could both be modified to solve similar problems and to be directly used for problems that is reducible to the original problem. Making this application framework efficient enough such that it can be used in practise to solve complex problems is thereby an important part of the purpose of this project.

1.3 Problem description

The project-matching problem stated by this thesis is essentially constituted by matching workers with projects which are in turn assigned to workers timeslots within a timetable. This process needs to respect worker-based constraints such as which projects that are preferred, the maximum work capacity of an individual and the preferred number of assigned simultaneous project roles. Meanwhile constraints based on the requirements of project and its associated project roles needs to be respected as well. These requirements consist of only assigning workers that are qualified, assigning each project role of a project and ensuring that projects start within its given time frame. The terminology with corresponding descriptions of the project-matching problem described above is found in Table 1.1.

Terminology	Description
Worker	Entity that possess project roles over its defined timeslots with the required work load.
Project role	Resource that can be assigned to a worker, that represents a certain group of task assignments within a project.
Timeslot	Represents the time units of which a worker disposes within the timetable, on which project roles are assigned to workers where a project role can stretch over several timeslots.
Project	Collection of project roles that have a common length as well as start and end times.
Preference	Value indicating a workers degree of satisfaction with a project role.
Skills	Resource used to represent qualifications required of a given worker for a certain project role.

Table 1.1: Terminology for problem description

1.4 Method

The purpose of this thesis is to develop an application framework containing an algorithm to solve the project-matching problem. This algorithm will be designed to solve specific instances representing the project-matching problem in the form of predefined test schedules. In order to justify the results these test schedules have at least one known solution each where no other solution is better, which offers a suggestion of a perfect solution as reference. This application framework will be developed using common software techniques and architectural patterns in order to promote future extensions while keeping a solid structure.

In order to produce an algorithm that solves the instances of our problem this thesis will focus on studying related works that addresses similar problems. This algorithm will thereby be built upon already known strategies and general methods that balances between quality of the solution as well as the algorithmic efficiency. The results will hence be presented with focus on both the quality of the results, which is how well the criteria of a produced schedule are met, as well as on the efficiency of the algorithm that produces these schedules.

1.5 Scope and limitations

This thesis aims to develop an application framework to efficiently solve a specific timetabling problem using a metaheuristic algorithm named Iterated local search. The quality of the solutions produced by the Iterated local search is decided by how close they are to the optimal solution, where the optimal solution is a matching where as many workers as possible are satisfied with the role in a specific project that they are assigned to. The purpose is however not to find an optimal solution, but rather to find a satisfying compromise between quality of result and time complexity of the algorithm. Results of the algorithm is measured analytically, while the time complexity of the implemented algorithms in the system is measured with various limits on maximum number of iterations.

Front-end and user interface is not a priority of this project, which means that the framework is limited to a minimum viable implementation in these aspects. The testing process of the framework does not contain a verification part. Software security and system security will not be considered during the implementation. Fault tolerance with regards to concurrent usage is not a priority of this project and is thereby kept to a minimum.

The framework is a proof of concept, which means that it is not a complete product. Focus has instead been on the theoretical background, quality and efficiency of the algorithm and calculations. The implementation is however open enough to allow for future modifications to among other things support the creation of systems that run continuously and to enable the insertion of a proper front-end.

The goals of this project are thereby listed as follows:

- Decide what constitutes an adequate solution regarding both quality and correctness.
- Create a proof of concept prototype of the application framework that produces such adequate solutions.

2
Theory

Since the project matching problem is constituted by matching a finite number of project roles and workers within a limited number of timeslots, the number of possible solutions is also finite. Achieving an optimal solution for the project matching problem is thereby a matter of minimising a penalty function for given constraints within a finite set, hence making it a combinatorial optimisation problem. [1, p. 1] This chapter provide the theoretical background of combinatorial optimisation problems, where the methods of solving combinatorial optimisation problems can be divided into the two categories exact methods and heuristic methods. As this thesis aims to balance between the quality of the results and the algorithms efficiency, the focus is on heuristic algorithms in general and the algorithm named Iterated local search in particular. In the section related work similar problem formulations and algorithmic strategies are presented.

2.1 Combinatorial optimisation

The project matching problem introduced in this thesis is a timetabling problem that satisfies certain constraints. A combinatorial optimisation problem minimising a penalty function controlled by a given set of constraints, hence the project matching problem is a combinatorial optimisation problem. [1, p. 1] [2, p. 9] Violating constrains within a combinatorial optimisation can either raise penalties or render the entire solution invalid, depending of the characteristic of this particular constraint. Constraints is thereby used to limit the number of accepted solutions within the solution space, as well as evaluate the valid solutions. Solutions that have certain similarities with each other are said to belong to the same neighbourhood, where the similarities are defined by a neighbourhood function. [2, p. 16]

2.1.1 Neighbourhoods

The neighbourhood of a given solution are other closely related or similar solutions. As the state space consists of neighbourhoods, it is essential to define good neighbourhood functions to find high quality local optimum solutions. A good local search needs to have a fine-tuned neighbourhood function to enable it to find similar but slightly better solutions. Neighbouring solutions in the project-matching problem

are for example solutions where a single project is moved one timeslot or where a project role is assigned to another worker, while solution where several projects is retracted, or all project roles have been reassigned would not be considered neighbouring. A more formal definition of neighbourhoods and neighbourhood function are stated below.

Given an instance of a combinatorial optimisation problem where X is the set of feasible solutions for the instance, a neighbourhood is defined as the solution $x \in X$ as $N(x) \subseteq X$ (where N is a function that maps a solution to a set of similar solutions). Solution x is a locally optimal with respect to a neighbourhood N if $c(x) \leq c(x') \ \forall x' \in N(x)$, where c is a function mapping a solution to a cost value. [3, p. 3]

2.1.2 Constraints

All combinatorial optimisation problems thrive to find a solution that respects certain constraints, or that tries to satisfy as many constraints as possible. Some constraints could be mandatory to respect in order for a solution to be regarded as valid, which often times are referred to as hard constraints. Other constraints could be provided with a weight to produce a penalty when violated, usually referred to as soft constraints. The purpose of an algorithm should then be to not break any hard constraint while trying to minimise the penalty caused by soft constraint violations [4, p. 342]. The quantity and strength of the constraints affects the size of and thereby also the number of neighbourhoods, which further have effect on how well the combinatorial optimisation problem is solved.

2.1.3 Solving combinatorial optimisation problems

The way of solving a CO problem is to find a globally optimal solution, in this case a solution with as low penalty as possible. A significant quantity of algorithms to solve these CO problems exists, which can be categorised as either complete or approximate. Complete algorithms guarantee an optimal solution in a bounded time for a finite set as in-data. Since CO problems are NP-hard, assuming $P \neq NP$, there exists no polynomial solutions for the complete algorithms. The approximate methods do not guarantee an optimal solution, but instead focuses on finding a solution that is good enough regarding solution quality while offering a decrease in time complexity compared to complete solutions. [5, pp. 1–2]

Metaheuristics is in general terms described as methods for exploring numerous solutions within a solution space by defining general constraints, which are subsequently instantiated for the specific problems to be solved. This method has to both explore new types of solutions or by making big changes to the current solutions and to

improve the already discovered ones by making small changes. [6, pp. 11–24]

An example of a metaheuristic algorithm is the Iterated local search [4, pp. 370–400], which creates an initial solution to be improved by an embedded heuristic throughout a series of iterations. In order for the Iterated local search to avoid iterating within the same neighbourhood and hence getting stuck in locally optimal solutions, a sub-algorithm named perturbation is used to alter the partial solutions during the run. Each iteration ends with an acceptance criterion that decides whether to keep the new solution or stay with the one from the previous iteration.

2.2 Related work

This section presents works related to this thesis, meaning that they have similar problem formulations or algorithmic strategies. The Aircraft landing problem is described together with an Iterated local search to solve it. Further the concept of University Timetabling Problem is reviewed together with the product UniTime that provides a solution.

2.2.1 Aircraft landing problem

The Aircraft Landing Problem is a combinatorial optimisation problem that consists of assigning a runway and a landing time to each arriving aircraft, where each aircraft has a target landing time. The goal is to minimise the total cost deviation from all target landing time while not violating the following set of constraints. Every aircraft is assigned to exactly one runway and at most one aircraft is assigned to a runway at any given time. Each aircraft's landing time needs to be within that aircraft's landing time window and the predefined separation time between two aircrafts landing on the same runway must be respected. [7, pp. 88–90]

In order to solve this problem, the proposed method is an implementation of the Iterated local search starting with a randomised greedy heuristic to provide an initial solution. This randomised greedy heuristic allocates the runway and landing time by randomly selecting 10% of the aircrafts while the rest of the aircrafts are assigned greedily based on their preferred landing time. [7, pp. 90–93]

The local search phase consists of a Variable Neighbourhood Descent (VSD), which is a single solution based metaheuristic that iteratively improves an initial solution using neighbourhood structures. The VSD tries to improve solutions produced by the perturbation phase by iteratively finding neighbouring solutions for a given number of iterations. More specifically the VSD has four different neighbourhoods where the first one is randomly selecting a runway and swaps all its aircrafts where it

leads to an improvement. The second neighbourhood randomly selects two runways and swap aircrafts between these where improvements are achieved and the third is randomly selecting a runway to move all its aircrafts to a different position on the same runway if this leads to an improvement. Finally, the forth neighbourhood randomly selects a runway and assigns each aircraft to another runway where it leads to an improvement. [7, pp. 90–93]

The perturbation is divided into four operations, from which one operation is randomly selected for each perturbation phase and applied the amount of times set by the perturbation strength. These perturbation operators randomly select one aircraft to be moved to a random position on the same runway, randomly selecting one aircraft to move it into a different runway, randomly selecting two different aircrafts from the same runway to swap their positions and randomly selecting two different aircrafts from different runways to swap their positions. The perturbation strength is calculated by the formula in Equation 2.1. The time varying variable is calculated by the formula in Equation 2.2, where TV_{min} and TV_{max} are the minimum and the maximum value of the time variation. $Iter$ is the current iteration and $Iter_{max}$ is the maximum number of iterations.
.

$$pt = \begin{cases} TV \times n & TV \geq 1 \\ 1 & \text{Otherwise} \end{cases} \tag{2.1}$$

$$TV = (TV_{max} - TV_{min}) \frac{Iter_{max} - Iter}{Iter_{max}} + TV_{min} \tag{2.2}$$

Exponential Monte Carlo is used as acceptance criterion which means that better solutions are always accepted while worse solutions are accepted with a probability of $R < \exp(-\Delta)$, where Δ is the difference in quality of the final solution compared to the initial solution and R is a random value between zero and one. [7, pp. 90–93]

2.2.2 University Timetabling Problem

Timetabling problems are a specific type of scheduling problem and defines a class of complex constrained optimisation problems of combinatorial nature. University timetabling problems originates from educational institutions that often encounter timetabling problems that manually requires a large amount of time and expensive resources. To control the complexity of the problems and to provide automated support for human timetables, much research in this area has been invested over the years. University timetabling problem can further be categorised into two groups: examination timetabling and course timetabling.

2.2.2.1 Examination timetabling

Examination timetabling is the procedure of scheduling exams into available rooms. Given is a set of examinations, a set of timeslots, a set of students, and a set of student registrations to examinations; the problem is stated as the scheduling the examinations, avoiding overlap of examinations of courses having common students, and spreading the examinations for the students as much as possible. This is usually defined as a set of constraints, varying between institutions. One common hard constraint is that no student is required to sit more than one examination in the same timeslot. [8, pp. 11–14]

2.2.2.2 Course timetabling

Course timetabling is the process of assigning students and professors to courses which are in turn assigned to rooms and timeslots such that no overlap occurs. Depending on institution, the course timetables might have to satisfy various constraints including free hours, professor's preferences, student preferences, etc. In [9] the objective is to generate a schedule such that the time assignment for each course and each professor is in accordance with the preferences provided for them. Each course and each professor assign a unique preference value to morning, afternoon, and evening. It is also possible that a professor or a course may have no preference for time of day, in which case all three times of day, morning, afternoon, and evening, are given a value of no preference.

