The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) ___________________________
Summary

This project aims to describe the train crew rostering process and discuss current approaches towards optimising a base roster with the help of TRACSRoster, an application provided by Tracsis plc, which is used in industry as an optimisation tool for train crew rostering. A new approach using agent-based systems will be described and its design will be presented along with a discussion about the ways in which this could be implemented and what results it might achieve.
Acknowledgements

I would like to thank my supervisor, Dr Raymond Kwan, for his essential support and guidance throughout the course of the project. I would also like to thank my project assessor, Dr Haiko Müller, for his valuable feedback on my mid-project report and the useful advice provided during the progress meeting.

Lastly, my sincerest thanks to Dr Sarah Fores for her constant support and advice and for being more than a wonderful tutor (even when she was not officially my tutor anymore) and Jill Duggleby for her patience and support throughout my final year at university.
Contents

1 Introduction

1.1 Aim ................................................................................................................. 1
1.2 Objectives ....................................................................................................... 1
1.3 Minimum requirements .................................................................................... 1
1.4 Deliverables ...................................................................................................... 2
1.5 Possible extensions .......................................................................................... 2
1.6 Relevance to degree programme ..................................................................... 2

2 Background

2.1 Overview of the train crew scheduling and rostering processes .................... 3
2.2 Train crew rostering problem .......................................................................... 8
  2.2.1 Overview ................................................................................................... 8
  2.2.2 Current approach ....................................................................................... 11
  2.2.3 Other approaches ....................................................................................... 14
2.3 Agent-based modelling ..................................................................................... 15
  2.3.1 Overview ................................................................................................... 15
  2.3.2 Where and why is the Agent-based Model appropriate to use ............... 18
  2.3.3 Agent-based systems and the rostering problem ...................................... 20

3 Schedule & Project management

3.1 Overview ......................................................................................................... 23
3.2 Initial plan ......................................................................................................... 23
3.3 Methodology .................................................................................................... 24
3.4 Schedule ........................................................................................................... 25

4 Design & Implementation

4.1 Overview ......................................................................................................... 27
4.2 Design .............................................................................................................. 27
  4.2.1 Architecture of the agents ....................................................................... 27
  4.2.2 Behaviour ................................................................................................. 30
  4.2.3 Locking ..................................................................................................... 35
4.3 Implementation ................................................................................................ 36
  4.3.1 Overview ................................................................................................... 36
  4.3.2 Algorithms ................................................................................................. 36
5 Evaluation

5.1 Overview ............................................................................................................. 43
5.2 Evaluation plan ................................................................................................... 43
  5.2.1 Overview of the methods ............................................................................ 43
  5.2.2 Criteria ......................................................................................................... 44
5.3 Implementation analysis ..................................................................................... 45
  5.3.1 Overview ....................................................................................................... 45
  5.3.2 Likelihood of approach ............................................................................... 45

6 Conclusions

6.1 Objectives and requirements ............................................................................. 47
6.2 Possible extensions & Future work .................................................................... 48

Bibliography

A Personal Reflection

B External Contributions

C Ethics

D Gantt charts
Figures

Background

Figure 1 Stages of transport operations planning and scheduling [6] ........................................... 4

Figure 2 Sunday schedule for a train driver [5] .................................................................................. 5

Figure 3 Extract from a roster which has broken rules ....................................................................... 7

Figure 4 Extract from TRACSRoster; courtesy of Tracsis plc ............................................................ 9

Figure 5 Example of a violation of the Long weekend every five lines rule ...................................... 13

Figure 6 The relationship between an agent and the environment it interacts with, in [10], Chapter 2 ........................................................................................................................................... 16

Design & Implementation

Figure 7 Basic structure of a roster table with two links ...................................................................... 27

Figure 8 Representation of a roster table with sinks for each column ................................................. 29

Figure 11 A roster element from a violated site is transferred to the corresponding sink .............. 31

Figure 10 An incomplete validation site after sending one of its roster elements to the sink ..... 32

Figure 11 The sink sends back to the violation site a random roster element from its collection 32

Figure 12 Representation of a shift-to-shift violation site ..................................................................... 36

Figure 13 Examples of shift-to-shift violation sites within a roster table ............................................. 37

Figure 14 Pseudocode for the method that runs the roster element agent ........................................ 38

Figure 15 Pseudocode for the method that runs the potential violation site agent ............................. 38

Figure 16 Locking sinks when adding/removing roster elements .......................................................... 39

Figure 17 Pseudocode for locking a sink when getting a roster element to return to the potential violation site ........................................................................................................................................ 40

Figure 18 Pseudocode for the method accessed by potential violation site agents that want to make changes to the roster table .................................................................................................................. 41
Figures (cont.)

Appendix D

**Figure D.1** Initial project plan ......................................................................................................................... 55

**Figure D.2** Revised project plan; included in the mid-project report ................................................................. 56

**Figure D.3** Final project plan ............................................................................................................................. 57
1 Introduction

1.1 Aim

The aim of this project is to investigate an agent-based approach for crew rostering. Within the system different aspects are modelled as autonomous agents that interact with each other and whose main interest is to create a better roster solution.

1.2 Objectives

The key objectives of this project are as follows:

- Investigate the rostering process by making use of the current train crew rostering system TRACSRoster provided by Tracs plc;

- Analyse and define the current rostering model based on exploration of the TRACSRoster system;

- Describe and discuss different approaches that could be implemented in order to improve the current system;

- Describe a framework design using agent-based systems;

- Create a prototype system as a proof of concept by using the current framework;

- Design experiments to help with the evaluation of the prototype system;

- Decide if the model proposed is appropriate and technically feasible for this system.

1.3 Minimum requirements

- Analysis and definition of the current rostering model;

- Analysis of different rostering models;

- Creating an agent-based model for crew rostering;
• Prototyping some key concepts in the model created.

1.4 Deliverables

• A framework design of autonomous agents to complement the existing rostering heuristic algorithm in TRACS_Roster.

1.5 Possible extensions

Possible extensions to the minimum requirements:

• Full implementation of the current model;

• Implement the model described using JADE or a different agent-based programming language;

• Investigate how the system behaves when running in parallel and compare its efficiency to the serial running time.

1.6 Relevance to degree programme

During the course of my degree I was given the opportunity to explore many different areas within Computer Science and now that I came to the end of it, I can declare that although I believe all the modules to have been extremely interesting, I found the most appealing the more scientific and mathematics-intensive ones. I have familiarised myself with the optimisation problems during the Computational Modelling and Combinatorial Optimisation modules and also during the Practical Problem Solving module in the second year when we investigated the train crew scheduling problem. Moreover, the Artificial Intelligence module introduced some of the latest ideas in computing and how agents could be created and modelled to expose human-like behaviour. I trust the approach explored in this report is interesting not only for combining ideas from two somewhat different fields within computing, but also for using a method which has not been extensively researched before.
2 Background

2.1 Overview of the train crew scheduling and rostering processes

The scheduling and rostering processes for the train industry represent two main optimisation problems which have been intensely investigated since the 1960’s, however, besides some initial investigations in early 1990’s, there were no automatic systems in use by the UK rail industry until the late 1990’s [3]. The main cause for this would be that the scheduling problems which arise within the rail industry are far more complex than its counterparts in other public transport industries in terms of the operational rules and constraints that the schedules have to satisfy and the large number of possible combinations for assigning train drivers to work that has to be done [3].

A nice and comprehensive overview of the general rostering process is presented in [4] by Ernst et al, where it is presented as a number of modules, and depending on the application, different modules are required. The modules presented in [4] include Demand modelling – which involves deciding how many staff are needed at different times over some period and is divided into Task based demand, Flexible demand and Shift based demand; Days off scheduling – where it has to be determined how to spread rest days between work days for different lines in a roster; Shift scheduling – in which the shifts that have to be worked must be selected from a potentially large pool of candidates, along with assigning which employees will work each shift such that the demand would be met; Line of work construction – which takes into consideration the rules concerning the lines of work and the demand pattern; Task assignment – as sometimes tasks may need to be associated with particular lines of work; and Staff assignment – which involves assigning staff to particular lines of work. These modules provide a general framework within which different rostering models can be placed [4]. Also, decomposing the problem into separate modules makes it more tractable [4].

Eventually, as described in [4], there are three main factors that differentiate the rostering problems and models. The first one is represented by the degree to which rest days, working lines and task assignments are integrated – which means that the complexity of the model can differ based on the decomposition strategy. Secondly, different rostering problems need different modules. And lastly, rostering models are differentiated by “the type of demand and the fundamental unit from which lines of work are constructed” [4].
Transport operations planning and scheduling encompasses a number of typical stages [1] and the crew rostering stage is the final one within this process. *Figure 1* illustrates these stages which are paramount to enable the efficient running of the transport operations. The rostering process has a number of stages itself and starts a couple of months in advance to being used and goes on until the day of the actual operation [2]. With rostering being the final stage, it indicates that after the rostering process is completed each roster package represents the work for a specific train crew member in one week.

![Figure 1 Stages of transport operations planning and scheduling [6]](image.png)

Train crew is a term used to describe the ensemble of people working on a train, whose working hours depend on the train timetables and whose jobs are to make the running of trains possible within the timetable and offer the best of services to clients. The train crew can include train drivers, train managers, conductors, cleaning staff, catering staff, etc. depending on the policy of the train company. By far, the most complex process involves delivering schedules and rosters for drivers as they must satisfy the largest number of rules and constraints in order to be able to work on a specific train. For this reason, there will be mostly used terms such as driver scheduling or driver rostering when giving examples and clarifications.

