Heuristic Algorithms for Static and Dynamic Frequency Assignment Problems

by

Khaled Alrajhi

Thesis submitted to Cardiff University

In candidature for the degree of

Doctor of Philosophy

School of Mathematics

Cardiff University

July 2016

The Most Beneficent, the Most Merciful

To

My Mother, Muzna, My Father, Abdulaziz

And

My wife, Hend

Acknowledgements

First and foremost, I praise and acknowledge Allah, the most beneficent and the most merciful. Secondly, there are many people without whom this thesis would never have come about. Firstly, Dr Jonathan Thompson for his supervision, support, friendship and patience throughout the past few years; secondly, Dr Rong Qu and Dr Christine Mumford for their invaluable advice and comments; thirdly, Professor Paul Harper, Professor Paul Rosin, Professor Nikolai Leonenko, Dr Maggie Chen, Dr Ahmed Kheiri, Dr Yu-Kun Lai, Dr Xianfang Sun, Dr Iskander Aliev, Dr Rhyd Lewis, Dr Timm Oertel, Dr Chris Davies, Dr Andrey Pepelyshev, Dr Stuart Allen, Mr Fawaz Alanazi and Mr Wasin Padungwech for their review and invaluable comments; fourthly, my parents and my wife for their love and their support, fifthly, my government for the sponsorship and support; finally, to all my friends and the staff of the School of Mathematics at the Cardiff University for their invaluable assistance.

Summary

This thesis considers the frequency assignment problem (FAP), which is a real world problem of assigning frequencies to wireless communication connections (also known as requests) while satisfying a set of constraints in order to prevent a loss of signal quality. This problem has many different applications such as mobile phones, TV broadcasting, radio and military operations. In this thesis, two variants of the FAP are considered, namely the static and the dynamic FAPs. The static FAP does not change over time, while the dynamic FAP changes over time as new requests gradually become known and frequencies need to be assigned to those requests effectively and promptly. The dynamic FAP has received little attention so far in the literature compared with the static FAP.

This thesis consists of two parts: the first part discusses and develops three heuristic algorithms, namely tabu search (TS), ant colony optimization (ACO) and hyper heuristic (HH), to solve the static FAP. These heuristic algorithms are chosen to represent different characteristics of heuristic algorithms in order to identify an appropriate solution method for this problem. Several novel and existing techniques have been used to improve the performance of these heuristic algorithms. In terms of TS, one of the novel techniques aims to determine a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model. These lower bounds are used to ensure that we never waste time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bounds, since there is no feasible solution in this search area. Another novel technique hybridises TS with multiple neighbourhood structures, one of which is used as a diversification technique. In terms of ACO, the concept of a well-known graph colouring algorithm, namely recursive largest first, is used. Moreover, some of the key factors in producing a high quality ACO implementation are examined such as different definitions of visibility and trail, and optimization of numerous parameters. In terms of HH, simple and advanced low level heuristics each with an associated independent tabu list are applied in this study. The lower bound on the number of frequencies that are required from each domain for a feasible solution to exist is also used.

Based on the experimental results, it is found that the best performing heuristic algorithm is TS, with HH also being competitive, whereas ACO achieves poor performance. Additionally, TS shows competitive performance compared with other algorithms in the literature.

In the second part of this thesis, various approaches are designed to solve the dynamic FAP. The best heuristic algorithms considered in the first part of this thesis are used to construct these approaches. It is interesting to investigate whether heuristic algorithms which work well on the static FAP also prove efficient on the dynamic FAP. Additionally, several techniques are applied to improve the performance of these approaches. One of these, called the *Gap* technique, is novel. This technique aims to identify a good frequency to be assigned to a given request. Based on the experimental results, it is found that the best approach for the dynamic FAP shows competitive results compared with other approaches in the literature. Finally, this thesis proposes a novel approach to solve the static FAP by modelling it as a dynamic FAP through dividing this problem into smaller sub-problems, which are then solved in turn in a dynamic process. The lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model, and the *Gap* technique are also used. The proposed approach shows the ability to improve the results which have been found by the heuristic algorithms in the first part of this thesis (which solve the static FAP as a whole). Moreover, it shows competitive results compared with other algorithms in the literature.