The proposed method to solve this problem is using a hybrid genetic algorithm, which is a metaheuristic algorithm that works by creating a population of different solutions where the best solutions may contribute to the next generation of solutions. The algorithm iterates over a fixed amount of generations or until some terminate condition is met. Since only the best solutions is selected the algorithm mutate each solution between the generations to ensure that new solutions are introduced.

2.2.2.3 UniTime

UniTime is a server-client application that among other things performs course and examination timetabling [10]. It also includes a library for constraint solving containing a framework for modelling problems containing variables, values and constraints. The framework is built upon an Iterative forward search which in contrast to local search does accept incomplete solutions, meaning that it does not break any hard constraints, but might leave some variables unassigned [11]. This makes the algorithm possible to pause at any given moment and particularly suitable for visualisations.

2.3 Iterated local search

Iterated local search, defined in Algorithm 1, is a metaheuristic algorithm that revolves around applying a given embedded heuristic throughout a series of iterations. This embedded heuristic could for example be a local search or any problem specific optimisation algorithm. In order to induce the embedded heuristic to discover various neighbourhoods and thus not get stuck in a local optimum, a sub-algorithm named perturbation is used. Perturbation alters solutions to become part of another neighbourhood such that the embedded heuristic can find a new local optimum. The magnitude of which the perturbation affects the solutions is set by a so called perturbation strength that might vary over the iterations, in order to for example allow the algorithm to explore various neighbourhoods in the earlier iterations, while later on in the run stabilise around a tolerable local optimum and thus minimise the risk of ending up with a bad final result.

The acceptance criterion decides at the end of each iteration, whether to keep the solution produced by this iteration or revert into the previous one. Since the embedded heuristic do not create solutions, but rather optimising already existing ones, an initial solution is created by a separate algorithm in the beginning of the run. This initial solution algorithm can for example be a greedy heuristic or a heuristic creating randomised solutions, where the choice of strategy depends on the rest of the Iterated local search.

Compared to other metaheuristic algorithms ILS is on a conceptual level relatively comprehensible, while still offering a certain level of modularity. With this in mind, the Iterated local search still performs well compared to other metaheuristic algorithms when being applied on the University timetabling problem described in subsection 2.2.2. [12] The four components that constitutes Iterated local search namely initial solution, embedded heuristic, perturbation and acceptance criterion are further described in the sections below.

Algorithm 1 General Iterated local search

```
1: function IteratedLocalSearch(S_0)
2:     S ← InitialSolution(S_0)
3:     EmbeddedHeuristic(S)
4:     while ConditionNotMet(S) do
5:         S' ← Pertubation(S)
6:         EmbeddedHeuristic(S')
7:         S ← AcceptanceCriterion(S', S)
8:     end while
9:     return S
10: end function
```

2.3.1 Initial Solution

The initial solution provides the embedded heuristic with a solution that is used as a starting point. This enables the embedded heuristic to focus on improving solutions rather than creating them, which often time is processes with different characteristics. For runs with a larger number of iterations the initial solution seems to have less importance compared to runs with fewer iterations, which should be factors when designing an ILS.

An initial solution is usually produced either randomly or by a greedy construction heuristic. The quality of the initial solution can affect the number of improvement steps needed for a solution to be accepted as the final solution as well as the quality of this final solution. Using a greedy approach for the initial solution when combined with a local search generally provides a better final solution than having the initial solution selected at random. [4, p. 371]

2.3.2 Embedded Heuristic

Despite the name Iterated local search, a variety of different heuristic algorithms that searches for a better solution in the local neighbourhood can be used as embedded heuristic and not just local search. Even other metaheuristics can be used as embedded heuristic, where neighbourhood-based metaheuristics are one example. Neighbourhood-based metaheuristics are an extension of Iterative improvement algorithms that avoids getting stuck in locally optimal solutions by allowing movement to worse solutions in the neighbourhood of the current solution. Without limitations these metaheuristics can potentially run forever, so it is necessary to limit their run time if they are to be embedded in an ILS.

As the performance of ILS thereby is dependent on the embedded heuristic, it is crucial that this embedded heuristic is both fit and optimised for the specific charac-

teristics of the given problem. This means that the ILS does not necessarily benefit from having a stronger embedded heuristic. For example, when the computation time is limited there are possible gains of using a faster embedded heuristic more frequently instead of prioritising accuracy. [4, p. 380]

2.3.3 Perturbation

The purpose of perturbation is to prevent the ILS from getting stuck at locally optimal solutions that are considerably worse than the global optimum. Overall the embedded heuristic should not be able to undo the perturbation, as this would drag the entire algorithm back to a previously visited optimum. In order to prevent the perturbation from being revoked it needs to provide solutions that are not directly reachable by the embedded heuristic. The perturbations magnitude should however not be unlimited as this make it less likely for better solutions to be found. In addition of having a balanced strength, the perturbation should preferably be suitable for the specific problem and well synchronised with the embedded heuristic. [4, p. 372]

Perturbation strength decides the magnitude of the perturbation, which weights the width of the search space against the magnitude of exploitation within the neighbourhoods of these search spaces. One suggested instance of a time variant perturbation strength starts with an initial value that gradually decreases over the iterations, represented by a time varying variable. The time varying variable are calculated in equation 2.2. [7, p. 92]

The perturbation strength usually affects the embedded heuristics time complexity as stronger perturbation tends to render more work for the embedded heuristic. It could thereby be relevant to have lesser perturbation strength for problems that are considered easy. [4, pp. 372–377]

For problems where the best solutions are in a tight neighbourhood the embedded heuristic should focus on this limited solution space without too strong interaction from the perturbation. In opposite, when the best solutions are spread such that they are not close to the global optimum it could be useful to have a stronger perturbation. [4, p. 382]

2.3.4 Acceptance Criterion

The Acceptance Criterion decides whether a solution produced by the embedded heuristic and perturbation should be accepted as the new current solution or not. It balances how much the algorithm should focus on accepting better solutions or permit worse solutions that could lead to better solutions over time. The best approach

of an acceptance criterion does thereby depend on both the choices of perturbation and embedded heuristic. [4, p. 381]

One example of acceptance criterion is the large-step Markov chain (Equation 2.3) that accepts improved solutions, as well as having a probability of accepting worse solutions. Since the possibility to turn a deterioration of the solution into a long-term improvement is usually higher for the earlier iterations (as the number of possibilities to improve is greater), the probability of accepting worse solutions are decreased as the number of iterations increase. This decrease of probability is defined in equation 2.3, where T represents the temperature which is a function that change throughout the run and thus affects the probability of accepting new solutions. [4, pp. 377–378] A low temperature renders lower probability of accepting new solutions, while a high temperature gives a high probability.

$$LSMC(S, S^*) = \begin{cases} S^* & \text{if } P(S^*) < P(S) \text{ or } \exp\left[(P(S) - P(S^*))/T\right] > R, \\ & \text{where } R \text{ is a random variable between } [0, 1] \\ S & \text{Otherwise} \end{cases} \tag{2.3}$$

2.4 Strategy pattern

In the context of object-oriented programming design patterns can be described as classes and objects that communicate in order to solve a recurring design problem. Even though the amount of design pattern is countless, there are some things that they all have in common. [13, pp. 12–14]

Strategy pattern is categorised as a behavioural software design pattern and the purpose of the strategy pattern is to enclose algorithms and their context into Strategy objects. Interfaces for these Strategy are designed to be general enough to support various algorithms so that the interface or algorithmic context does not need to be modified when new algorithms are added. [13, pp. 55–56]

3
Method

As the purpose of this thesis is to develop an application framework containing an algorithm to solve the project-matching problem, it is crucial to obtain representative instances of the project matching problem to both concretise the problem as well as to provide the framework with input data. These instances are created in the form of two predefined test schedules, one that is easier (Appendix B) while the other is more complex (Appendix A). In order to justify the results these test schedules have at least one known solution each with no other possible solution that is better, in order to offer a suggestion of a perfect solution as reference. The way this is performed is that they meet all the constraints and thus do not generate any penalty, while at least most of the other solutions do. When the algorithms penalty approaches zero it is thereby safe to assume that a feasible solution are being produced, which makes it possible to measure the algorithms performance regarding the quality of its results.

The core of the application framework is based to handle these representative instances of the project matching problem and uses an implementation of a problem specific ILS algorithm to solve these instances. The produced results will be measured on how well the constraints are met for different number of iterations with different time-varying perturbation values and on how much execution time that these runs demand.

In the section Problem definition, the concept of constraints is introduced and further described together with a more formal mathematical description of the optimisation problem. The section Iterated local search suggests an algorithm to solve this problem. In the succeeding section named Requirements and Specifications the algorithms performance and their solutions quality are addressed. The selection Overview of the application framework introduces the actual implementation of the application framework and explains how the components are linked together.

3.1 Problem definition

As the project-matching problem is essentially constituted by matching workers with projects which are in turn assigned to workers timeslots within a timetable respecting certain criterion's, a series of problem specific constraints has been set up. These constraints comprise the worker-based properties of only assigning workers that are qualified. In addition, there are project and project role-based properties that en-

sure that only qualified workers are assigning, that each project role of a project is assigned and that projects start within their given time frame.

Since this is a combinatorial optimisation problem, we state it using a optimisation problem formulation in standard form using problem specific definition variables and functions, which is further presented and specified throughout this section.

3.1.1 Problem specific constrains

The constraints are divided into the two categories hard constraints and soft constraints. The hard constraints need to be respected for a solution to be regarded as valid, such that each iteration produce a valid solution.

Violations of soft constraints are allowed, but they cause penalties. The total penalty rendered for each soft constraint depends both on the magnitude of the violation and on the importance of this constraint compared to the other constraints, which is represented as a penalty constant for each soft constraint. This penalty is generated by a penalty function that is linear, which means that the penalty grows linearly with the violations magnitude, where for example working ten time units less than the minimum required amount generates twice as much penalty as working five time units below this limit. The hard constraints used in the problem description are defined in Table 3.1 and the soft constraints are defined in Table 3.2.

Constraint	**Description**
H_1	No worker shall have its maximum work capacity exceeded.
H_2	Only one instance of each project exists at the same time.
H_3	Project capacity is of fixed sizes and must be filled to be running.
H_4	A worker must have all attributes required for a certain project role.
H_5	A project can only be scheduled within its given time interval.

Table 3.1: Hard Constraints

Constraint	Description
S_1	Workers should not be assigned to less work than their minimum required amount.
S_2	Workers should be assigned to the projects that they prefer the most.
S_3	Workers should be scheduled within their preferred number of simultaneous active projects roles.

Table 3.2: Soft Constraints

3.1.2 Definitions and variables

In order to both define and adequately solve the problem of this thesis a series of decision variables, constants and functions are used. The constants are set before the algorithm starts, while the decision variables are assigned values during the run in order to find optimal solutions. These constants are defined below in Table 3.3, followed by the decision variables in Table 3.4. The functions are stated as Figure 3.1.

Variable	**Description**
Q	Number of projects.
N	Number of project roles.
M	Number of workers.
T	Number of timeslots.
$\hat{V}$	Maximum preference value.
$\mathbb{P}_1$	Penalty constant for assigning less work then tolerance level.
$\mathbb{P}_2$	Penalty constant for preference unsatisfactory.
$\mathbb{P}_3$	Penalty constant for violating the preferred number of projects.