During the crew scheduling stage, a set of timetables effective for the next period is created. The crew scheduling process creates shifts which cover all the train work on each individual day of the week [2]. The shifts created are not yet assigned to any specific crew members, however, the number of shifts represents the minimum number of crew members required to cover all the work for each day of the week [2]. It is stated that currently there is a minimum number of staff
needed to cover the work, yet many more crew members will have to be employed to be able to cover for the rest days, annual holiday leaves, sick leaves and other such situations [2].

<table>
<thead>
<tr>
<th>Driver</th>
<th>No.</th>
<th>Train No.</th>
<th>Task</th>
<th>On</th>
<th>Off</th>
<th>Hours</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>BU2388</td>
<td>HA403</td>
<td>MOB</td>
<td>Edinburgh</td>
<td>11:50</td>
<td>21:30</td>
<td>09:40</td>
<td>SUN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bathgate</td>
<td>12:38</td>
<td>12:48</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Edinburgh</td>
<td>13:14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2** Sunday schedule for a train driver [5]

In **Figure 2** above, there is presented a schedule for a train driver on a Sunday, with the sign on time at 11.50 and sign off time at 21.30. This type of diagram represents the output from the driver scheduling stage. There is no specific driver assigned to these schedules, however, they represent the work that has to be done in order to have the timetable completely covered. After the driver scheduling stage is completed, the number of schedules created represents the minimum number of drivers needed to cover all the work.

**Figure 3** depicts an extract from a base roster. There are seven weeks worth of work presented; however, not all days are working days, as within a week there are a maximum number of hours a driver can complete. “Across the week, individual crew members would need rest days and
may have to attend training events. Furthermore, provisions for sickness, annual leaves and spare coverage all add to the total work force. Therefore, the actual number of crew members employed is a multiple of the maximum number of shifts on any day of the week. For example, for train drivers this multiplier is about 2.0.” [5]
Figure 3 Extract from a roster which has broken rules
2.2 Train crew rostering problem

2.2.1 Overview

As defined in [1] and [2], train crew rostering involves packing daily driver shifts together with rest days and any other pre-planned duties in order to cover train work on each individual day of the week. The first part of the rostering process involves determining the number of staff that has the skills to meet the requirements for the services [4]. All the industry regulations associated with the workplace agreements [4] and all the rules established beforehand between the companies and the unions must be also considered during the process. As stressed in [4] by Ernst et al, "It is extremely difficult to find good solutions to these highly constrained and complex problems and even more difficult to determine optimal solutions that minimise costs, meet employee preferences, distribute shifts equitably among employees and satisfy all the workplace constraints."

For the purpose of this project, TRACSRoster – a rostering tool developed by Tracsis plc to help create better solutions in a shorter timeframe – will be used in order to provide more insight into the rostering process, the models which are currently in use and how they might be able to be improved in the future.

Firstly, the base rostering [2] process forms these packages called roster lines [2] which represent a week's worth of work for a crew member. Each such crew member takes up roster lines on a rotating weekly basis; however, the base roster is a theoretical concept as it is created on the assumption that no one would become unavailable and there would be no changes to the timetables, both of which represent issues which cannot be known in advance [2]. As presented in [2], the concepts of weekly rostering – the actual duties for specific crew members for a target week in the calendar and daily rostering – which deals with any last minute changes before the start of the operational day, both represent more explicit ways to deal with assigning crew to specific tasks but have their foundation on the base roster.

The base train crew rostering problem is represented as a two-dimensional matrix as it can be seen in Figure 4. The matrix has seven columns each corresponding to one day of the week, i.e. each row in the roster represents a sequence of daily duties and rest days for one crew member. Thus, each cell in the matrix specifies either a day's work duties or if it is a rest day. Each column in the matrix specifies all the work duties performed by the crew members for that day of the week.
As pointed out in [2], there are different kinds of duties each cell in the roster could be assigned, and some of these can be easily recognised in Figure 4:

- **Scheduled turns**, which mainly correspond to the work specified in the crew schedules;

- **Training turns (TT)**, which represent the turns used for training the staff;

- **As-required turns (A/R)**, which are only assigned a start time and an end time as these are included in the timetable to cover for crew members who become unavailable; these usually become fixed at more specific duties in the weekly rostering process [2];

- **Other activities**, such as regular health and safety training days and staff development courses [2] or route retention for drivers;

Besides duties, a cell could also be assigned to be a **Rest day (RD)**. These have to be included in the roster in order to satisfy some of the labour rules imposed, for instance, a train driver can only work a maximum number of hours in a week.

The number of **as-required turns** needed in a base roster varies between different companies and the grades of the crew but is usually in the range of 20-40% of the total number of turns that have to be covered [2]. **As-required turns** may also be converted from **rest days** as sometimes there may be an excess number of these, above the minimum required in the roster.

Every type of duty has further attributes which are used when creating the roster and trying to satisfy the rostering rules. Some such examples are presented in [2]: classifying turns into **early** and **late** turns according to their start times – one rule that might occur is not to allow a mixture of **early** and **late** turns within the same week; having a maximum number of roster cells which
are assigned duties; or having a minimum time gap between any two consecutive duties for a crew member.

The main rostering objective is to distribute the work duties in an efficient, safe and fair manner for each and every crew member [2].

It is usually the case for train crew members to rotate each week of work and take up the next line in the roster [2]. While it is easier and faster to create a large roster, if the roster becomes too large, the crew members might not complete the roster at all. This is a case when the number of lines in the roster is larger than the number of weeks the roster is valid for [2]. For this reason it might be the case that the roster is divided into a number of rotating smaller structures, referred to as links, such that the workload can be evenly distributed within themselves and the amount of work for each crew member is uniform [2].

The distribution of work duties and rest days is based upon different hard and soft rules which have different priorities when constructing the base roster. Variations of the rules exist between companies and even between crew depots of the same company [2]. For instance, depending on the grade of the crew members, there might be specific requirements for how the rest days are distributed within the roster: train drivers might prefer to concentrate their workload in some parts of the roster and have some lines left as rest days, whereas train managers often prefer having long weekends (Monday or Friday as a rest day on top of Saturday and Sunday) [2].

Rules are usually created on the basis of health and safety considerations and negotiations between crew representatives and the company's management [2]. It is usually the case that there are some hard rules, which must always be respected, along with soft rules which could be broken in some situations and under certain conditions [7]. Besides the company specific rules, there are some common restrictions which can be found in almost every train crew rostering process and are represented by the following rules:

- **Continuous driving time:** the maximum period a driver can work without having a break;

- **Rest period:** when a minimum number of hours must be left between two consecutive work duties;

- **Sign on time consistency:** the difference between the sign on time of two consecutive duties must not exceed a maximum;
• _Line duration limit_: the total number of shifts in a week must not exceed a maximum;

• _Isolated rest days_: a single rest day in between two days with work duties are to be avoided (this is usually a _soft rule_);

• _Long weekends every number of lines_: when a _long weekend_ (Friday – Sunday or Saturday – Monday) must be provided at least every number of consecutive weeks;

• _Number of rest days grouped together_: when no more than a maximum number of rest days can be grouped together;

• _Balance hours across links_: where the difference between the average number of minutes composing lines in each of the selected links should not go beyond a maximum;

• _Saturday to Sunday movement_: when the allocation of Sunday work is dictated by Saturday, i.e. if a particular Saturday is assigned to be a RD then Sunday should be a RD as well, however, if Saturday is assigned a duty, Sunday should be worked as well (is is usually considered as a _soft rule_);

With all these considered, a _base roster_ is created by trying to satisfy all the rules. Train companies usually create the _soft_ rules in order to make the rosters more robust and pleasing to the work force, however, if any soft rules are broken the roster may still be viable. Usually, the _hard_ and _soft rules_ are generally treated similarly by attaching a penalty cost to each violation. Nonetheless, it is almost never the case that all the rules, hard or soft ones, would be satisfied during the construction of the _base roster_, which leads to different optimisation methods that must be applied in order to make the roster better, i.e. trying to satisfy all the rules based on their assigned priority.

In TRACSRoster, the measure used to check the optimality of a roster is its _penalty_. Each rule has a _priority_ that differentiates the hard rules from the soft ones and a _penalty_ which determines how badly the optimality of the roster is affected when that rule is broken. The penalty of the entire roster is calculated by adding up the penalties from every instance of the broken rules within the roster. This means that the lower the penalty of the roster, the better the solution.
2.2.2 Current approach

For outlining the current approach used by TRACSRoster in optimising a roster – part of the framework of this system will be used to describe a new model – the new concepts will be presented, based on the exploration of the system.

As stated previously, the concept of penalty is used to measure the quality of a solution for both the roster itself and any rules that have to be satisfied within it.