Acronyms

ACO	Ant colony optimization
ACS	Ant colony system
AS	Ant system
CN-tabu	Consistent neighbourhood in tabu search
FAP	Frequency assignment problem
GA	Genetic algorithm
GCP	Graph colouring problem
HH	Hyper heuristic
KCP	K-colouring problem
LLHs	Low level heuristics
LS	Local search
MMAS	MAX-MIN ant system
MO-FAP	Minimum order frequency assignment problem
MS-FAP	Minimum span frequency assignment problem
MI-FAP	Minimum interference frequency assignment problem
RS	Random search
RTS	Reactive tabu search
SA	Simulated annealing
SLS	Stochastic local search
TS	Tabu search
TSP	Traveling salesman problem
VRP	Vehicle routing problem

List of Publications

- Alrajhi K., Thompson J. and Padungwech W., 2016. A heuristic approach for the dynamic frequency assignment problem. *Journal of Computers & Operations Research.* [in review]
- Alrajhi K. and Padungwech W., 2016. A dynamic tabu search algorithm for solving the static frequency assignment problem. *Lecture Notes in Computer Science, proceedings of the 16th UK Workshop on Computational Intelligence, UKCI 2016*. Springer.
- Alrajhi K., Thompson J. and Padungwech W., 2016. Tabu search hybridized with multiple neighbourhood structures for the static frequency assignment problem. *Lecture Notes in Computer Science 9668, pp. 157-170, proceedings of the 10th International Workshop on Hybrid Meta-heuristics, HM 2016.* Springer.

List of Presentations

- Alrajhi K. and Padungwech W., 2016. A dynamic tabu search algorithm for solving the static frequency assignment problem. *The 16th UK Workshop on Computational Intelligence, UKCI 2016.* Lancaster, UK.
- Alrajhi K., Thompson J. and Padungwech W., 2016. Tabu search hybridized with multiple neighbourhood structures for the static frequency assignment problem. *The 10th International Workshop on Hybrid Meta-heuristics, HM 2016.* Plymouth, UK.
- Alrajhi K. and Thompson J., 2016. Heuristic algorithms for static and dynamic frequency assignment problem. *The Wales Mathematics Colloquium Conference*. Powys, UK.
- Alrajhi K. and Thompson J., 2016. Heuristic algorithms for static and dynamic frequency assignment problem. *The 5th Society for Industrial and Applied Mathematics, SIAM, National Student Chapter Conference*. Cardiff, UK.
- Alrajhi K. and Thompson J., 2016. Heuristic algorithms for the static frequency assignment problem. *The 5th Student Conference on Operational Research, SCOR16*. Nottingham, UK.

- Alrajhi K. and Thompson J., 2015. Static and dynamic frequency assignment problems. *The 29th Belgian Conference on Operational Research, ORBEL29*. Antwerp, Belgium.
- Alrajhi K. and Thompson J., 2013. Meta-heuristic for the static frequency assignment problem. *The Conference of the South Wales Operational Research Discussion Society, SWORDS.* Cardiff, UK.
- Alrajhi K. and Thompson J., 2013. Meta-heuristics for the static frequency assignment problem. *The 55th Conference of the Operational Research Society, OR55.* Exeter, UK.
- Alrajhi K. and Thompson J., 2012. Tabu search and ant colony optimization. *The Conference of the South Wales Operational Research Discussion Society, SWORDS.* Cardiff, UK.

List of Posters

- Alrajhi K. and Thompson J., 2015. Tabu search for the dynamic frequency assignment problem. *The 3rd Society for Industrial and Applied Mathematics, SIAM, National Student Chapter Conference*. Cardiff, UK.

Contents

List of Figures

Chapter 6 – Approaches for Dynamic and Static FAP

Chapter 7 – Conclusions and Future Work

List of Tables

Chapter 1 – Introduction

Chapter 2 – Literature Review

Chapter 3 – Tabu Search for the Static FAP

Chapter 4 – Ant Colony Optimization for the Static FAP

Chapter 5 – Hyper Heuristic for the Static FAP

Chapter 6 – Approaches for Dynamic and Static FAP

Chapter 1

Introduction

1.1 Context

Scarcity can be considered as the problem of having almost unlimited human wants in a world of limited resources. Many essential commodities such as oil, water and food are scarce due to their limited supply and rapidly increasing demand. One essential resource that is increasingly scarce, though not often considered by economists, is the radio frequency spectrum. The demand placed upon the usable spectrum increased exponentially between 1950 and 1980 [83]. Since then, the demand of frequencies for communication devices has further increased with the advent of high definition (HD), satellite television channels, mobile phones, satellite navigation systems and Wi-Fi, all of which require frequencies from the crowded radio spectrum. Mobile phone operators have paid vast sums of money to purchase frequency bands from the Office of Communications[1] (known as OFCOM) [118]. This demonstrates the value of using the spectrum as efficiently as possible. However, the radio spectrum from which frequencies can be allocated is highly limited. For example, OFCOM limits frequencies