Table 3.3: Constants for a given problem instance.

Variable	Description
x_i	The assigned timeslot of project i.
p_{mt}	Number of simultaneous projects roles worker m is assigned to on timeslot t.
b_i	Number of project roles that constitutes project i.
e_i	Earliest allowed start time for project i.
l_i	Latest allowed start time for project i.
c_n	Work capacity needed per timeslot for project role n.
$\hat{c}_m$	Maximum work capacity per timeslot for worker m.
$\check{c}_m$	Minimum work capacity per timeslot for worker m.
p'_m	Preferred number of projects roles worker m want to work on simultaneously.
v_{nm}	Preference value for project role n in worker m.

Table 3.4: Decision variables for a given problem instance.

$$\tau_i = \begin{cases} 1 & \text{Project is either full or empty} \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

$$a_{ir} = \begin{cases} 1 & \text{Worker } r \text{ is assigned to project role } i \\ 0 & \text{Otherwise} \end{cases} \tag{3.2}$$

$$k_{ir} = \begin{cases} 1 & \text{Project role } i \text{ is staffed by worker } r \\ 0 & \text{Otherwise} \end{cases} \tag{3.3}$$

$$y_{irt} = \begin{cases} 1 & \text{Worker } r \text{ meets the requirements for project role } i \text{ at timeslot } t \\ 0 & \text{Otherwise} \end{cases} \tag{3.4}$$

$$d(a,b) = \begin{cases} a \text{ divided by } b & \text{if } b > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{3.5}$$

Figure 3.1: Functions

3.1.3 Standard formulation of the problem

The standard formulation of the project matching system has the purpose of stating the problem of assigning project roles to workers, while respecting given constrains where the objective is to minimise the penalty produced by a penalty function. In Table 3.5 a brief description of the equations given in the standard formulation of Figure 3.2 is given and Figure 3.2 states the problem formulation in standard form. Each constraint violation is quantified, multiplied with the respective penalty constant and summarised in equation 3.6a, followed by all the subject expressions.

Equation	Description
3.6a	The penalty-function that should be minimised.
3.6b	Ensures that every project is either plenary or empty.
3.6c	Ensures that the assigned timeslot x_i is within the interval $[e_i, l_i]$, where e_i is earliest start time and l_i is latest start time.
3.6d	Ensures that the number of projects does not exceed the number of project roles. This makes sure that every project has at least one project role.
3.6e	Ensures that every project role is assigned to at most one worker and that this worker meets the requirements.
3.6f	The total work amount required of worker r at timeslot t.
3.6g	The total preference value for all the assigned work of worker r at timeslot t.
3.6h	The penalty given if the amount of assigned work is below the minimum required work capacity of worker r.
3.6i	Ensures that the amount of assigned work does not exceed the maximum work capacity of worker r.

Table 3.5: Description of equations in Figure 3.2

The penalty function to be minimised is defined in equation 3.6a, which maps over all workers timeslots and summarises each penalty bringing component into the total penalty. Each component corresponds to a soft constraint and is provided with a penalty constant used to decide its relative importance compared to the other components within the optimisation. The first component corresponds to soft constraint S_1 and has the purpose of rendering penalty when the assigned workload of a worker is lower than the defined minimum limit, which is stated by equation 3.6h. Workload above the defined minimum limit renders zero penalty, while a decrease in workload below this limit linearly increases the penalty.

The second component corresponds to soft constraint S_2, which is used to linearly add penalty depending on the workers preference value towards the assigned project role. This component is represented by the equations 3.6f, 3.6g and 3.6i, where equation 3.6g is a given timeslots total workload multiplied with the corresponding preference values, while equation 3.6f is solely the total workload of that timeslot. Function 3.5 in Figure 3.1 which divides equation 3.6g with equation 3.6f over all timeslots in the penalty function in equation 3.6a gives the overall preference value weighted by the project roles workload. Equation 3.6i re-ensures that the sum of all workloads of a given timeslot do not exceed the maximum work capacity of a worker, i.e. hard constraint H_1.

The third component is used to linearly render more penalty the further the number of simultaneous project roles assigned during one timeslot are from the workers preferred number, i.e. it corresponds to soft constraint S_3. This difference is represented as the deviation from the ideal value divided by the ideal value.

In order to keep projects within their time frame, equation 3.6c restricts the projects

$$\min \quad \sum_{m=1}^{M}\sum_{t=1}^{T}\left(\Delta_{mt}\mathbb{P}_1 + \left(\hat{V} - d(\Phi_{mt}, \Omega_{mt})\right)\mathbb{P}_2 + \left(\frac{|p'_m - p_{mt}|}{p'_m}\right)\mathbb{P}_3\right) \tag{3.6a}$$

s.t.

$$\prod_{i=1}^{Q}\tau_i = 1, \tag{3.6b}$$

$$e_i \leq x_i \leq l_i \qquad i = 1, 2, \ldots, Q, \tag{3.6c}$$

$$1 \leq b_i \qquad i = 1, 2, \ldots, Q, \tag{3.6d}$$

$$0 \leq \sum_{m=1}^{M} a_{nm}k_{nm} \leq 1 \qquad n = 1, 2, \ldots, N, \tag{3.6e}$$

$$\Omega_{mt} = \sum_{i=1}^{N} c_i y_{imt} \qquad m = 1, 2, \ldots, M, \quad t = 1, 2, \ldots, T, \tag{3.6f}$$

$$\Phi_{mt} = \sum_{i=1}^{N} c_i y_{imt} v_{im} \qquad m = 1, 2, \ldots, M, \quad t = 1, 2, \ldots, T, \tag{3.6g}$$

$$\Delta_{mt} = \max(0, \check{c}_m - \Omega_{mt}) \quad m = 1, 2, \ldots, M, \quad t = 1, 2, \ldots, T, \tag{3.6h}$$

$$\Omega_{mt} \leq \hat{c}_m \qquad m = 1, 2, \ldots, M, \quad t = 1, 2, \ldots, T \tag{3.6i}$$

Figure 3.2: Project-Matching Optimisation

start time to be within a given interval. A minimum of one project role must be associated to a project in order to not be considered void, which is ensured by equation 3.6d. Equation 3.6e makes sure that a project role cannot be assigned to multiple workers by checking that each project role are assigned to at most one worker. For a project to be considered active all project roles must be assigned, which is ensured by Equation 3.6b. Since function 3.1 in Figure 3.1 return zero for projects that are neither full nor empty and otherwise returns one, the product is zero if not all projects are filled or empty.

3.2 Iterated local search

The pseudo-code of the Iterated local search is defined in Algorithm 2, where the generate initial solution is a greedy algorithm that generates an initial solution. Perturbation ensures that the embedded heuristic do not get stuck in a local optimum, using four different problem specific perturbation operators, where one of them are randomly selected in each iteration.

Embedded heuristic goes through a number of phases, where each phase improves the solution by for example filling empty projects or re-assigning workers. The acceptance criterion decides whether to keep solution produced by the current iteration, or to stay with the previous solution, which is done by always keeping better solutions produced by the current iteration, but sometimes keep worse solutions produced by the current iteration. The perturbation, embedded heuristic and acceptance criterion are iterated while the penalty is greater than zero, or until the maximum number of iterations are reached.

Algorithm 2 Problem specific ILS

```
1: function IteratedLocalSearch(S_0)
2:     S ← GenerateInitialSolution(S_0)
3:     LocalSearch(S)
4:     while ConditionNotMet(S) do
5:         S' ← Pertubation(S)
6:         LocalSearch(S')
7:         S ← LSMC(S', S)
8:     end while
9:     return S
10: end function
```

3.2.1 Initial Solution

This paper provides a greedy approach for the initial solution. Given a schedule the list of workers is sorted based on the number of projects that they are qualified for and the list of project roles is sorted based on how many workers that are qualified for them, both in ascending order. For each worker, the list of project roles is iterated until a project role that is not already assigned is found, that the worker is qualified for and that fits the schedule. If none of the project roles fits, the worker is not assigned at all. The pseudo-code of this implementation is written below in Algorithm 3.

Algorithm 3 Generate initial solution

```
1: function GenerateInitialSolution(S)
2:     W ← S.Workers
3:     Pr ← S.ProjectRoles
4:     Sort W in descending order based on number of preferred projects
5:     Sort Pr in descending order based on number of candidates
6:     for w ∈ W do
7:         for pr ∈ Pr do
8:             if w is candidate & w prefer pr enough & w can take pr then
9:                 w ← pr
10:            end if
11:        end for
12:    end for
13: end function
```

3.2.2 Embedded Heuristic

The local search receives a possibly invalid solution from the perturbation and transform it into a valid solution while attempting to reduce its penalty. The local search is divided into a series of phases, where each phase has its own method of either improving the solution or creating opportunities for the following phase.

Phase 1

The initial phase has the purpose of reducing the difference between the target number of simultaneous project roles and the actual number. This is performed in the two different steps that are presented below.

For every timeslot of each worker in the schedule, break its matchings with the set of project roles of lowest preference value such that the amount of simultaneous project roles assigned does not exceed the preferred amount. The set of unassigned project roles are added to a taboo list, with the purpose of prohibiting them from being assigned to the same worker later in this phase.

Given a matching between a worker and a project role, replace this matching with the subsets of other project roles positioned on the same timeslots such that the worker is as close as possible to its preferred number of simultaneous projects. If several subsets get the worker equally close to the preferred number of simultaneous projects, then pick the one with the best average preference value. The project role which is left empty attempts to match with one of its candidates and the workers who lost their project roles tries to match with workers from their preference list, where each of these independent matching is based on the criterion of rendering the lowest possible penalty.

Phase 2

The second phase attempts to fill each empty project by selecting the best fits between worker and project role using the following procedure.

The project roles of each empty project are iterated to localise what timeslots the project should be placed such that its project roles have the best chance to be assigned. The project timeslot position with best compositions of candidates (highest preference value) are then decided.

Phase 3

The third phase aims to first optimise the filled projects and then to transform the schedule into a valid solution, by ensuring that all projects are either fully assigned or empty.

First the algorithm aims to move projects in order to even out workload through the schedule. This is done by analysing the amount of workload each timeslot has per column to identify the columns with low workload in order to move nearby projects to them.

The next step is to switch candidates for project roles that are already assigned within a filled project. This by selecting a project role and one of its candidates that is more suitable than the currently assigned. If the candidate has other project roles that blocks a direct switch of candidates, then the algorithm breaks those matchings and assigns them to other workers using this same procedure. In order to prevent the algorithm from getting stuck in an endless continuum of recursions, a taboo list for candidates already used by this process and a certain recursion depth is set. If a match of switching candidates have been found and if the calculated penalty is lower than before the swap is made.

The last procedure of phase 3 is to either assign all project roles or break all the matchings between project roles and workers for projects that are not filled. For each non-assigned project role of every not filled and not empty project, make a matching with the candidate with highest preference value if it has enough work capacity for all the project roles timeslots and if it is not placed on the taboo list, otherwise try with the next candidate. If all project roles of a project get a matching then keep the matching, otherwise discard them and kick all workers from the project.

3.2.3 Acceptance Criterion

The acceptance criterion decides whether to keep the changes to the schedule done by perturbation and local search in the current iteration or not. The acceptance criterion used for this algorithm is the Large-step Markov Chain which is described in Equation 2.3, explained in subsection 2.3.4 and implemented in pseudo-code in Algorithm 4. Temperature function used in Algorithm 4 is defined in Algorithm 5.