On the roster, each rule applied has to be checked more than once depending on how many times its pattern is to be found in the roster. This introduces the concept of violation sites which represent potential locations for a rule violation. Rules might have different patterns for their violation sites and there can be identified a number of such site locations:

- **Single shift site** – each cell in the roster can represent a potential violation site; this pattern can be used for checking the validity of the duration of the shift, for example;

- **Shift to shift site** – each combination of two consecutive cells in the roster can make up a potential violation site; for instance, this pattern is used when checking if there is a valid number of hours between the sign off time of the first shift and the sign on time of the second one;

- **Line site** – where each line in the roster is considered to be a potential violation site; this pattern can be applied for the rule concerning the maximum number of hours which can be worked in a week;

- **Block of lines site** – where a block made up of a number of consecutive lines is a potential violation site;

- **Block of shifts site** – where a block of cells in the roster is a potential violation site; for instance, for any three consecutive cells in a table, checking whether there is an isolated rest day between any two cells containing work duties is a soft rule whose violation is to be avoided; this is also the case with having a day containing a work duty between two rest days which would make the staff prone to skipping that shift (claiming sickness, for instance) in order to get three rest days in a row;

- **Column in link site** – where a specific column in a link is a potential violation site;
- **Column in block of lines** – for a specific block of lines, one of the columns (a day of the week) is a potential violation site;

- **Block of long weekends** – for a number of consecutive lines, the long weekend pattern can make up a potential violation site; as an example, this pattern can be used for checking that within a number of consecutive weeks, the crew member is given a long weekend;

- **Whole link site** – where each sublink in a roster is a potential violation site itself;

- **Whole table site** – where the entire roster is a potential violation site;

For each and every rule that has to be satisfied in a roster there is a number of such potential violation sites considered. Generally, after the creation of the base roster, a rule creates a collection with all the possible potential violation sites that have to be checked to make sure the rule is not broken (all of these together can add up to thousands of violation sites within one base roster).

![Example of a violation of the “Long weekend every five lines” rule in lines 9, 10, 11 and 13; extract from TRACSRoster, courtesy of Tracsis plc.](image)

Each such potential violation site has a penalty assigned; when its penalty is zero, it means that the site is not violated and whenever the penalty is greater than zero, it means that the rule is broken for that site. For most of the potential violation sites, the pattern of cell arrangements is enough to be able to create all the possible violation sites from the beginning, which means that any changes in the work duty distribution would not affect the validity of a site. The violation sites do not move around in the roster as changes are made during the rostering process. They are created when the algorithm starts, one per potential violation, and basically sit in the background watching the part of the roster that they were initialised to watch, as the turns move around. However, there is an exception to this in the case of block rules, which is a special
case where the violation sites can be destroyed and new ones created as the block structure of each link changes.

Following the creation of the potential violation sites, the system checks if any of the sites are broken (their penalty is greater than zero), and if this is the case it sorts out the sites that violate rules in a descending order based on their penalties (finds the worst site), and then tries to solve the violation of the rule by moving the cells involved in the site violation. Thus, for each cell which is part of the violated site, it tries to swap that cell with any other cells within the same column in the roster (it is only sensible that cells in a roster can only be moved within a column as these are part of a specific day of train operations). For each and every potential swap, the penalty for the validation site is recalculated and, in the end, a swap is made if it makes the penalty of the site after the swap either zero or lower than it was before.

If there is no cell swap that would make the validation site feasible or have a lower penalty in the roster, then the site is marked as unsolvable and displayed as a rule breaking site at the end of the rostering process.

The system gives the user the freedom to swap or relocate lines in the roster, which is a useful approach usually used when optimising a roster manually, however, the optimisation approach does not make use of this feature automatically within the algorithm. Furthermore, every cell in the roster which is not a turn created from the service timetables, i.e. as-required turns, rest days or health and safety turns can be converted to a different type of turn or a rest day. This is also a very useful feature as one change in a cell can make a significant difference on the violation of sites, however, the current approach does not convert cells automatically so this is left in the hands of the person in charge of optimising the roster.

2.2.3 Other approaches

The approach used in optimising the base roster in TRACSRoster is an intelligent local search which works quite fast even for the largest data sets so it can produce better rosters in a short period of time, but does not have all the capabilities (such as the ones presented above) included in the automated process.

Two different approaches are described in [2]: an approach based on local search and another one based on a genetic algorithm. The local search approach formed the basis for what TRACSRoster is today (though many changes have been made to the initial algorithm which did not involve partitioning the roster into potential violation sites, for example).
The genetic algorithm for train crew rostering described in [2] is based on concepts of biological evolution with a population of solutions. A specialised crossover operator is designed for this algorithm such that there would be more diversity of the gene combinations in the offspring. Mutation works by changing the assignment for a selected infeasible duty cell; it is swapped with the best (according to the penalty function) of some randomly selected candidate cells from the same column (day of the week).

The results presented in [2] are based on some tests carried out using five real data sets. The results showed that local search has performed better than the genetic algorithm on all the five data sets used. The local search algorithm converged to the solution faster and the resulted roster had lower penalty costs (better solutions) than in the case of the genetic algorithm used.

Some other rostering methods and techniques are reviewed in [4] and their applicability to different rostering problems is discussed in more detail. As it is not the purpose of this paper to present all the approaches used in train crew rostering, there will be enumerated just some of the most important ones: demand modelling – which puts emphasis on the demand model used, as this has a large effect on the remaining rostering steps; artificial intelligence approaches – which include fuzzy set theory, search and expert systems; constraint programming – which is extremely useful when the problem involves many constraints and any feasible solution is enough (even if not the best); meta-heuristics – which are typically hybrids of heuristic algorithms and some examples include SA (Simulated Annealing), TS (Tabu Search), GA (Genetic Algorithms), neural networks and ant-colony optimisation; mathematical programming approaches – where the problems are formulated as LPs (linear programs) or ILPs (integer linear programs) or general mathematical programs and these generally achieve the lowest cost solution, yet there are a number of difficulties that prevent them from being applied in all the cases.

### 2.3 Agent-based modelling

#### 2.3.1 Overview

Artificial intelligence, or AI for short, is one of the newest fields in science and encompasses a wide range of subfields from the general (learning and perception) to the more specific ones such as proving mathematical theorems and diagnosing diseases [8]. Furthermore, AI tries to establish the connection between knowledge and action. As it is emphasised in [8] by Russel and Norvig, intelligence requires actions as well as reasoning, so, "only by understanding how
actions are justified can we understand how to build an agent whose actions are justifiable (or rational)” [8, p.7].

There are many ways in which agents have been defined in the literature. Jennings and Wooldridge describe it in [9] as “a computer system situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives”, and go further with trying to explain the concept better by creating an analogy between the notion of autonomy with respect to agents and encapsulation with respect to object-oriented systems. Personally, although the definition provided is emphasising the main characteristic of agents - autonomy, I believe the comparison between agents and objects is misleading in the sense that it might be grasped that agents are not objects, but something else, which is not necessarily true.

![Figure 6](image)

**Figure 6** The relationship between an agent and the environment it interacts with, in [10], Chapter 2.

Russel and Norvig present a rather more complex definition:

An agent is just something that acts (agent comes from the Latin agree, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals. [8, p.4]

As presented in Figure 6 and also described in [8], an agent is anything that can perceive its environment through sensors and act upon that environment through actuators, with the general assumption that every agent can perceive its own actions, although it doesn't necessarily know the effects.
A *rational agent* is an agent that does the right thing, i.e. takes the right actions such that it would be the most successful [8]. One way to measure the success of an agent is proposed in [8] by Russel and Norvig, and is called a *performance measure* which is used as a criteria of success of an agent’s behaviour.

The *performance measure* evaluates any given sequence of environment states, thus, "it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave" [8, p.37]. It is believed that this is the most sensible approach as defining success based on the agent states could be misleading. If we would be to describe success based on the agent’s judgment of its performance, then the agent could consider itself rational simply by tricking itself that its performance was perfect. Thus, a *rational agent* should always take an action that is expected to maximise its *performance measure*, which should be based on the environment states.

The *rationality* of an agent at any given time depends on the performance measure that defines the criteria of success, the agent’s prior knowledge of the environment, the actions it can perform and the percept sequence to date [8]. Thus, having defined the rationality of an agent, it is clearer now how to ensure that an agent can perform at its best.

Before going on with constructing agents there would be one more issue to discuss: the problems to which rational agents are the solutions, or the so called *task environments*. As presented above, the rationality of an agent is described in terms of four properties which can be grouped to form *PEAS* (*Performance, Environment, Actuators, and Sensors*) which can also be used to describe *task environments*.

The range of *task environments* within AI is vast, however, there are a fairly small number of properties which can be used to categorise these, and some of them are presented in [8] in more detail; to summarise, these are the main characteristics: *fully vs. partially observable* – do the agent’s sensors detect all aspects that are *relevant* to the choice of action?; *single agent vs. multi-agent*; *deterministic vs. stochastic* – is the next state of the environment completely determined by the current state and the action executed by the agent?; *episodic vs. sequential* – is the agent’s experience divided into single episodes in which the agent receives a percept and then performs a single action?; *static vs. dynamic* – can the environment change while an agent is deliberating?; *discrete vs. continuous* – this distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent; and finally, *known vs. unknown* – this distinction does not refer to the environment itself, but to the agent’s knowledge about the
environment: does it know the outcomes for all the actions, or does it have to learn how it works in order to make good decisions?

Based on these properties one can decide what type of task environment the agents should try to solve. It should be clearer now how to specify the architecture of an agent. The final step would be to program the agents based on the specified architecture.

**2.3.2 Where and why is the Agent-Based Model appropriate to use**

Now that the concept of agents has been explained and a range of properties which are helpful in constructing the agents have been outlined, the next question raised would be: where is it appropriate to consider developing such architectures based on agents and why would this be?