Algorithm 4 Large-Step Markov Chain

function LSMC(S', S)
  $T \leftarrow$ TEMPERATURE()
  $F \leftarrow$ PENALTY(S)
  $F' \leftarrow$ PENALTY(S')
  if $F > F'$ **then**
    $P \leftarrow \exp((F - F')/T)$
    $\Omega \leftarrow \begin{cases} S' & \text{with probability: } P \\ S & \text{with probability: } (1 - P) \end{cases}$
  else
    $\Omega \leftarrow S'$
  end if
  return Ω
end function

As temperature in Algorithm 5 decreases the probability of accepting worse solutions over time specified in the Algorithm 4, the temperature itself must decrease over time such that new neighbourhoods can be explored in the earlier iterations and safe neighbourhoods can be exploited in later iterations. This temperature decreases linearly as the number of iterations increase, where the maximum value is the temperature constant and minimum is 0.

Algorithm 5 Temperature function

function TEMPERATURE
  $d \leftarrow (1 -$ CURRENTITERATION/MAXITERATION)
  $result \leftarrow d \times$ TEMPERATURECONSTANT
  if $result < 1$ **then**
    return 1
  else
    return $result$
  end if
end function

3.2.4 Perturbation

The perturbation is divided into four different operators, where each iteration randomly selects one of them. The perturbation strength is individually determined in each operator where its magnitude is based from a TV value calculated as equation 2.2.

The first operator aims to break matchings between workers and their project roles in order to enable new matchings to be made by the local search. It forces a random project role that have more than one candidate to kick its assigned worker. The number of project roles that are selected is set by the perturbation strength which itself is based on the total number of project roles in the schedule multiplied by the TV value.

The second operator tries to replace a matching between a worker and a project role with a subset of project roles that fits in order to achieve a number of simultaneous projects that is closer to the ideal number. A random project role that is assigned is selected, as well as the set of project roles that can be assigned in this position. The subset of project roles that can replace the assigned project role which renders the lowest penalty is then assigned to the worker. The perturbation strength is based on the total number of project roles in the schedule multiplied by the TV value.

Perturbation operator number three forces empty projects into the schedule. It decides the most suitable start time of the project by measuring the lowest area of workload in defined project range and then selects the candidates best suited for the project roles. If the candidates are unable to be assigned to the project role because of too much assigned work, then discard matchings with project roles with the lowest preference value in range of the project such that the project role can be assigned. The strength of this operation is based on how many projects that are empty in the beginning of the current iteration.

The last operator kicks all assigned workers from a number of arbitrary projects, where this number depends on the number of projects in the schedule multiplied by the TV value.

3.3 Requirements and Specifications

The algorithms developed need to be able to produce qualitative solutions, which means solutions with as low penalty as possible. In order to reach these solutions, the algorithm needs to be efficient enough in regard of both time and space complexity. As the input sets might be large, the algorithms need to be able to handle growing input sizes in an adequate manner.

The components of the framework should not share more than necessary between each other. The framework has a predesigned data-flow between components but should be open to extensions and new implementations.

This application framework is implemented using C# in the .Net framework and runs with windows as platform.

3.4 Overview of the application framework

The framework is designed to follow a hexagonal architecture, also known as ports-and-adapters architecture (since it uses the abstractions of so called ports and adapters). The purpose of this architecture is to decouple the application's core logic from its service. This allows different services to act as adapters that can be plugged into the port of the applications core logic, which enables the application to be ran without these services.

In Figure 3.3 the application core logic is represented by the domain where all domain specifics such as schedule, worker, project and project role are defined. The main core application logic is the hard constraints (defined in subsection 3.1.1), which the data structures in the domain is designed to follow, while the soft constraints are considered to be the algorithms responsibility. The Algorithm is from the domains perspective merely a function that takes a schedule and returns another schedule, which constitutes the first port of the domain. In section 3.2 the determined algorithm to use is Iterated local search, hence the adapter is implemented to use this algorithm with the domain specific models.

In order to access and store the domain models from the client-side both an API adapter as well as a repository adapter is used. For the application framework a simple console application is used for setup and to run the domain, simulating the client-side usage. The repository adapters responsibility is solely to save and load single specific schedules. The communication between the adapter and the port is immutable such that the two areas are kept separate, in order to not allow any service to corrupt domain specific data or vice versa.

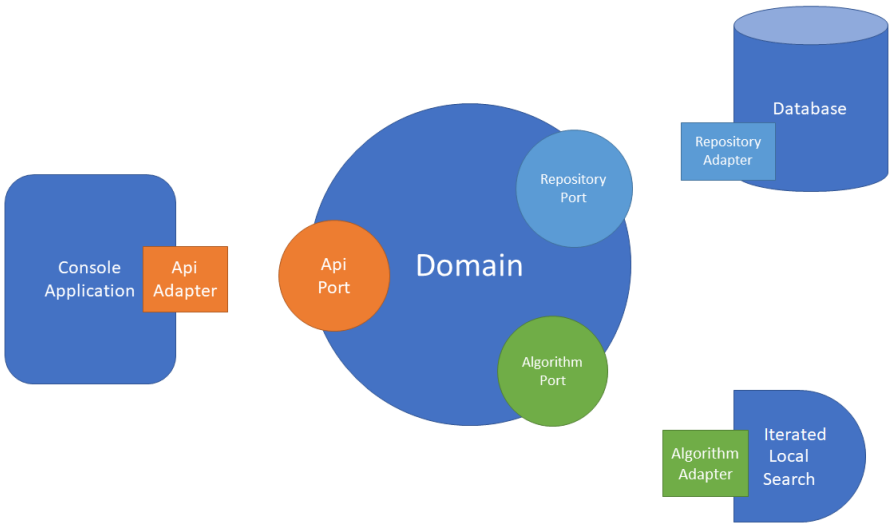

Figure 3.3: Overview of framework architecture

3.4.1 Domain

The domain layer is the central layer containing pure functional objects only. All impure functions used should be abstract towards the adapters and not be directly implemented in the domain.

The object most exposed to adapters is the schedule data structure which can be viewed in Figure 3.4, consisting of a set of projects, a set of workers and a set of project roles. Projects are parents to a set of project roles, where each project role is connected with both an assigned worker and a set of candidates. Each worker controls the timeslots from a timeslot handler where several timeslots constitutes a time interval. Every worker has a set of preferences using project roles as identifier and a value to set a unique preference.

The schedule has an immutable object builder class which is a light data structure containing the core values only in order to replicate the schedule and to send data to an adapter. The resulting data structure contains a build method that creates a new schedule identical to the original schedule.

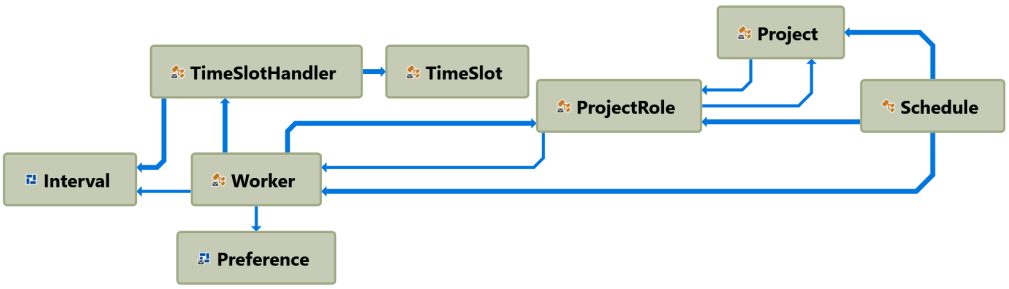

Figure 3.4: Implementation of the relational based data structures.

3.4.2 Api adapter

The api adapter is used by the clients to communicate with the framework, which means that all client usages of the domain are requested through the api port. The implemented api adapter uses a simple console application as front end, which handles input and output via the terminal window. The api adapter makes use of a finite state machine to both navigate within and use the functions exposed from the domain.

3.4.3 Algorithm adapter

The algorithm adapter uses its own domain to process the schedule data with the purpose of processing the entities from the domain layer to return a configured schedule. One of the domain entities is penalty, which is used to guide the algorithm within the solution space, where low penalty is closer to the desired outcome determined by defined conditions. The penalties are modelled based on the standard formulation defined in subsection 3.1.3. The data structure of the penalty, stated in Code 1, is implemented using the strategy pattern (explained in subsection 2.4) to abstract the soft constraints that may change over time.

Code 1 Penalty strategy

```
public interface IPenaltyStrategy
{
    double ConstraintResult(params object[] objs);
}
```

The algorithm adapter is designed such that it is easy to extend. Its core is constituted by a single method that receives a schedule as input and returns a schedule as output. A pattern used for similar instances is the strategy pattern (explained in subsection 2.4), used by the algorithm adapter to implement the wanted behaviour on an abstract level, where the resulting interface is defined in Code 2.

Code 2 Algorithm strategy

```
public interface IAlgorithmStrategy
{
    Schedule Run(Schedule schedule);
}
```

The algorithm adapter uses the Iterated local search implementation that is defined in Algorithm 2. In Code 3 the run function in an abstract class of Iterated local search that inherits the algorithm strategy interface defined in Code 2. The init method in Code 2 initialises the algorithm specific variables kept between the iterations such as current iterations, max iteration and TV values.

Code 3 Iterated local search

```
public Schedule Run(Schedule s)
{
    Init();
    Schedule s0 = InitialSolution(s);
    s0 = LocalSearch(s0);

    while (!Criterion(s0))
    {
        Schedule s1 = Perturbation(s0);
        s1 = LocalSearch(s1);
        s0 = Acceptance(s0, s1);
    }
    return s0;
}
```

3.4.4 Repository adapter

Another adapter that is part of the framework is the repository, which currently is used to preserve the data between the different test runs, but would in a final application be a central cache or storage of one or multiple schedules. In this implementation the repository adapter implements a well-defined port interface in the domain and uses a database to preserve data. The schedule data structure in Figure 3.4 is highly connected and thus must have a well-designed database. The following choices of databases where considered:

- Relational database – Relational databases is a collection of data which are stored in a tabular form, based on the relational model. In Relational database, each table contain rows and columns, where rows represent records and columns represent attributes.
- Document database – Documents databases stores its data in JSON-like documents that have key–value pairs. Document databases offer a flexible data model, making it easy to store and combine data of any structure and allow dynamic modification of the schema without downtime or performance impact.
- Graph database – Graph database is a database which uses graph structures with nodes, edges and properties to represent and store data. It uses a graph model which means that relations are built between the stored objects.

Relational databases were originally designed in tabular structure, which is not suitable for modelling relationships. This because tables need to be joined together using foreign keys to connect information from different tables. [14] [15]

Document databases and Graph databases on the other hand belongs to the database-category NOSQL. NOSQL databases are optimised for handling large quantities of data in which elements are not closely related, and therefore expensive joins are not needed. These databases were all built with a focus on scalability, which means that

they all include some form of sharing or partitioning. [16]

The database used in this application framework is the document database, since it is compatible with the configurations and is well fit to load the schedules. As the application framework follows the ports and adapter design pattern, the current database could relatively easy be replaced by any other database.

3.5 Analysing

This chapter provides the tools for analysing both the correctness and the quality of the solutions produced by the algorithm.

3.5.1 Analysing correctness

A solution is considered correct if all the hard constraints in Table 3.1 are met. The frameworks hard constraints are built into the data structure of the domain such that an error is thrown in case of rule violations. This means that for any solution produced, no hard constraints are violated.

3.5.2 Analysing quality

The quality of the solution is defined by how well the three soft constraints are met. A distinction is made between how the quality of the solution result is analysed in this section and how the algorithm measures the quality of the same soft constraints, as the algorithm summaries the number of weighted constraint violation magnitudes as penalty while the solution quality is the average of each constraint violation over all workers per timeslots. The quality analysis is thereby not dependant on input size or assumed constraint importance, but instead present an analysis of each constraint independently. The quality of each soft constraint is calculated as follows in the equations below, where each equation uses variables defined in Tables 3.3, 3.4 and functions defined in Figure 3.1.

$$\sum_{m=1}^{M}\sum_{n=1}^{N}\frac{a_{nm}c_n v_{nm} length_n}{Den} \tag{3.7}$$

$$\textbf{where } Den = \sum_{m=1}^{M}\sum_{n=1}^{N} a_{nm}c_n length_n$$

$$\frac{\sum_{m=1}^{M}\sum_{t=1}^{T}\max\left(0, \check{c}_m - \sum_{n=1}^{N} c_n\right)}{T \times M} \tag{3.8}$$

$$\frac{\sum_{m=1}^{M}\sum_{t=1}^{T}\text{abs}\left(p'_m - p_{mt}\right)}{T \times M} \tag{3.9}$$

In equation 3.7 the average amount of work time less than the minimum required amount of each worker over the total amount of timeslots is given, which represents the violations against S_1.

Equation 3.8 represents the average grade of all worker-project matches over every timeslot, where this grade is weighted proportionally depending on how big quota of the total assigned work amount of a worker that each project role constitutes. This is a visualisation of the violations against soft constraint S_2.

Lastly equation 3.9 constitutes the average difference in number of assigned projects per timeslot for each worker compared to the workers preferred number of simultaneous active projects. This offers a value on how much S_3 are violated.

3.5.3 Algorithmic efficiency

Efficiency are measured by how the execution time of the algorithm varies depending on the maximum iteration limit on a given input. This measure has to be weighed against the quality of the results as the number of iterations affects the results. In order to make the time simulations less affected by other processes, the runs are performed on one single core with high threading priority. Before each run the algorithm are continuously iterated in order to give the same cache pre-condition between the run in order to even out differences in memory management and thereby improve the accuracy of time estimations.

4
Results

The results in this section are produced by running instances of the project matching problem (briefly stated as matching workers with projects which are in turn assigned to workers timeslots within a timetable while respecting certain specified constraint) in the application framework containing an Iterated local search algorithm. These instances of the project matching problem are two test schedules, where the first schedule which can be found in appendix B is considered easy and the second schedule found in appendix A is considered hard. The easy schedule is less extensive than the hard schedule as it consists of less projects, project roles and workers, hence having a lower number of possible solutions. It also has a less complex structure as all the projects have the same length, which makes it easier for the algorithm to navigate through the solution space.

The results of these runs are presented in terms of both solution quality and algorithmic efficiency. The solution quality is based on the frequency and magnitude of violations of the soft constraints. Algorithmic performance is measured on the execution time of the algorithm running each of these two test schedules using various limits of maximum iterations, which offers a suggestion of how the execution time increases as the number of iterations grows. This makes it possible to compare how runs with various max iterations corresponds to both solution quality as well execution time. The quality of results is given in section Solution quality whereas the algorithmic performance is given in section Algorithmic efficiency.
The application framework produces XML-files with the resulting schedules optimised by the algorithm, which can be analysed for single solutions. As the results of each test schedule for every parameter setting are based on the average of 100 runs however, these XML-files are not added to this report and instead the analysis of the solution quality which is an average over all the solutions is presented.

4.1 Solution quality

Below are the results of the two test schedules presented in terms of penalty, average difference in workload, average rate of assigned project role and average difference of running simultaneous projects. The results presented below are an average of 100 runs using the same input.

ILS have multiple variables that can be configured in order to optimise the algo-

rithm for a certain instance of a problem, such as the number of iterations or the maximum and minimum allowed TV value that controls how aggressive the perturbation strength is over time. These variables are highly dependent on the particular problem instance; hence each new instance of a problem should have these variables tuned to enable the best possible solutions. For the two test schedules given in this thesis the penalty constants where kept at the same values through all runs where instead the ILS specific variables varied. Even though only the ILS specific variables did vary in the runs producing the results presented in this chapter, the runs were preceded by an overall tuning of the penalty constants to provide good settings. We used several runs of both the easy and the hard schedule using multiple combinations of these ILS specific variables and choose constants that would balance how much penalty each constraint would produce. The resulting constants can be viewed in Table 4.1.

Table 4.1: Penalty constants used throughout the tests.

Constant	Value
P_1	1
P_2	10
P_3	100

Worth noticing is that the perturbation algorithm is composed by four components, where one of the components is not affected by the TV value, but is instead dependant on the number of empty projects. Three of the components uses the TV value to set its perturbation strength, where the TV value is determined by the linear function defined in Equation 2.2 that spans between the minimum TV value and the maximum TV value over the iterations. A minimum TV value of zero means that the perturbation strength has the smallest possible value at the end of the run and a maximum TV value of one means that perturbation strength is at the maximum value from the start. If the maximum TV value is greater than one, then the produced perturbation strength is still maximised until the value of the slope in the current iteration is less than one. Likewise, if the minimum TV values are less than zero then the produced perturbation strength is at it minimum when the value of the slope is less or equal to zero. We used values centred around one for the maximum TV value and values centred around zero for the minimum TV value in the runs of which the results given in the following subsections are based on.

Another variable used to control the ILS is the number of maximum allowed iterations. In general, higher numbers of iterations creates solutions of increased quality, even though this increase in quality stagnates when a large number of iterations have been reached. Thus, the results presented uses four different values of maximum iteration, which spans from what is considered a low value to a high value.

4.1.1 Average difference in workload

Table 4.2 and Table 4.3 contains the analysing equation 3.7 results after running easy schedule and hard schedule. The values in the tables represents the violations against S_1, which increases the penalty for each worker assigned to less work than their minimum required amount proportional to the amount of work deficit and the penalty functions weights. For example, a value of 15.0 means that all workers on average lack 15.0 units of work over all timeslots in order to work the ideal amount.

Table 4.2: Average difference in workload in easy schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	14.2	2.9	1.8	0.6
0.1	1	15.0	4.4	3.6	0.2
0.2	1	17.2	7.0	2.5	0.8
0.3	1	18.9	9.5	3.5	3.3
0.4	1	14.6	6.6	1.5	0.0
0.1	2	16.4	10.9	5.8	2.6
0.1	0.9	13.2	4.3	1.3	0.4
0.4	0.9	15.0	5.7	2.7	0.8
0.4	1.1	17.1	9.5	2.8	1.8

Table 4.2 shows that when the easy test schedule runs with the maximum number of iterations set to 15000, the minimum TV value set to 0.4 and the maximum TV value set to 1 the resulting value is of average difference in work amount is 0.0. This means that each worker has the ideal value of work amount over all its timeslots, i.e constraint S_1 had no violations. As all workers ideal value of work amount on every timeslot is all values between 100 and 120, the difference in work amount of 0.0 means that all workers have between 100 and 120 in work amount over all timeslots.

With a lower number of iterations such as for example 1000, the lowest achieved average difference in work amount is 13.2. This means that the average worker over all timeslots is 13.2 work units below reaching the interval of ideal values 100 to 120.

Table 4.3: Average difference in workload in hard schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	9.3	9.3	8.6	8.7
0.1	1	9.5	9.1	9.2	8.5
0.4	1	12.6	11.3	10.1	9.8
0.1	2	10.0	9.1	9.2	8.4
0.1	3	10.1	9.4	9.3	9.0
0.2	2	10.4	9.5	9.6	9.1

Table 4.3 shows that when the hard test schedule runs with the maximum number of iterations set to 15000, the minimum TV value set to 0.1 and the maximum TV value set to 2 the lowest resulting value is of average difference in work amount is achieved. This means that the average worker over all timeslots is 8.4 work units below reaching the interval of ideal values 100 to 120.

4.1.2 Average rate of projects

Table 4.4 and Table 4.5 contains the analysing equation 3.8 results after running easy schedule and hard schedule. These values represent the average grade of all worker-project matches over every timeslot, where this grade is weighted proportionally depending on how big quota of the total assigned work amount of a worker that each project role constitutes. This is a visualisation of the violations against soft constraint S_2. For example, a value of 4.0 means that each worker is assigned to project roles that they on average give the preference value 4 out of 5.

Table 4.4: Average rate of assigned project role in easy schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	4.948	4.992	4.991	4.995
0.1	1	4.947	4.983	4.984	4.998
0.2	1	4.934	4.973	4.982	4.994
0.3	1	4.924	4.952	4.982	4.984
0.4	1	4.940	4.971	4.992	5.000
0.1	2	4.931	4.948	4.969	4.984
0.1	0.9	4.945	4.983	4.996	4.996
0.4	0.9	4.938	4.976	4.988	4.996
0.4	1.1	4.929	4.958	4.985	4.992

Table 4.4 shows that when the easy test schedule runs with the maximum number of iterations set to 15000, the minimum TV value set to 0.4 and the maximum TV value set to 1 the resulting value is of average is 5. This means that all workers only have project roles assigned with the preference value 5 (which is the highest preference value for this schedule), i.e constraint S_2 has no violations, where the proportion of worker and project role matchings with the preference value set to 5 is around 15 percent for this problem instance.

Table 4.5: Average rate of assigned project role in hard schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	4.922	4.936	4.918	4.925
0.1	1	4.926	4.923	4.923	4.913
0.4	1	4.957	4.935	4.931	4.928
0.1	2	4.921	4.930	4.934	4.923
0.1	3	4.928	4.927	4.938	4.928
0.2	2	4.921	4.929	4.927	4.939

Table 4.5 shows that when the hard test schedule runs with the maximum number of iterations set to 15000, the minimum TV value set to 0.4 and the maximum TV value set to 1 the resulting value is of average is 4.957. For this problem instance around 22 percent of the potential worker project role matching have the preference value 5, around 48 percent has the value of 4, around 29 percent has the value 3, 0.5 perfect of value 2 and 0 percent has the value 1.

4.1.3 Average difference in simultaneous running project roles

Table 4.6 and Table 4.7 contains the analysing equation 3.9 results after running easy schedule and hard schedule. The resulting values constitutes the average difference in number of assigned projects per timeslot for each worker compared to the workers preferred number of simultaneous active projects. This offers a value on how much S_3 are violated. For example, a value of one means that on average all workers deviate one project role over all timeslots from their preferred number of simultaneous assigned project roles throughout the problem instance.

Table 4.6: Average difference of running simultaneous project roles in the easy schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	0.170	0.035	0.021	0.007
0.1	1	0.178	0.053	0.043	0.002
0.2	1	0.206	0.083	0.028	0.009
0.3	1	0.226	0.114	0.042	0.039
0.4	1	0.175	0.079	0.018	0.000
0.1	2	0.195	0.129	0.068	0.030
0.1	0.9	0.154	0.050	0.015	0.004
0.4	0.9	0.180	0.068	0.032	0.009
0.4	1.1	0.205	0.114	0.033	0.021

Table 4.6 shows that when the easy test schedule runs with a maximum number of iterations set to 15000, the minimum TV value set to 0.4 and the maximum TV value set to 1 the resulting value is of average is 0.0. This means that all worker only has their preferred number of simultaneous assigned project roles over all timeslots, i.e constraint S_3 has no violations.

Table 4.7: Average difference of running simultaneous project roles in hard schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	0.125	0.128	0.119	0.119
0.1	1	0.130	0.123	0.126	0.118
0.4	1	0.174	0.156	0.141	0.136
0.1	2	0.138	0.123	0.125	0.115
0.1	3	0.139	0.128	0.125	0.123
0.2	2	0.144	0.131	0.131	0.124

Table 4.7 shows that when the hard test schedule runs with a maximum number of iterations set to 15000, the minimum TV value set to 0.1 and the maximum TV value set to 2 the resulting value is of average is 0.115. This means that all worker has a deviation of 0.115 from their preferred number of simultaneous assigned project roles over all timeslots.