In [11], Wooldridge suggests a list of environments for which the agent-based model seems the most appropriate solution:

- When the environment is *open* or at least *highly dynamic, uncertain* or *complex*, the system needs to be capable of flexible and autonomous action.

- The agents are a *natural metaphor* for the environment (e.g. organisations, commercial environments) – some environments are naturally modelled as *societies of agents* which are either working together to achieve a goal or compete with one another.

- Environments in which the *distribution of data, control* or *expertise* implies that a centralised solution is extremely difficult to achieve, if not impossible. For instance, a distributed system of databases in which each database is under a different control could be modelled as a multi-agent system in which each database is a semi-autonomous component.

- For *legacy systems* as one problem increasingly faced by software developers is that of software that is technologically out of date but functionally essential to a company. Such software is generally hard to be discarded of because of the costs of rewriting it, yet it is required to interact with newer software components which were never imagined by the original designers. Thus, one solution would be to wrap those software components and provide them with agent functionality which would make it easier to interact with the other parts of the system.
Agent-based systems have often been suggested as an alternative to traditional heuristics techniques, as promising methods for problem domains which are distributed, complex and heterogeneous [11]. Davidsson, Persson and Holmgren compared the strengths and weaknesses of the two approaches in [14]. Their purpose was to find out if some hybrid structures which would focus on the strengths of the two approaches could return better solutions. In order to do this they established a series of characteristics which could differentiate the two methods and help them decide which one would be better for which specific problem and maybe find hybrid solutions for some problems. These are the characteristics they suggested as a common ground for the comparison between the two techniques: size (number of resources to be allocated), cost of communication, communication and computational stability, modularity, time-scale/adaptability (time between re-allocation of resources), changeability (how often the structure of the domain changes), quality of the solution (how important it is to find a good allocation), quality assurance and integrity (importance of not distributing sensitive information) [14]. According to their comparison, Davidsson et al could draw the conclusion that agent-based approaches are preferable when: the size of the problem is large, communication and computational stability is low, the time scale of the domain is short, the domain is modular in nature, the structure of the domain changes frequently and there is sensitive information that should be kept locally [14]. On the other hand, the classical optimisation techniques are to be preferred when: the cost of communication is high, the domain is monolithic in nature, the quality of the solution is very important and it is desired for the quality of the solution to be guaranteed [14]. Thus, their analysis points to the idea that the two approaches complement each other and should be combined when possible, in order to result in better solutions.

Jennings and Wooldridge present in [9] a series of industrial, commercial, medical and even leisure applications that use the agent-based technology in their implementation. Industrial applications include: process control (e.g. electricity transportation management, particle accelerator control), manufacturing and air-traffic control; commercial applications include: information management (e.g. electronic mail filtering, Internet news filtering, digital libraries), electronic commerce and business process management; the medical applications consist of patient monitoring and a health care system which would devise care programmes to treat patients and finally, leisure applications such as computer games, interactive theatres and related virtual reality applications as these systems tend to make use of semi-autonomous animated characters which can be easily implemented as agents [9].
In order to decide which would be the fields that mostly use the agent-based models, Barbati, Bruno and Genovese did a literature review in [12] in which they tried to identify the scientific literature that made use of the agent-based model to solve optimisation problems. The conclusions they reached after their research was that agent-based models seemed to be particularly suitable to tackle scheduling problems [12]. The list with the application fields presented in [12] that the authors compiled includes, besides scheduling: transportation and logistics, supply chain planning, general planning, facility location and bin-packing problems.

2.3.3 Agent-based systems and the rostering problem

The problem which is tackled in this project deals with optimising a base roster that has already been created. A variety of approaches have been used in experimenting with optimising rosters (see 2.2.3 Other approaches), however, currently, heuristics are generally the method of choice for rostering software designed to deal with real world objectives and rules which cannot be solved easily using mathematical approaches [4].

TRACSRoster was used as a starting point in understanding and presenting the rostering problem and also as a framework which could be used, eventually, as the foundation for further developing the roster optimisation process. The obvious approach towards improving the current optimisation method would involve extending the framework used in TRACSRoster, i.e. using different heuristic methods or automating some other methods which are currently manually used to optimise the base roster.

Initially, a range of heuristic methods were introduced as candidates for improving the optimisation of the base roster. The approaches included:

- Solving the violations within the roster by prioritising the rules. Rather than considering the violation sites with the highest penalties (which would represent the current implementation), bring in the rules incrementally, i.e. the violations for specific rules would be solved first, followed by rules with lower priority, regardless of the penalty of the violation sites.

- Solving the violations by solving specific types of potential violation sites first. As described in 2.2.2 Current approach, there are a number of potential violation sites created within the roster; the idea would be to try and solve a type of violation site before the others as each type of violation site pattern applies to some types of rules.
Using a 2-step look-ahead method based on the current framework. For each potential violation site created, the current system provides a list with improvements, thus, for a site which is violated, a list with the moves between different cells within the roster and the penalties resulted from the moves is provided so that the system can decide if it is worth moving a cell or not. The 2-step look-ahead approach would take the top improvements in this list and iterate over the algorithm once more to see if the top improvement considered before would actually be the best choice or one of the lower rated improvements would further improve the quality of the roster.

Line swapping is a handy method usually used when trying to optimise a roster manually. Sometimes, rather than trying to move around shifts in order to solve rule violations, moving just one line (or a block of lines) within a roster can fix more than one violation at the same time. This is an easy and useful method to have implemented which sometimes solves violations faster than the other methods.

Learning from experience which moves are useless and avoiding them throughout the optimisation process.

The methods presented above represent some initial ideas proposed earlier during the evolution of the project when the approach for optimising the roster was based on extending what was already provided by TRACSRoster.

The methods are mainly based on local search heuristics which implies that the solution is re-evaluated after each move within the roster, regardless of which method is used. All the methods described are based on an analysis of the effects of the rules imposed on the roster, so, it all goes down to what rules are applied and how they interact with each other.

Heuristic methods are used to create a good base roster which should be optimised further. However, after the creation of the base roster, using heuristics involves creating some intelligent, logic-intensive methods which do not represent a new approach towards roster optimisation anymore.

As a consequence, a list with desirable features that should be included in a method was created in order to decide what approach to follow in trying to optimise the base roster. These features included: autonomy of the violation sites, randomness, little or no communication between the sites which also leads to using some key measures in determining the quality of the roster rather than re-evaluating it after each cell movement (as it is the case with the local searches).
What is desirable of this approach would be to get as far as possible from the strong interconnection between the components and re-evaluation of the entire system after each alteration in the roster table.

All the characteristics presented above bring us back to the *open, highly dynamic, uncertain* and *complex* environments specific to agent-based systems. The roster table has many features in common with the environments specific to agent-based systems: it is modular, can be decomposed in cells, potential violation sites, lines, columns, etc.; dynamic in structure, the cells within the roster can move around quickly; it represents a complex, large scale system. Creating an approach based on autonomous agents means that each agent would try to maximise its output rather than interact with the other agents and optimise the system globally from the very beginning.
3 Schedule & Project management

3.1 Overview

The project takes place between week 11 of semester 1 (5\textsuperscript{th} – 11\textsuperscript{th} December 2011) and week 13 of semester 2 (14\textsuperscript{th} – 17\textsuperscript{th} May 2012). The deadline for the completion of the report is the 9\textsuperscript{th} May 2012 in week 12 of semester 2 and the presentation of the project should take place on the 18\textsuperscript{th} May 2012 in week 13.

3.2 Initial plan

The introduction of the report includes descriptions for the aim, objectives, minimum requirements and the deliverables of the project which are to be agreed upon in week 1. There has been allowed one week to complete this part of the project as the minimum requirements form should have been submitted at the end of week 1.

In order to manage the project better it was divided into a number of stages which were to be followed in order to keep track of the project development.

Firstly, a thorough background research of the rostering problem has to be carried out. The research includes extensive reading in the field of scheduling and rostering along with intensive exploration of the rostering optimisation package provided by Tracsis plc, TRACSRoster.

Secondly, after having a good understanding of the problem and current approaches used towards solving it, a discussion of the possible methods that could be undertaken towards solving the problem should take place, and some initial methods are proposed.

Thirdly, a planning of the implementation has to be devised and a decision upon the tools that have to be used is taken. Depending on the decision of the method that would be undertaken, an initial architecture of the model is designed and the approach in constructing the algorithm to implement it is discussed. Furthermore, an initial evaluation plan should be devised such that when the evaluation stage comes up, the initial one would only have to be extended.

The next stage in the process includes implementing the algorithm and based on the evolution of the implementation or any ideas that might be suggested afterwards, revisions of the architecture of the model and subsequently, the algorithm, might arise.
Finally, an evaluation of the method and algorithm should be performed. The evaluation should be a comparison between the current implementation in TRACSRoster and the newly implemented approach.

Ideally, the writing of the report should begin after the start of the background research and go on throughout the development of the project.

The background research itself is an ongoing process. As different methods of approaching the problem are discovered, an assessment should be made for each one of them and the best decision should be taken towards solving the problem.

### 3.3 Methodology

Although the software development methodologies discussed in the *Software Systems Engineering* module in the second year do not commonly apply to a Final Year Project which has a more evaluative and research-based purpose, the methodology undertaken for this project resembles the *Prototyping model*. Prototyping is usually used in industry when there is no clear idea of the method that should be implemented and different approaches are prototyped such that the one that would fit the requirements best will be chosen in the end [15].