4.1.4 Average penalty result

In Table 4.8 and in Table 4.9 the penalty results after running easy schedule and hard schedule with penalty constants (Table 4.1) is displayed.

Table 4.8: Penalty in easy schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	1411.2	288	177.8	58.8
0.1	1	1488.8	438.8	363.2	19.6
0.2	1	1703.6	698.4	249.6	82.6
0.3	1	1871.2	941.2	347.6	326
0.4	1	1445	653.8	148.2	0
0.1	2	1619.8	1082	570.4	256
0.1	0.9	1310.4	426.8	129.2	39.2
0.4	0.9	1484.6	564	267	79.2
0.4	1.1	1692.4	942.6	276.4	178.2

Overall the penalties in Table 4.8 decreases as the number of maximum iterations increases. When the easy test schedule runs with a maximum number of iterations set to 15000, the minimum TV value set to 0.4 and the maximum TV value set to 1 the resulting penalty is 0.0. This means that none of the soft constraints are violated, as can be seen in the tables 4.2, 4.4 and 4.6.

Table 4.9: Penalty in hard schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	1748.9	1738.6	1626.9	1627.9
0.1	1	1806.5	1715.8	1716.9	1620.3
0.4	1	2336.3	2115.0	1914.6	1856.6
0.1	2	1893.2	1704.1	1709.0	1587.3
0.1	3	1905.2	1759.6	1725.4	1688.7
0.2	2	1944.9	1783.0	1795.8	1695.3

Overall the penalties in Table 4.9 decreases as the number of maximum iterations increases. When the hard test schedule runs with a maximum number of iterations set to 15000, the minimum TV value set to 0.1 and the maximum TV value set to 2 the resulting penalty is 1587.3. For the minimum TV value of 0.1 and maximum TV value of 2 the penalty steadily decreases over the increasing number of maximum iterations, due to the fact that the 3 weighted soft constraints in total decreases their produced penalty, even though single constraints temporarily increases for certain values of max iteration as can be seen in the tables 4.3, 4.5 and 4.7.

4.2 Algorithmic efficiency

The following results is measured running on a single core on highest priority using a laptop with the following hardware: A CPU Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz base clock speed and 2.90 GHz turbo clock speed, 16 GB Random Access Memory and storage of 1 TB of Solid State Disk. These time measures are based on the algorithms run only, where the count start from the beginning of the initial solution and finish when either the algorithms condition has been met or when the number of maximum iterations have been met.

Table 4.10 and Table 4.11 shows the average run time of the algorithm measured in milliseconds for the easy schedule respective hard schedule. In both of these tables, higher maximum and minimum TV values tends to result in more longer execution times as the perturbation operators become more extensive. Naturally higher maximum numbers of iterations give larger execution times, which has to be put in relation to the increased quality of the solutions that this render. Overall the later iterations of a run improve the quality of results less than the earlier iterations, hence in the end making it an issue of balancing between quality of the solutions and execution time of the algorithm.

Table 4.10: Run time (milliseconds) easy schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0.1	0.9	2347.3	11084.9	21658.7	23506.4
0.4	1	3089.8	12164.2	19602.3	34605.1

The execution time overall grows linearly with the increased number of maximum iterations, where the exceptions from this can be explained by that the runs with a higher number of maximum allowed iterations more often finds an optimal result and thereby do not run all these allowed number of iterations.

Table 4.11: Run time (milliseconds) hard schedule

TV		Number of iterations			
Min	Max	1000	5000	10000	15000
0	1	10687.4	56890.4	134627.0	202762.8
0.1	2	21161.8	110893.0	262623.9	310072.7

The execution time overall grows linearly with the increased number of maximum iterations, except for when the hard test schedule runs with the minimum TV value set to 0.1 and the maximum TV value set to 2, between maximum allowed number of iterations 10000 and 15000.

5

Discussion

5.1 Algorithm adaptations and evaluation

In order to find qualitative solutions, it is crucial to develop embedded heuristic operators that correspond to each single soft constraint effective enough. It is important to make sure that these operators are mutually proportional in strength as they primarily favour certain specific constraints. Ensuring that constraint S_3 is respected requires an operation strong enough to swap a single project role with several project roles at the same time, in order to enforce a number of simultaneous projects closer to the target while still not violating constraint H_1 due to the potential difference in required workload between single project roles. For constraint S_2 the challenge is not the overflow of workload for single workers as project roles corresponding to the number of project roles are discarded, but rather that the discarded project roles must be filled in order to not violate constraint H_3, which requires a chain of assignments as each discarded project role needed to be assigned to a new worker. Constraint S_1 need to acknowledge constrain H_1, but should also make sure to work in compliance with the constrains S_2 and S_3.

In the same way perturbation has to make sure that the algorithm does not end up with a suboptimal solution regarding any of the soft constraints, which more specifically is to avoid suboptimal matchings between project roles and workers because of the complexity of the problem structure. The main strategy in order to prevent this is to break matchings with low preference value, both on the scale of single matchings as well as for full projects. This operation alone tends however to allow the embedded heuristic to recreate recently broken matchings but compensate this by increasing the perturbation strength forces the algorithm into random restarts rather than solely avoiding getting stuck in local optimums, which means that additional perturbation operators are needed as complement. In order to avoid certain projects to ensure that certain projects do not get overlooked because they are harder to fill the perturbation assigns the workers with highest preference value to random projects. To ensure that soft constraint S_3 do not get stuck in a local minimum the perturbation forces sets of project roles to switch with single project roles. Overall setting the perturbation strange is a delicate issue, which makes it important to tune each perturbation operator's perturbation strength function individually.

5.2 Quality of result

A distinction has been made between solution quality and penalty, as the magnitude of the different penalties depend on how much each soft constraint are weighted, based on assumed importance. These weights must be tuned for each instance of the project matching problem in order to prioritise between violations of the different soft constraints. The penalty itself is a measure on how much each respectively constraint is violated, but since the penalty functions also depends on its predefined weight this is not a good measure for analysing. In addition, it does not consider that a larger number of workers or timeslots (balanced with a matching number of projects) tends to proportionally increase the amount of generated penalty, as the amount of violations naturally becomes larger. Penalty is thereby not a good measure of a solutions quality. Instead each soft constraint is measured linearly by how far from optimal they are on the basis of each timeslot for every worker.

As there is a large quantity of possible solutions of the project scheduling problem even with moderate input sizes, it is hard to compare a given solution to all possible solutions. By the same reason it is difficult to find both the best and worst solution in regards of penalty, as this would demand all solutions to be checked. The total number of legal solutions to the problem can however be calculated mathematically, but this does not provide their respective solution quality in regards with the soft constraints. Comparing with the theoretically optimal solution that meets all constraints and thus rendering the penalty 0 is not ideal either, as this might not be a possible solution of the given instance of the problem. Furthermore, the amount of solutions between a suboptimal solution produced by the algorithm and the theoretical optimal solution are not known, as well as the amount of solutions between the suboptimal solution and the theoretically worst solution. This is because the amount of solutions is not linear to the penalty of these solutions, meaning that the solutions penalties are not evenly distributed between the lowest and the highest possible penalty produced by a solution.

For the easy schedule, the TV values [0.4, 1] on 15000 iterations renders an optimal schedule regarding all the soft constraints over 100 runs. When looking at a lower number of iterations such as 1000, the TV values [0.1, 0.9] achieved better than the TV values [0.4, 1] regarding all the soft constraints. This might be due to that 0,4 as a minimum TV value is relatively high, making the neighbourhood of the solution space quite large. For 1000 iterations a large neighbourhood could cause a problem as there is not enough iterations to find a satisfactory solution within this, while the smaller solution space given by the TV values [0.1, 0.9] can focus on finding a relatively good local minimum. At 15000 iterations however, a relatively large solution space provides the better solutions, as there are enough iterations to sufficiently explore this solution space.

The hard schedule gets stuck in some local minimums such that it cannot find the global minimum. This could probably be improved by further tuning of the penalty

constants to make it find other paths in the solution space. Also, a perturbation operator more customised for this specific schedule could possibly improve this. Furthermore, the embedded heuristic could be more adjustable throughout the iteration in order to focus on satisfying the constraint that for the moment renders the most penalty.

5.3 Algorithmic efficiency

The runs with TV values [0.4, 1] rendered an overall longer run time for 1000 and 5000 iterations than for runs with the TV values [0.1, 0.9]. This due to a more extensive perturbation producing larger neighbourhoods, which further makes the embedded heuristic more extensive. For runs of 10000 and 15000 the runs with TV values [0.4, 1] are on average shorter than for runs with TV values [0.1, 0.9]. The reason for this might be that because runs with TV values [0.4, 1] to a greater extent than runs with TV values [0.1, 0.9] finds the optimal solutions, which makes it finish and thus running fewer iterations in total. The same applies to the hard schedule, except that runs with TV values [0.1, 2] has a higher execution time over all the values of maximum iterations, as the optimal schedule is never found, thus making it run all the iterations (even for the higher maximum iteration values) unlike the runs for the easy schedule.

5.4 The framework

The initial ambition was to create a framework to automatize the assignment of workers to projects in accordance with the problem description. Organisations allocate resources such as both time as well as financial means on scheduling, both in order to produce the schedules as well as the cost when these contains flaws resulting in inefficiency. Sometimes these schedules might need to be recreated since a minor change made by a worker or within the structure of a project could make the entire schedule unstable. It is hard to claim with certainty that the proposed system and framework can produce a sufficient schedule compared to manual planning, but the extensive number of possible solutions for combinatorial optimization problems such as the two test schedules given by this report provides at least some indications of this.

An example of usage for the application framework is to schedule work shifts for fast food restaurants, where each shift needs a combination of personal with certain work skills. By reducing fast food restaurants scheduling problem into the problem description defined in this thesis it can be solved by the application framework. A work shift can be viewed as a project where the set of personal is the project roles needed and the set of skills, such as cooking and service, must be fulfilled to keep the restaurant operational. Each worker has its own set of skills and can could then

choose how many simultaneous project roles they could handle. This could then be applied to a restaurant chain where each restaurant in its geographical location could issue its own projects and the set of all workers could be a candidate on their nearby restaurants and prefer the projects they want. A timeslot could be viewed as a week, where each worker has a minimum and a maximum number of hours to work. Another example of usage is to create schedules for hospital appointments, in order to maximise the usage of medical staff. Medical staff would in this case be represented by workers with skill sets corresponding to their work title and medical specialisation. The patient appointments are represented by projects, where each project role represents some medical personnel with a certain skill.

The drawback from implementing the application framework on a non-real-world problem with our problem description is the lack of available data. A more straightforward implementation of the framework could have been made with a large quantity of input data and already known solution results. Creating manageable premises for perfect schedule solutions that also catches the corner cases of the problem at hand and data structural bugs costed both implementation time and caused insecurity regarding the quality of the results.

5.5 Ethical issues

Project matching today is a complex process which involves a lot of conversation between worker and administrator, a system using our framework would ease the workload from the administrative side. A natural effect of this is that the system might take someone's job as an administrator, which might increase the unemployment rate. Employees could in addition get a feeling of lost power over their own working situation as digitalised systems now manages the projects instead. It could however also make workers more honest about whether they really want to take part in a specific project or not, as it is more comfortable to be of trouble towards a digital system than to a human.

A person's private data should not be accessible to others, since the project matching framework will contain the preferences of each worker, which could consist of sensitive information. There is of course a risk of data breach, but that risk would probably be minor compared to the amount of small talks and gossip about confidential information which occurs when humans are responsible for assigning people to projects.

The project matching framework is intended to be quite dynamic, especially with variables used within the preference list. There is no guarantee that variables used to describe or prefer a worker would take equality into account, such as race, colour, religion, gender etc. This would rather depend on the administrator. It is also important that the framework does not perform a matching that is biased, which means that it should not favour any workers over others if they have the same pre-

requisites and professional background. These issues could potentially be solved by having an algorithm performing the matching rather than a human with conscious or subconscious prejudice towards other people.

Another dynamic factor in the framework would be the project variables and information. A worker must have the correct information about the projects to be able to make a good decision. If this is not the case people might end up in projects they don't want, making them either feel misplaced or even worse working on something they oppose by for example moral, cultural or religious reasons.

Consider a big company which several projects and even more workers. The company recently bought a system that uses the project matching framework, which resulted in that a majority of the administrative personnel had to go. An effect of this system however is that the projects get nicely balanced and that the workers feel more satisfied with the projects they get assigned to. As technology in general advance it might feel like a natural process that some jobs disappear.

However, if the system seems to favour some people in the different matchings, where can they then turn to demand justice? Sometimes a fair system is not as important to people as a system that listens to them.

6
Conclusion

Given the two test schedules, the algorithm produces quite good approximate solutions which improves as the number of iterations increases within the interval 1000 to 15000 iterations. For the easy schedule, given the TV values [0.4, 1] and the maximum allowed number of iterations set as 15000, an optimal solution is produced. The algorithm does not provide an optimal solution for the hard schedule, due to it getting stuck at various local minimums.

The hexagonal architecture allows the framework to be modular and extendable, even though it is currently a proof of concept rather than a final product. The domain and algorithm adapter provide a more finite implementation while api adapter and repository adapter is a minimum viable implementation.

Iterated local search can be applied to a problem such as the project matching problem introduced in this thesis. Implementing problem specific embedded heuristics and perturbation operators as well as setting proper TV values and penalty constants are all vital parts in producing adequate solutions. Finally, providing a clear definition of solution quality is crucial in order to evaluate the algorithm performance.

7
Future work

The application framework has potential to create real world application systems where each system can carry their own set of soft constraints. The domain does however not possess this flexibility towards the hard constraints, as the data structures are built around the constraints. Using a method that validate if an action is correct in each single data structure provides the opportunity to define a rule set in terms of hard constraints.

The algorithm makes use of each workers preferences in order to weight solution results, which means that each worker must activity rate new project roles as well as update their preference values towards already rated project roles if their preference change over time. A recommendation system used to analyse the preference history could propose a set of projects that may meet the workers preferences in order to make the rating process easier.

The current penalty constants are set as a compromise between what works best for the two test schedules given in this thesis. In order to make the algorithm more generic it might be good to set the penalty constants based on more schedules and possibly construct an algorithm to set the penalties based on test runs on a set of test schedules.

It could also be applicable to enable the system to continuously receive new projects and workers over time. One way to achieve this functionality could be to lock already started projects in the schedule and not allow the algorithm to move them or change the assigned workers, while regularly performing new runs as new projects and workers gets added as input.

Bibliography

[1] Jon Lee. *A first course in combinatorial optimization*. English. illustrat. Cambridge, UK;New York; Cambridge University Press, 2004. ISBN: 9780521010122;0521010128;9780511187831;0521811511;9780511187834;

[2] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. English. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 3642165443;9783642165443;97836421

[3] David Pisinger and Stefan Røpke. *Large Neighborhood Search*. Ed. by Michel Gendreau. 2nd ed. Springer, 2010, pp. 399–420. ISBN: 978-1-4419-1663-1.

[4] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics*. English. 2nd;2; vol. 1460. New York: Springer, 2010;2014; ISBN: 1441916636;9781441916631;

[5] Vijay V. Vazirani. *Approximation Algorithms*. English. 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. ISBN: 3642084699;3662045656;9783662045657;

[6] Teofilo F Gonzalez. *Handbook of approximation algorithms and metaheuristics*. CRC Press, 2007.

[7] Nasser R. Sabar and Graham Kendall. "An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem". In: *ASAP Research Group, Nottingham, United Kingdom* (2015). URL: http://www.sciencedirect.com/science/article/pii/S0305048315000523.

[8] Salwani Abdullah. "Heuristic approaches for university timetabling problems". PhD thesis. University of Nottingham Nottingham, 2006.

[9] Siddharth Dahiya. "Course Scheduling with Preference Optimization". PhD thesis. The Pennsylvania State University, 2015.

[10] Apereo. "UniTime". In: (2017). URL: http://www.unitime.org.

[11] Tomáš Müller. "Constraint-based Timetabling". In: *Ph.D. Thesis, Prague, Czech Republic* (2005). URL: http://www.unitime.org/papers/phd05.pdf.

[12] Amin Hadidi. "A survey of approaches for university course timetabling problem". In: (July 2015).

[13] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. English. Reading, Mass: Addison-Wesley, 1995. ISBN: 0201633612;9780201633610;

[14] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 26.1 (Jan. 1983), pp. 64–69. ISSN: 0001-0782. URL: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/357980.358007.

[15] Surajit Medhi and Hemanta K. Baruah. "RELATIONAL DATABASE AND GRAPH DATABASE: A COMPARATIVE ANALYSIS". English. In: *Journal of Process Management. New Technologies* 5.2 (2017), pp. 1–9.

[16] Ganesh C. Deka. *NoSQL: database for storage and retrieval of data in cloud.* English. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2017. ISBN: 1498784372;9781498784375;

A

Hard schedule

This schedule is made by hand and altered to have zero penalty according to the standard formulation we defined in section 3.1.3. This schedule is made to represent a hard schedule, as a complement to Easy schedule.

A.1 Definitions

The schedule contains nine workers, with the work capacities, Preference of Simultaneous Active Project Roles (PSAPR) and skill sets seen in Table A.1. Table A.2 contains preference value of the project roles for each worker, where these preference values are paired with their corresponding project role and belongs to a worker. Table A.3 states that the schedule contains 18 projects with the length of two time-slots each, where the projects has a various number of project roles. A project is allowed to start within the interval given by Earliest and Latest Start Interval. In Table A.4 the required skills and work capacity of a worker for each of the 37 project roles are given. As each worker contains ten timeslots each, the total number of timeslots are 90 over these 37 project roles.

	Capacity			
Worker	**Min**	**Max**	**PSAPR**	**Skills**
w1	100	120	1	1, 2
w2	100	120	1	3, 4
w3	100	120	2	1, 3, 4
w4	100	120	2	2, 3, 4
w5	100	120	1	1, 3
w6	100	120	1	2, 4
w7	100	120	1	2, 3
w8	100	120	1	1, 2
w9	100	120	1	1, 4

Table A.1: Workers in test schedule

Worker	Project roles with corresponding preference value
w1	(pr_1, 2), (pr_4, 3), (pr_5, 4), (pr_6, 5), (pr_{12}, 4), (pr_{18}, 3), (pr_{19}, 4), (pr_{21}, 3), (pr_{22}, 5), (pr_{24}, 5), (pr_{25}, 4), (pr_{29}, 4), (pr_{30}, 4), (pr_{32}, 4), (pr_{33}, 4), (pr_{34}, 4), (pr_{35}, 4)
w2	(pr_3, 4), (pr_8, 3), (pr_9, 4), (pr_{10}, 5), (pr_{11}, 4), (pr_{13}, 4), (pr_{14}, 3), (pr_{15}, 4), (pr_{16}, 3), (pr_{23}, 3), (pr_{26}, 5), (pr_{26}, 5), (pr_{27}, 3), (pr_{28}, 5), (pr_{31}, 4), (pr_{36}, 4)
w3	(pr_1, 4), (pr_2, 5), (pr_3, 4), (pr_6, 4), (pr_8, 3), (pr_9, 4), (pr_{10}, 4), (pr_{11}, 5), (pr_{12}, 5), (pr_{13}, 5), (pr_{14}, 4), (pr_{15}, 5), (pr_{16}, 4), (pr_{17}, 4), (pr_{19}, 4), (pr_{22}, 3), (pr_{23}, 4), (pr_{25}, 3), (pr_{26}, 4), (pr_{27}, 5), (pr_{28}, 4), (pr_{31}, 4), (pr_{32}, 4), (pr_{33}, 3), (pr_{34}, 4), (pr_{35}, 5), (pr_{36}, 3)
w4	(pr_3, 3), (pr_4, 5), (pr_5, 4), (pr_7, 5), (pr_8, 4), (pr_9, 3), (pr_{10}, 4), (pr_{11}, 3), (pr_{13}, 4), (pr_{14}, 5), (pr_{15}, 4), (pr_{16}, 3), (pr_{18}, 3), (pr_{20}, 5), (pr_{21}, 5), (pr_{23}, 4), (pr_{24}, 4), (pr_{26}, 4), (pr_{27}, 3), (pr_{28}, 4), (pr_{29}, 3), (pr_{30}, 4), (pr_{31}, 3), (pr_{36}, 5), (pr_{37}, 3)
w5	(pr_1, 4), (pr_2, 4), (pr_6, 3), (pr_{12}, 4), (pr_{13}, 3), (pr_{16}, 5), (pr_{17}, 5), (pr_{19}, 4), (pr_{22}, 4), (pr_{25}, 3), (pr_{26}, 4), (pr_{28}, 4), (pr_{31}, 3), (pr_{32}, 4), (pr_{33}, 5), (pr_{34}, 5), (pr_{35}, 4), (pr_{36}, 3)
w6	(pr_3, 5), (pr_4, 4), (pr_5, 5), (pr_8, 5), (pr_9, 3), (pr_{14}, 4), (pr_{15}, 3), (pr_{18}, 4), (pr_{21}, 3), (pr_{23}, 4), (pr_{24}, 4), (pr_{27}, 3), (pr_{29}, 4), (pr_{30}, 3), (pr_{37}, 3)
w7	(pr_4, 3), (pr_5, 4), (pr_7, 3), (pr_{13}, 4), (pr_{16}, 4), (pr_{18}, 5), (pr_{20}, 3), (pr_{21}, 4), (pr_{24}, 3), (pr_{26}, 4), (pr_{28}, 4), (pr_{29}, 5), (pr_{30}, 4), (pr_{31}, 5), (pr_{36}, 3)
w8	(pr_1, 3), (pr_4, 3), (pr_5, 4), (pr_6, 4), (pr_{12}, 4), (pr_{18}, 4), (pr_{19}, 5), (pr_{21}, 3), (pr_{22}, 4), (pr_{24}, 4), (pr_{25}, 4), (pr_{29}, 3), (pr_{30}, 5), (pr_{32}, 5), (pr_{33}, 4), (pr_{34}, 3), (pr_{35}, 3)
w9	(pr_1, 5), (pr_3, 3), (pr_6, 3), (pr_8, 4), (pr_9, 5), (pr_{12}, 3), (pr_{14}, 4), (pr_{15}, 4), (pr_{19}, 4), (pr_{22}, 4), (pr_{23}, 5), (pr_{25}, 5), (pr_{27}, 3), (pr_{32}, 4), (pr_{33}, 3), (pr_{34}, 4), (pr_{35}, 4)

Table A.2: Workers preference towards project roles

Project	Start Interval			Project roles
	Earliest	Latest	Length	
p1	2	7	4	pr_1, pr_2, pr_3
p2	6	8	3	pr_4, pr_5
p3	1	3	2	pr_6,pr_7,pr_8,pr_9
p4	1	5	4	pr_{10}, pr_{11}
p5	1	9	2	pr_{12}
p6	6	8	3	pr_{13}, pr_{14}
p7	1	7	1	pr_{37}
p8	6	9	2	pr_{15}, pr_{16}
p9	1	4	5	pr_{17}, pr_{18}, pr_{19}, pr_{20}
p10	1	5	5	pr_{21}, pr_{22}
p11	5	8	3	pr_{23}, pr_{24}
p12	1	6	1	pr_{25}
p13	1	7	2	pr_{26}, pr_{27}
p14	4	7	4	pr_{28}, pr_{29}, pr_{30}
p15	3	8	1	pr_{31}, pr_{32}, pr_{33}
p16	3	8	2	pr_{34}
p17	1	6	3	pr_{35}
p18	6	6	2	pr_{36}