Thus, the process to be followed falls between these stages:

- **Planning** – establishing the problem that has to be tackled;

- **Analysis, design and implementation** – this is a cyclic stage within the methodology which involves building a prototype and assessing its quality, and then deciding whether it is a satisfactory prototype or not; in case it is not, an analysis is carried out and a new design will be prototyped;

- **Final prototype** – is the last version of the previous stage and should meet all (or most) of the requirements;

This methodology allows the project development to flow in a more flexible manner and is more appropriate than other approaches because the roster optimisation problem can be tackled from many different perspectives, and at the start of the project there have been open discussions about which one to follow.
3.4 Schedule

The initial project schedule (see Figure D.1) was designed during week 1 as a desirable approach towards producing the project. The original schedule has changed at different points during the semester as a series of issues have been encountered.

Firstly, the core background research was planned to take place during the first five weeks of the semester; this should have been enough to have a clear vision of the rostering system in general and the train crew rostering in particular as these are presented in the literature, and any approaches considered towards optimising a roster. However, a significant part of the project is based on the software developed by Tracsis plc, TRACSRoster, and having a good understanding of what approaches it takes towards creating optimal rosters is essential. A software developer at Tracsis has offered his support with regards to getting the repository with the application and any additional software needed to be able to run it; he also provided some data that could be used for tests. There have been two questions and answers sessions in the first couple of weeks since the start of the project in which some issues regarding the understanding of TRACSRoster have been discussed. An impediment that was encountered when trying to get a grasp of the system was the lack of documentation, or a class diagram, which would have been extremely useful in this case; on top of this, the contact person from Tracsis happened to be on leave for a couple of weeks during that period which made it harder to resolve any issues that came across during that time.

Throughout the first four weeks I have explored the different solvers used in TRACSRoster and debated the different approaches implemented and how these performed in order to decide on a method to adopt for optimising the base roster. The first idea towards a solution was to create an agent-based system, more specific, to apply the ant-colony optimisation method. After some research on the method and the implementations which are currently in use, it was decided that this would not be the best approach towards a roster optimisation.

At the time of the mid-project report hand-in I have decided to go back a step and take the safer approach by investigating some local-search methods (the ones described in 2.3.3 Agent-based systems and the rostering problem) which would have been easier to implement considering the current framework of TRACSRoster which is also based on heuristics, however, the idea of creating an agent-based system was not entirely out of the question.
The revised project plan created on week 6 for the mid-project report is presented in *Figure D.2* in Appendix D.

Although the approach based on local searches started to take shape, at the end of week 8, a different agent-based model was proposed and it was discussed throughout week 9; at this point it was decided to go further with this model as it has not been approached before (compared to the local search heuristics) and try to fit the development in the remaining weeks.

The decision to pursue the agent-based model – regardless of the little time left to implement the architecture, test it and do the write-up of the project – was based on pursuing a novel idea which has not been exhaustively researched before and a good architecture on which the model could be created. It was decided to implement a small-scale prototype of the design, so that it could just be used for evaluation purposes.

The architecture created will be presented in detail in the following chapter (see *4 Design & Implementation*). The finalised project plan is presented in *Figure D.3 Final project plan* in Appendix D.
4  Design & Implementation

4.1  Overview

The purpose of this chapter is to provide an analysis of the designed architecture and the implementation. The design part will be providing an insight of the new approach with descriptions of the agents, how they interact within the system and their behaviour. In the implementation part there will be a description of how the architecture could be implemented.

4.2  Design

4.2.1  Architecture of the agents

The decision to take the agent-based approach towards optimising a base roster was based on the initial architecture of a roster table and its components. The roster table itself can be looked at as a matrix with seven columns (one for each day of the week) and a large number of lines as portrayed in a simplified version in Figure 7 below. The roster might be decomposed in different smaller tables (called links; see 2.2.1 Train crew rostering problem - Overview) due to different rules imposed; however, the same approach is applied to every link.

<table>
<thead>
<tr>
<th>Sunday</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
<th>Saturday</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>link 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>link 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7 Basic structure of a roster table with two links

As discussed in 2.2 Train crew rostering problem the main structures within a roster are the cells - each of them represents a day worth of work for a member of the crew and it can either correspond to a shift that has to be worked, an “as-required” turn, a training turn or a rest day,
and the violation sites – which represent different compositions of cells depending on the types of rules that have to be satisfied in the final roster. There are different other types of structures in which a roster can be broken down such as a line in the roster (this can also represent a violation site – see 2.2.2 Current approach), a column, etc., however, the smallest structures have been considered in order to have the freedom to move them around easily and be able to consider all the possible combinations. Thus, the two structures described above will be used to portray two types of agent for the approach.

Before describing the structure of the agents, their basic capabilities should be defined:

- The agents should be able to send and broadcast event messages, which would be used to keep them in sync with the changes within the roster table or ask for permissions and other such requirements;

- Some agents should be able to trigger modifications to the working solution;

- All the agents should be able to listen and respond to relevant event messages (relevant in the sense that some messages might be broadcasted to all the agents but they would be appropriate only to some of them);

- Lastly, the agents should be autonomous, unsupervised and probabilistic in some of their decision making.

The choice of agents is based on the two main structures within the roster: the cell and the violation site. Thus, the two types of agents are the roster element and potential violation site. The architecture for the two types of agents will be described further below.

Before describing the architecture of the agents the concept of a sink has to be introduced. A sink would essentially be a collection of roster elements. For a roster table there would be seven sinks available, one for each day of the week (a way to visualise this idea is presented in Figure 8). The sinks would be used by the potential violation sites in the following way: when a potential violation site is violated (it breaks a rule, for instance) it should determine the roster element (in some cases there might be more than one) that is part of it and is the most “harmful” (in the sense that it has the highest penalty or it is the only roster element that breaks the site); this roster element could then be discarded by the potential violation site by placing it in the sink specific to the column in the table (day of the week) which corresponds to that roster element (cells in a roster table can only move their location vertically – i.e. switch to a different line in
the table; turns which correspond to a day of the week cannot be switched with turns from other working days). After discarding a roster element, the potential violation site has to take up a different one in place as it cannot be valid unless it has all its components. The way in which a different roster element is picked up from the sink could depend on the range of elements available, or it can be a random choice (this will be discussed in more detail in 4.2.2 Behaviour).

A roster element (RE) agent is based on the idea of a cell in the roster table. The implication of this would be that the number of roster elements in the system is equal to the number of cells in the roster table. A roster element agent has to:

- Know its position in the roster table at any time;
- Expose its penalty and keep it updated - the penalty in a roster element would be the sum of all the penalties of the potential violation sites it is part of;
- Know if it is in the sink or not;

A potential violation site agent is clearly based on the idea of a violation site introduced in TRACSRoster. Although the two concepts have similar names, they do not have much in common in terms of how they should work within the system they are part of. This will be talked about in more detail in the next chapter that describes the behaviour of the agents and of the system itself.

<table>
<thead>
<tr>
<th>Sunday</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
<th>Saturday</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>link 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>link 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8** Representation of a roster table with sinks for each column
A potential violation site agent has to:

- Know the roster elements which form it at any time;
- Know the overall effect it has on the roster table; this should be expressed by exposing a penalty;
- Take the decision to move a roster element in the sink when this is the case;
- Broadcast its penalty to all the roster elements which form it each and every time this changes;

Now that the agents and the concept of a sink have been introduced, and their basic capabilities described, the behaviour of the system is next to be explained.

4.2.2 Behaviour

In order to better understand how the system works, a comparison between how the violation sites within TRACSRoster and the potential violation sites in the agent-based model differ, was found to be useful.

The main characteristic of the potential violation site agents is their autonomy. They know nothing about the other agents of their type, they are only aware of the roster elements which are part of their structure. Compared to the violation sites within TRACSRoster, these agents are not aware of the other roster elements in the roster besides the ones they are formed of as they don’t have to look for improvements themselves like it was the case before. The potential violation sites represent the concept of selfish agents that do not have any interest in the quality of other agents of their type, their only purpose is to satisfy the rules they have been imposed and improve their output which is known by evaluating their penalty. The lower the penalty of a potential violation site, the higher its quality gets.

In the situation when a potential violation site agent has its penalty equal to zero, which translates into the rule imposed being satisfied, this means that the ideal situation for this agent would be for no roster element that forms it to be moved into the sink.

On the other hand, when a potential violation site agent has its penalty larger than zero, it will analyse the roster elements that form it and try to get rid of the one that has the largest penalty. This rationale is applied on the basis that the roster element in case will, most probably, be part
of other potential violation sites, so, the larger its penalty, the larger the number of sites it breaks. Sometimes it might be the case that when discharging of a roster element and trying to exchange it with a better one, more than one site might improve its penalty; however, this is not a characteristic the system has described, but rather a more random decision taken within the sink.

The best way to describe how a potential violation site agent behaves when its penalty is larger than zero is by providing an example to support it. In Figure 11 below, there is a roster table which has a violation site on line 3. The violation site is made up of three roster elements on Thursday, Friday and Saturday. Let RE-1, the roster element on Saturday, be the one with the highest penalty between the three. RE-1 is decided to be transferred to the sink (process portrayed in Figure 9).

![Figure 9](image)

**Figure 9** A roster element from a violated site is transferred to the corresponding sink

After placing RE-1 in the sink, the violation site remains incomplete (see Figure 10), so it cannot be further evaluated.
The next step is for the potential violation site agent to request another roster element from the sink in order to be complete. The sink will return a roster element that is available at that moment by randomly picking up one from the collection.
After each change within the roster elements of a potential violation site, the latter has to recalculate its penalty and broadcast it to all the roster elements in its composition. If the new penalty of the site is still non-zero, the process described above will start again. If the exchange of roster elements with the sink was successful and the site is not violated anymore, no more changes have to be made for the moment; however, it still does not mean that the site would stay the same. The roster elements can be part of more than one potential violation site, and usually this is the case, such that at any point, the structure of a potential violation site agent could change as another agent has moved a roster element. This constant exchange of roster elements between the potential violation site agents and the sinks resembles a race in which each site tries to minimise its penalty and eventually gets to satisfy the rule it breaks.

As mentioned earlier, the roster elements themselves have a penalty which is the sum of all the penalties of the potential violation sites that they are part of. This means that a potential violation site might have a penalty (determined by the rule it must satisfy), but the roster elements that are part of that site, taken separately, might have different penalties and in most cases these will be higher.

\[
\text{penalty(PVS)} \leq \text{penalty(RE}_i)\]

*PVS*: Potential Violation Site,

*RE*: Roster Element,

\(i\): index of a roster element included in the potential violation site,

Thus, the roster elements have to keep themselves updated but also broadcast messages to all the potential violation sites they are part of, each and every time their penalty changes. They also have to accept messages from the potential violation sites which might in turn update their penalties based on changes made to the other roster elements. The exchange of penalty values is crucial to the evaluation of the penalty of the entire roster table in order to make sure it is consistent.

With regards to the behaviour of the sinks, these can be looked at as some collectors specific to each day of the week with elements discarded by the potential violation sites. Originally, the sinks are empty, but once the agents are run, they start receiving and returning roster elements. The role of the sinks is to supervise the exchange of roster elements that takes place within the roster table. The sinks are the only objects within the system that know the number of discarded roster elements and what they are, whereas in the current approach, the violation sites would be able to know the outcome of their moves for any cell in their structure by accessing the
collection with improvements. The initial thought is to have the sinks return a random element back to a potential violation site, however, this could be customised such that, depending on what element a potential violation site sends to the sink, it would receive back either a roster element of a different type (the rule that the potential violation site must satisfy could be also taken into consideration) or simply a random one which could include elements of the same type. There are different ways in which this exchange could be approached, however, the entire architecture is based on agents as this makes the system more dynamic and unpredictable, so imposing too many restrictions might not serve the purpose of the method.

Having an understanding of how the two types of agents behave and what is the purpose of the sinks, the next step would be to describe the more high-level parts of the system. As stated before, there are two types of agents: roster element and potential violation site. After being provided the base roster, based on the rules that have to be satisfied, the system would create all the possible agents; the number of roster elements would be the same with the number of cells in the roster table, whereas the number of potential violation sites varies depending on the rules applied. However, all the instances of the agents have to be created from the very beginning.

One of the most important parts of the system is represented by each agent running on its own thread; this raises some issues within the architecture regarding the access to data collections, but also makes it possible for the agents to be independent and run asynchronously.

One other main characteristic is the random behaviour of the agents. This can either be explicitly used in a probabilistic manner for the firing of events and sending messages, or could be due to the asynchronous behaviour of each agent running on a different thread – e.g. if different agents want to access the sink at the same time, which of them will actually get to access it cannot be known in advance.

Finally, the system has to specify a stopping criterion. In the most fortunate situation, the base roster will be optimised such that all the violation sites will have a zero penalty – which means that all the rules have been satisfied, thus the agents can stop running and the optimised roster table can be returned. However, this is usually not the case; so, a different stopping criterion has to be considered such that the agents would not run forever. One of the most common ways would be to set a time threshold, and once that period of time since the agents have been running is reached, they should stop and the best solution up to date should be returned.

The system will also have to implement a way in which the best solution to date would be remembered in memory and each and every time a new table would be complete – at a point
when the violation sites have all their roster elements in place, i.e. the sinks are empty, the system would check the penalty of the latest solution against the top one to date and keep the best one.

4.2.3 Locking

Along with the desirable asynchronous behaviour of agents comes the issue of keeping consistent the elements which can be accessed by more than one thread. The objects that have to be overviewed for consistency issues are the roster table and the sinks.

The main roster table would be accessed by potential violation sites each and every time there is a change within their roster elements. The easiest way to approach this would be to lock the entire roster table each and every time a change has to be made. This would definitely ensure its consistency as it would be thread safe – only one thread could access it at any time. Nonetheless, this approach would not be the most efficient. In the case of a large roster with some hundreds of roster lines, the number of potential violation sites would be extremely large, thus the necessity for more than one thread to be able to access some parts of the roster table at the same time. In order to maintain the consistency of the roster table there might be devised different ways in which this issue can be approached. For instance, it is known that the exchange of roster elements between the potential violation sites and the sinks is reduced to columns – i.e. a roster element can only be changed with another one in the sink from the same column in the roster table. In this way, it would be feasible to give the potential violation site agents the possibility to access the table at the same time if the changes that have to be made are not in the same column in the roster. This could be extended further by either trying to partition the roster in such a way that no deadlocks or concurrency issues would occur or imposing more checks when allowing a thread to access the table.

As for the sinks, the system has to make sure that at no time will a sink try to send back the same roster element to more than one potential violation site, or missing to add a roster element - as omitting one would make the table invalid; so, both adding and removing roster elements are critical moves that have to be supervised carefully. This means that each sink will have to apply a locking mechanism for both adding and removing roster elements so that only one add or remove can be executed at any one time.
4.3 Implementation

4.3.1 Overview

This section will present the steps taken towards implementing the approach discussed in the design section described above. Having TRACSRoster as a piece of software that implements a system that helps with the rostering process and its optimisation and using it in order to gain the knowledge required to understand the rostering process and the stages undergone by a roster table when being optimised, it was only natural to consider an implementation of the new architecture based on the application already in use.

4.3.2 Algorithms

In section 2.2.2 Current approach, an overview of the method currently used within TRACSRoster is presented. The concept of a violation site is also introduced as a potential location for a rule violation. Potential violation site agents expose common properties with the violation sites; however, there is a wide range of structures for the violation sites (see 2.2.2 Current approach), so for the purpose of prototyping the system, only one type of structure was chosen to be implemented, for only one rule.

![Shift-to-shift structure of a violation site](image)

**Figure 12** Representation of a shift-to-shift violation site

The shift-to-shift structure of a violation site is presented in Figure 12 above. This structure is used for checking the validity of a range of rules such as:

- **Rest time between turns rule** – checks that the period of time between the sign off time of the first turn and the sign on time of the second turn is valid.

- **Shift precedence rule** – checks that the categories of the two shifts do not conflict; for instance, a late shift in the first day is not allowed to be followed by an early shift in the following day.
• **Maximum sign on variation rule** – checks that the time interval between the sign on time of the first shift and the sign on time of the second shift is a valid one.

The *shift-to-shift* violation site was picked up to be implemented because of its simple structure which makes it easier to understand how the system works; however, it also raises some issues regarding the sharing of *roster elements* between the *potential violation sites* so it serves its purpose very well. For each and every rule imposed on the roster, all the possible violation sites for that rule are created, so in this case, the *rest time between turns* rule was chosen to be used.

In *Figure 13* above there are represented four *shift-to-shift* violation sites: the first two ones are on line two for Sunday and Monday and on line three for Monday and Tuesday and the other two ones are on line three for Thursday-Friday and Friday-Saturday. The roster cell in black on Friday represents the *roster element* which is shared by the violation sites.

The implementation of the *roster element* agent involves exposing a series of properties available for the *sinks* and the *potential violation site* agents. These include: the *type of shift* the *roster element* is allocated - i.e. scheduled live turn, non-live turn, training turn, as-required turn or rest day, the position (*line* and *day*) the element in occupying in the roster table (in the case when the element is not in the sink), a Boolean value that keeps record of whether the element is in the sink or not, and a *penalty* property which would obviously return the penalty at any time.
For the implementation of the potential violation site agent we have to go back to the shift-to-shift violation site in order to derive the exposed properties. Creating this type of agent means that it would only have to expose the two roster elements in its composition (either by keeping in sync a collection with the elements or keeping them as two separate properties) and the penalty of the violation site as returned by the evaluation function within the rule.

For the agents to be autonomous and unsupervised they will each be running on a thread. The methods which would run the agents are presented below for each type.