Table A.3: Projects in test schedule

Project role	**Capacity**	**Skills required**
pr_1	100	1
pr_2	60	1, 3
pr_3	100	4
pr_4	60	2
pr_5	100	2
pr_6	100	1
pr_7	60	2, 3
pr_8	100	4
pr_9	100	4
pr_{10}	100	3, 4
pr_{11}	60	3, 4
pr_{12}	60	1
pr_{13}	60	3
pr_{14}	60	4
pr_{15}	50	4
pr_{16}	100	3
pr_{17}	110	1, 3
pr_{18}	100	2
pr_{19}	110	1
pr_{20}	40	2, 3
pr_{21}	70	4
pr_{22}	100	1
pr_{23}	100	4
pr_{24}	100	2
pr_{25}	100	1
pr_{26}	100	3
pr_{27}	30	4
pr_{28}	100	3
pr_{29}	100	2
pr_{30}	100	2
pr_{31}	100	3
pr_{32}	100	1
pr_{33}	100	1
pr_{34}	100	1
pr_{35}	80	1
pr_{36}	60	3
pr_{37}	100	2, 4

Table A.4: Project roles in test schedule

A.2 Representation of given data

Below in Figure A.1 is a presentation of solution where test schedule 1 renders zero penalty, which means that each of the three soft constraints are fully met.

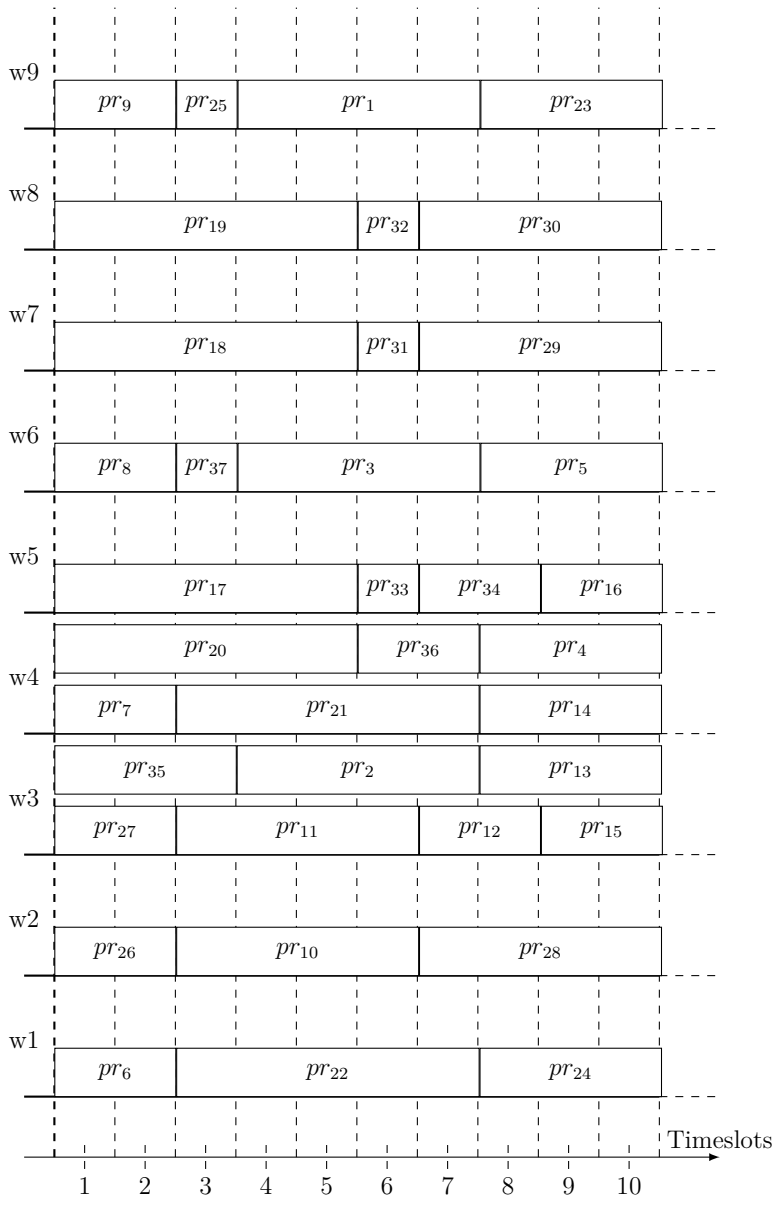

Figure A.1: Representation of a optimal valid schedule

B

Easy schedule

This schedule is made by hand and altered to have zero penalty according to the standard formulation we defined in section 3.1.3. This schedule is made to represent an easy schedule, as a complement to Hard schedule.

B.1 Definitions

The schedule contains five workers, with the work capacities, Preference of Simultaneous Active Project Roles (PSAPR) and skill sets seen in Table B.1. Table B.2 contains preference value of the project roles for each worker, where these preference values are paired with their corresponding project role and belongs to a worker. Table B.3 states that the schedule contains ten projects with the length of two timeslots each, where every project has three project roles. A project is allowed to start within the interval given by Earliest and Latest Start Interval. In Table B.4 the required skills and work capacity of a worker for each of the 60 project roles are given. As each worker contains ten timeslots each, the total number of timeslots are 50 over these 60 project roles.

	Capacity			
Worker	**Min**	**Max**	**PSAPR**	**Skills**
w1	100	120	1	1, 25, 10, 28, 13, 2, 4, 19, 8, 24
w2	100	120	1	16, 2, 4, 26, 19, 11, 8, 7, 14, 22, 3, 18, 25, 6, 10, 21, 29, 28, 15, 23
w3	100	120	1	17, 5, 20, 29, 15, 16, 27, 11, 9, 13
w4	100	120	1	3, 27, 12, 30, 23, 17, 5, 20, 7, 14
w5	100	120	1	18, 6, 21, 9, 24, 1, 26, 12, 30, 22

Table B.1: Workers in test schedule

Worker	**Project roles with corresponding preference value**
w1	(pr_1, 5), (pr_{25}, 5), (pr_{10}, 5), (pr_{28}, 5), (pr_{13}, 5), (pr_2, 3), (pr_4, 3), (pr_{19}, 3), (pr_8, 3), (pr_{24}, 3)
w2	(pr_{16}, 5), (pr_2, 5), (pr_4, 5), (pr_{26}, 5), (pr_{19}, 5), (pr_{11}, 5), (pr_8, 5), (pr_7, 5), (pr_{14}, 5), (pr_{22}, 5), (pr_3, 3), (pr_{18}, 3), (pr_{25}, 3), (pr_6, 3), (pr_{10}, 3), (pr_{21}, 3), (pr_{29}, 3), (pr_{28}, 3), (pr_{15}, 3), (pr_{23}, 3),
w3	(pr_{17}, 5), (pr_5, 5), (pr_{20}, 5), (pr_{29}, 5), (pr_{15}, 5), (pr_{16}, 3), (pr_{27}, 3), (pr_{11}, 3), (pr_9, 3), (pr_{13}, 3)
w4	(pr_3, 5), (pr_{27}, 5), (pr_{12}, 5), (pr_{30}, 5), (pr_{23}, 5), (pr_{17}, 3), (pr_5, 3), (pr_{20}, 3), (pr_7, 3), (pr_{14}, 3)
w5	(pr_{18}, 5), (pr_6, 5), (pr_{21}, 5), (pr_9, 5), (pr_{24}, 5), (pr_1, 3), (pr_{26}, 3), (pr_{12}, 3), (pr_{30}, 3), (pr_{22}, 3)

Table B.2: Workers preference towards project roles

Project	Start Interval			Project roles
	Earliest	Latest	Length	
p1	1	5	2	pr_1, pr_2, pr_3
p2	3	7	2	pr_4, pr_5, pr_6
p3	5	9	2	pr_7, pr_8, pr_9
p4	1	5	2	pr_{10}, pr_{11}, pr_{12}
p5	5	9	2	pr_{13}, pr_{14}, pr_{15}
p6	1	5	2	pr_{16}, pr_{17}, pr_{18}
p7	2	6	2	pr_{19}, pr_{20}, pr_{21}
p8	5	9	2	pr_{22}, pr_{23}, pr_{24}
p9	2	6	2	pr_{25}, pr_{26}, pr_{27}
p10	4	8	2	pr_{28}, pr_{29}, pr_{30}

Table B.3: Projects in test schedule

Project role	**Capacity**	**Skills required**
pr_1	100	1
pr_2	50	1
pr_3	100	1
pr_4	50	1
pr_5	100	1
pr_6	100	1
pr_7	50	1
pr_8	50	1
pr_9	100	1
pr_{10}	100	1
pr_{11}	50	1
pr_{12}	100	1
pr_{13}	100	1
pr_{14}	50	1
pr_{15}	100	1
pr_{16}	50	1
pr_{17}	100	1
pr_{18}	100	1
pr_{19}	50	1
pr_{20}	100	1
pr_{21}	100	1
pr_{22}	50	1
pr_{23}	100	1
pr_{24}	100	1
pr_{25}	100	1
pr_{26}	50	1
pr_{27}	100	1
pr_{28}	100	1
pr_{29}	100	1
pr_{30}	100	1

Table B.4: Project roles in test schedule

B.2 Representation of given data

Below in Figure B.1 is a presentation of solution where test schedule 2 renders zero penalty, which means that each of the three soft constraints are fully met.

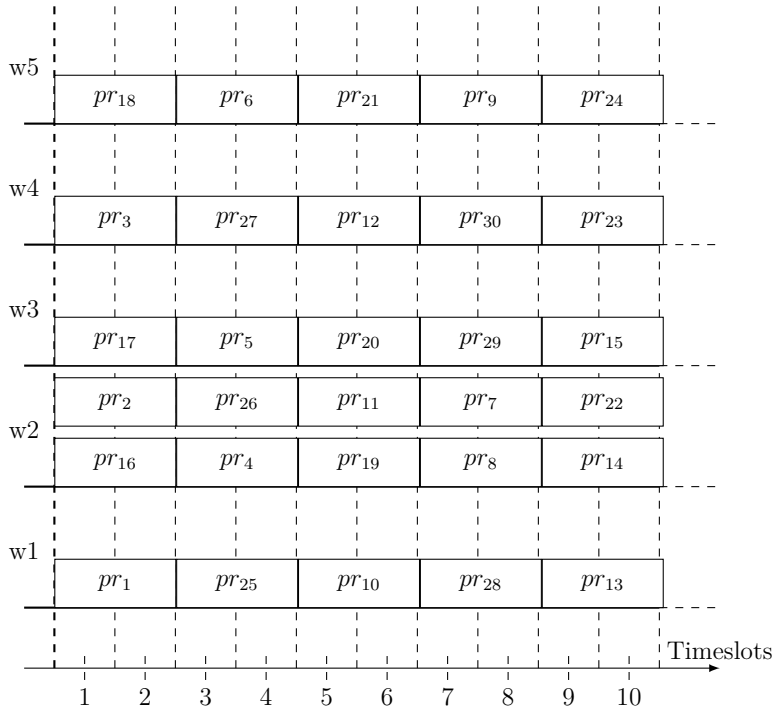

Figure B.1: Representation of a optimal valid schedule