**Figure 14** Pseudocode for the method that runs the roster element agent

```plaintext
function RunRosterElementAgent ()
    while ( ! (thresholdReached || foundSolution)) do
        if (penaltyHasChanged) then
            BROADCAST_PENALTY_TO_POTENTIAL_VIOLATION_SITES ()
        fi
    od
```

**Figure 15** Pseudocode for the method that runs the potential violation site agent

```plaintext
function RunPotentialViolationSiteAgent ()
    while ( ! (thresholdSitelsBroken || foundSolution)) do
        if (violationSitelsIsBroken) then
            rosterElementToMoveToSink ← FIND_RE_WITH_HIGHEST_PENALTY ()
            MOVE_RE_TO_CORRESPONDING_SINK (rosterElementToMoveToSink)
            newRosterElement ← GET_NEW_RE_FROM_SINK ()
            UPDATE_AGENT (newRosterElement)
            UPDATE_PENALTY ()
            BROADCAST_PENALTY_TO_REs ()
            SEND_NEW_RE_TO_SOLUTION_OBSERVER ()
            BROADCAST_NEW_PENALTY_TO_SOLUTION_OBSERVER ()
        fi
    od
```
Each of these methods will be run in a separate *thread* for each agent. The agents communicate by raising events each time a relevant property changes. For the *roster element* agent, this involves any changes that could affect its penalty: when a *potential violation site* that includes the *roster element*, changes its penalty and broadcasts it to its *roster elements*, the latter have to update their current penalty and broadcast it back to all the *potential violation sites* they are part of. On the *potential violation site* side, when its penalty changes, it has to broadcast it to all the *roster elements* which form it and also be able to listen to event messages regarding any changes within the *roster elements* that form it – this has to do with the fact that more than one *potential violation site* can have the same *roster element* in its composition so changes to one *roster element* can be applied from more than one place (see *Figure 13*, p.37, both the *Thursday-Friday* and the *Friday-Saturday* violation sites, as agents, are able to put the common *roster element* – Thursday – in the sink without asking for permission from the other one).

As it can be observed from the *pseudocode*, the methods that run the agents include a *while-loop* with some termination conditions. There are two termination conditions, and either of them has to be satisfied in order for the loop to stop. The first condition is *thresholdReached*; a time threshold is set for the system such that if there was no solution found for the roster table – i.e. there are *potential violation sites* still broken – the system will not run forever. The second condition, *foundSolution*, represents a Boolean value which will be set to *true* when the system finds a solution - i.e. no *potential violation site* is broken anymore.

As for the *sinks*, the easiest way in which these can be organised would be as a collection of collections. As specified in 4.3.2 *Locking*, there must be paid great attention to the way these collections are accessed by the agents as consistency is crucial to the good functioning of the system.

```
function AddREToSink (rosterElement)
lock (REsInSinkCollection)
    REsInSinkCollection.ADD (rosterElement)
unlock

function RemoveREFromSink (rosterElement)
lock (REsInSinkCollection)
    REsInSinkCollection.REMOVE (rosterElement)
unlock
```

*Figure 16* Locking *sinks* when adding/removing *roster elements*
In Figure 16 there are presented two pieces of pseudocode that show how locks can be used to define a critical section on the adding or removing of roster elements to/from the collection in a sink. There are different other ways of implementing locking such as using semaphores or monitors and these should be considered in the case of trying to make the system more efficient or if a different approach would be considered for accessing the sinks.

```plaintext
function ReturnREFromSink (dayOfTheWeek)
    sink ← GET_SINK_ON (dayOfTheWeek)
    lock (sink.REsInSinkCollection)
        REToReturn ← GET_RANDOM_RE_FROM (sink)
        SEND_RE_TO_POTENTIAL_VIOLATION_SITE (REToReturn)
        UPDATE_SINK (sink)
    unlock
```

**Figure 17** Pseudocode for locking a sink when getting a roster element from the corresponding sink to return to the potential violation site

As emphasised earlier in the design section, randomness in the design of this approach is very important. Firstly, it is introduced automatically by the use of threads to run the agents – as these cannot be controlled when trying to access a collection/resource or trying to make a change, i.e. the order in which they access it is not controlled by any algorithm. However, this is not quite enough to ensure the system will run smoothly. There is a risk of the system getting stuck in a loop in its interaction with certain potential violation site agents by always sending the same roster element back from the sink and the other way round, or even worse (and harder to detect) having a longer cycle which involves a sequence of roster elements which are sent back and forth. For this reason, a way to keep in check that this would not happen must be implemented. A mechanism which would detect such cycles could be implemented on the side of the potential violation site agents.

Finally, in order to store the solutions to the system and keep in check and approve of any changes made by the potential violation sites, a solution observer has to exist in the background. This solution observer should contain the table with the base roster at the beginning and while the agents are running, it should listen and accept any changes made to it. A vital role played by the solution observer is to keep the roster table consistent and make sure of the validity of any changes attempted by the potential violation sites. It should implement a locking mechanism as
suggested in 4.2.3 Locking: either lock the entire table for each and every change – although this would be the safest way to make changes to the table, considering the number of potential violation sites, each running on its own thread, the queue with threads trying to access the table might get very large; or only lock specific locations within the table which would ensure that there would be no changes anywhere else within the table but that section – as it might be the case with locking the columns, i.e. roster elements can only be moved upwards or downwards within a column so locking each column separately might be a good approach.

```python
function MakeChangeToRosterTable (potentialViolationSite)
    position←GET_POSITION_OF_PVS (potentialViolationSite)
    lock (rosterTable)
        CHECK_FOR_CHANGES_IN_THE_TABLE (position, potentialViolationSite)
        UPDATE_CHANGES (position, potentialViolationSite)
        oldPenalty←rosterTable.Penalty
        UPDATE_ROSTER_PENALTY (potentialViolationSite)
        if (oldPenalty > rosterTable.Penalty) then
            KEEP_NEW_ROSTER_SOLUTION (rosterTable)
        fi
        if (rosterTable.Penalty == 0) then
            isOptimal←TRUE
            STOP_AGENTS_AND_OUTPUT_SOLUTION ();
        fi
    unlock

Figure 18 Pseudocode for the method accessed by potential violation site agents that want to make changes to the roster table
```

Furthermore, what has to be remembered is that, because of the autonomy of the agents and the way they run and search for solutions, if the system does not stop on the basis of finding a solution which does not break any of the rules, but because of the time threshold, the way the solution varies is not tractable. New solutions would be found as the roster elements change their position in the table, but the quality of the working solutions may go up and down between any two consecutive solutions. This differs from the current way of optimising the table which is
based on heuristic searches and allows changes within the roster table only if the current best solution is improved. For this reason it is important for the solution observer to keep record of the current best solution and keep this updated if better solutions occur. Once the time threshold is up and the agents stop running, the best solution to date should be returned.

4.4 Conclusion

This chapter introduced the design for the method proposed and suggested some ideas for its implementation. Since the agent-based approach towards optimising a train crew roster has barely been discussed in the literature, if at all, this can be regarded as a new way towards solving this problem and it could be extensively examined in many more pages, and different ways to implement the design could be debated in order to decide which would be the best. However, for the purpose of this report and the period of time given for exploring such a broad subject, the focus was on discussing the main components within the design and expanding only on the critical parts of it.
5 Evaluation

5.1 Overview

The goal of the project is to provide a model that would optimise the current train crew rostering approach. For this purpose, a rostering system already in use – TRACSRoster – has been explored for the purpose of better understanding the current optimisation process. The agent-based model created tries to use the current framework of TRACSRoster as part of its initial implementation.

In this chapter a set of criteria will be proposed for evaluating the approach described in the previous chapters. A comparison between the proposed method and other approaches will be discussed and future work and extensions to the method will be proposed. Finally, some of the issues encountered during the development phase will be discussed in more detail.

5.2 Evaluation plan

5.2.1 Overview of the methods

The criteria proposed for the evaluation was devised after creating the basic structure of the new model. The approach described in this report is based on the idea that the system would be provided with a base roster table created in advance (by TRACSRoster, for instance) and the system would try and optimise it in order to reduce the penalty of the roster and the number of broken rules, thus getting a higher quality solution.

In order to evaluate the new system, its performance would have to be compared to a different optimisation system such as TRACSRoster.

The main difference between the two architectures is in the type of methods used and the evaluation should focus how the methods manage to result in a better solution rather than how well these would perform in a specific timeframe. The quality of the resulted solution is more important that the time it would take a system to achieve that solution. Crew scheduling and optimisation involve very large datasets and the way in which these systems are optimised is known to involve running different optimisation systems for long hours or even over the night in order to be able to get some good solutions, thus, this criteria would be relevant when the same method would be implemented in different ways, however, this is not the case.
As described in section 2.2.2 Current approach, TRACSRoster uses different heuristic methods in order to improve the solution provided by the base roster. For each violation found in the roster, it looks over all the possible swaps that could be made and then see which one of them would result in a lower penalty and perform that swap. Once the system reaches the stage when there are no other swaps that could be performed such that the solution would improve, it stops and returns the results.

The agent-based method proposed in this report is based on some autonomous, unsupervised agents that would run asynchronously, each of them trying to improve its own quality which would eventually lead to a better solution for the entire roster.

5.2.2 Criteria

In order to decide which approach would be more efficient in returning better solutions, it would be preferable to evaluate the systems on base rosters which involve different levels of difficulty when being solved. Taking into consideration the fact that the initial version of the agent-base system only considers the implementation of a shift-to-shift violation site for a sign on/off gap rule, the testing would be restricted by the number of rules which can be applied on the roster, so, the quality of the base roster must be changed in order to monitor the performance of the two systems.

Once the agent-based system would implement potential violation site agents for more than one rule, a better idea would be to use the same base roster and increase the number of rules which are applied to it and see how the performance of the systems changes. This approach would probably be preferred to the one described above.

Considering the fact that both systems evaluate the quality of their solutions using a penalty measurement, the best criteria for comparing the outcome of the two systems would be a comparison between their penalties.

Another criterion that has to be used (which also represents an issue that has to be raised when running the agent-based system) is represented by the time threshold which should be specified for the agents to stop running. A range of durations should be used and some measurements on how the system performs should be considered in order to determine which time interval is the best. In theory, this would be quite hard to assess considering the nature of the system which is mostly based on the idea of randomness and selfish agents, however, there is a chance of observing a pattern in the way elements are returned from the sink, or it might be that some
agents will always get the \textit{roster elements} of their choice whereas for some others that might never happen.

\section*{5.3 Implementation analysis}

\subsection*{5.3.1 Overview}

As pointed out in 2.3 \textit{Schedule}, the course of the project changed at two points during the semester which shortened the time for both design and implementation. As the implementation started at a late stage it was decided to use bits of the code from TRACSRoster in order to implement the approach. This would have been the only way in which the code could be finished on time. Thus, the implementation was decided to be done in C# (as TRACSRoster is implemented in this programming language).

Unfortunately, although most of the framework has been implemented, there were found some bugs relating to the threading part of the model. As the project was implemented in C# so that it did not have to be written from the very beginning, it was a bit harder to implement the autonomous agent-based system by using the strongly connected architecture implemented for the heuristic approach in TRACSRoster. Therefore, even if the approach for the evaluation part was clear, there was not enough time to finish the coding, perform the evaluation and do the write-up of the final report.

Even so, there are some factors that could be discussed in terms of how likely it would be for such an approach to outperform the extensively used heuristic methods.

\subsection*{5.3.2 Likelihood of approach}

As far as the scheduling optimisation problems are concerned, heuristic methods have proved to be the most used approaches. Heuristics have been extensively employed in optimising the solutions to these highly constrained approaches as they proved to be very reliable for this type of problems. However, more recently, new research brought to light new methods which could be applied for this sort of problems and one such method is presented in this report, the agent-based systems.

Agent-based systems have all the chances to be the next big approach towards scheduling optimisation as they include in their basic framework some ideas which are fundamental to scheduling such as \textit{robustness} – in the sense that agents could span at any time and would not affect the structure of the system, unlike the case with heuristics where the collections with
elements have to be maintained, and *flexibility* – in the sense that the agents are goal-specific and work for their own benefit rather than trying to solve conflicts globally as it would be the case with the heuristic methods.

Moreover, the “amount” of *randomness* involved in the agent-based systems has all the chances to lead to new and unexpected solutions to these massive problems.

There are many advantages the agent-based systems expose to the area of scheduling and optimisation, and some of these are presented in [16] by Mes et al who considered the problem of real-time transportation – scheduling of full truckload transportation orders. They present a review of some look-ahead heuristics which are usually used for this type of problems which require fast solving and go on to introduce an agent-based approach where intelligent agents acting as vehicles, schedule their own routes. The advantages this system presents include how fast it works, the relatively little amount of information it requires and the fact that it facilitates easy schedule adjustments with regards to the information updates [16]. The paper goes on to compare the heuristic and agent-based approach. After carrying out the proposed experiment, Mes et al reached the conclusion that "the agent approach yields a high performance in terms of vehicle utilisation and service level."

Although extensive research still has to be carried out in this field, the characteristics of agent-based models and the research to date points to the idea that this approach might be successfully implemented in the near future to solve real-life problems and perform at least as well as heuristics do nowadays.
6 Conclusions

6.1 Objectives and requirements

The objectives and minimum requirements for this project have been outlined in 1 Introduction. More details on how these criteria were met will be listed below:

- The rostering process has been explained with the use of TRACSRoster – a software tool used for train crew rostering optimisation in industry. This has been achieved in section 2.2 Train crew rostering process under 2.2.2 Overview.

- A thorough analysis has been carried out on the current rostering system (i.e. TRACSRoster) and details were provided for the most important parts of the architecture. This has been achieved in section 2.2.2 Current approach.

- Different other approaches used in crew roster optimisation have been discussed and the differences between them presented. This has been achieved in section 2.2.3 Other approaches.

- An agent-based model for a new train crew rostering system has been discussed and the design was systematically presented in 4 Design & Implementation mostly under 4.2 Design.

- The implementation of the newly designed architecture has been explained and the main challenges discussed under section 4.3 Implementation. A proof of concept for the system has also been outlined under the same section; however, the time limit for completing the project proved to be too short to finish the implementation of the prototype.

- Under section 5 Evaluation, a plan for evaluating the new system has been described and the criteria that should be used for a comparison between the different approaches towards the rostering process has been specified and supported with arguments.

- A discussion of the appropriateness of the proposed system has been carried out in 5.3 Implementation analysis. The ideas presented have been backed up with some experiment results carried on an agent-based, fully implemented system.
6.2 Possible extensions & Future work

Possible extensions to this project will be further presented:

- The obvious approach would be to fully implement the model proposed in this project and finish carrying out the evaluation proposed and compare the achieved results with the assumptions made.

- A possible extension would be to have a totally separate implementation of the design using an agent-based programming language (such as JADE for example). This would involve an implementation of importing a base roster table and not using any previous design for a different approach towards rostering and although this might take more time to get done, it would represent a clearer system that could be easily extended.

- As the approach considered in this project is based on autonomous agents and is mostly distributed, a parallel implementation would be interesting to implement. This would provide the proper environment for designing more types of agents and the time it would take for the system to run and achieve a better quality solution might be shortened.
Bibliography


Appendix A

Personal reflection

Personally, what I liked most about this project was exploring the agent-based approaches more thoroughly as we did not go into much detail in the Artificial Intelligence module in the second year, and combining them with an actual problem within transportation – the train crew rostering.

As this section is required to be written as a piece of advice for future students dealing with the same type of projects, I will try to outline the main issues I have encountered and some ways in which I could have approached the project so that the process would have been easier and the outcome of the report better.

To begin with, if the topic of the project is of a larger scale, and, rather than knowing from the very beginning the exact methods that will be investigated, the student is given the opportunity to put his mark on the project and come up with ways in which it could be approached, the decision of the method used should be taken at an early stage within the semester. Even if at a later stage one might find a more interesting way to approach the project, it would be to the best to stick to what was first decided. I could not stress the importance of this as personally, this has been the main struggle and source of problems during the semester.

Moreover, it is known from the very beginning that extensive background research must be carried on the problem, and, a critical approach should be taken. However, when working on a project within a “niche field”, one might be dealing with the issue of not finding many relevant sources. Even so, I firmly believe that one must try to find information about research carried by different people because papers are written in an extremely persuasive manner and you might find yourself in the position of agreeing with papers that contradict.

Furthermore, some students are in a position of having to deal with inspecting frameworks developed by different people or companies. This part of the project is a black box as, even if you know the higher-level approach of the problem, you never know how it has been implemented, so, enough time should be allocated for this part. If you will have to deal with smaller pieces of code, or libraries, it might be an easier task to undertake, however, if you are dealing with an entire framework as it was the case for me, and there is no documentation or class diagrams provided for it, you have to prepare for a time-consuming and frustrating
procedure of trying to break down the architecture. I must admit that this part has taken me longer than expected.

If some of the evaluation of the project involves implementing the developed design, then I believe some time should be spent on deciding what programming language to use for writing the implementation. This might not be the case for small pieces of code that only implement some small-sized algorithms, however, if the project is about designing a system and then the evaluation involves prototyping that system, the programming language is definitely an issue to be thought of. In my case, I have decided to write my implementation in the same programming language as the provided application. I believe this has not been the best approach as my design was based on other assumptions, and, using an agent-based programming language would have probably made my work easier and I might have managed to finish the work on schedule.

Lastly, probably in most of the final year projects, students state the time management issues in their reflection section. Time management issues arise within different stages during the process, when some parts are either under- or overestimated with respect to the time it takes to get them done. I believe this happens so often because, during our degree, we have not been involved in developing projects of such nature or length. Most of the development done during the course has been based on code already provided and there was no thorough research that we were supposed to do in advance, so tackling this type of larger scale project is challenging. My suggestion would be to create the project plan based on your own strengths; for instance, if someone struggles with the writing part, then more time should be allowed for it.

If I were to undertake this type of research project again, the crucial difference that I would adopt during its course would be deciding on what approach to take for the project at an early stage and try to stick to it if nothing vital would impede this.

All in all, this project has been a really interesting exploration of how combining two fields could lead to discovering new ways of solving some real-life problems and also provided some insight into research from a Computer Science perspective.
Appendix B

External contributions

The external contributions towards this project include the code for TRACSRoster, provided by Tracsis plc and developed within the company. This code has been used for understanding and describing the rostering approach used within their framework; small parts of the code have also been used when implementing some parts of the prototype system (this is explained and described in different parts of the project).
Appendix C

Ethics

No ethics issues have been raised or encountered within the development of this project.
**Initial project plan**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
<th>Week 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researching relevant literature/ software</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing initial architecture model/algorithm</td>
<td>☒</td>
<td></td>
<td></td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing the algorithm</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revising architecture model/algorithm</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing evaluation procedure</td>
<td>☒</td>
<td></td>
<td></td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performing algorithm tests</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing the report</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure D.1** Initial project plan
Figure D.2 Revised project plan; included in the Mid-project report

<table>
<thead>
<tr>
<th>Activity</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
<th>Week 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researching relevant literature/software</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing initial architecture model/algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing the algorithm</td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revising architecture model/algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing evaluation procedure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performing algorithm tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing the report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure D.3 Final project plan