[1] Office of Communications, commonly known as OFCOM, is the government-approved regulatory and competition authority for the broadcasting, telecommunications and postal industries of the UK.

from 9 KHz to 275 GHz [118]. Therefore, this is a good opportunity for researchers to develop algorithms to improve the efficiency of allocating frequencies from a limited radio spectrum.

This has led to considerable academic interest in variants of the frequency assignment problem (FAP). These problems generally involve assigning frequencies to wireless communication connections (known as requests) while satisfying a set of constraints and optimizing a given objective. In this thesis, two variants of the FAP, namely the static and the dynamic FAPs, are studied. Research has mostly focused on the static FAP, where all features of this problem are known at the beginning and do not change over time. More recently, a new variant of the FAP, known as the dynamic FAP, was proposed in [55]. This problem is based on a military application in which features of the problem change over time. The dynamic FAP has to be solved in real time and therefore, the computational time required by any solution method is of particular importance, unlike the static FAP.

In the first part of this thesis, several heuristic algorithms are investigated and developed to solve the static FAP. These heuristic algorithms include some meta-heuristics and a hyper heuristic. Meta-heuristics can be defined as high-level frameworks for designing and developing heuristic algorithms to find high quality solutions [141]. In contrast, a hyper heuristic can be defined as a master process that controls other heuristics to produce high quality solutions. The main difference between meta-heuristics and hyper heuristics is that meta-heuristics work on a solution space, while hyper heuristics work on a space of heuristics. The use of heuristic algorithms is justified as the static FAP is NP-complete [66].

This thesis aims to compare meta-heuristics from different classes, where there are several classifications for meta-heuristics in the literature. One of these is introduced in [119], which classifies meta-heuristics into three classes as follows:

- *Construction-based algorithms*: these algorithms construct new solutions from scratch. Examples of construction-based algorithms are ant colony optimization [45] and greedy randomised adaptive search procedure [59].

- *Population-based algorithms*: these algorithms involve populations of solutions, where different parts of the solution space are searched simultaneously. Examples of these algorithms are ant colony optimization and genetic algorithms [91], and scatter search [70].
- *Local search-based algorithms*: these algorithms solve problems by moving from one solution to another in the solution space. There are many such algorithms, including tabu search [71], simulated annealing [98], noising algorithms [28], threshold acceptance [52], and variable neighbourhood search [112].

Moreover, there are other classifications of meta-heuristics in the literature such as [12], which suggests that meta-heuristics can be classified in different ways according to different characteristics. For example, meta-heuristics can be classified as nature-inspired versus non-nature inspired, population-based versus single point search, dynamic objective function versus static objective function, one neighbourhood versus multi-neighbourhood structures, and memory usage versus memory-less algorithms.

In the second part of this thesis, various approaches are designed to solve the dynamic FAP. The best heuristic algorithms considered in the first part of this thesis are used to construct these approaches. Hence, it is interesting to investigate whether heuristic algorithms which work well on static problems also prove efficient on the dynamic problems. Furthermore, this thesis proposes a novel approach to solve the static FAP by modelling it as a dynamic FAP.

1.2 Overview of Research Presented in this Thesis

In this thesis, three different heuristic algorithms are evaluated and developed, namely tabu search (TS), ant colony optimization (ACO) and hyper heuristic (HH), to solve the static FAP. They are chosen to represent different characteristics of heuristic algorithms in order to identify the most appropriate solution method for such problem. As the static FAP can be modelled as a graph colouring problem [83], existing knowledge of this underlying model can be used to guide the implementation of the selected heuristic algorithms. These heuristic algorithms are assessed using public benchmark datasets of the static FAP which are denoted by CELAR and GRAPH.

TS and ACO represent two different classes of meta-heuristics, where TS represents a class of the local search-based algorithms and ACO represents a construction-based algorithm which incorporates a learning component. TS can be described as a neighbourhood search algorithm which uses memory in the form of a tabu list to restrict the choices of the next solution in order to prevent the search from returning to previously visited solutions. This algorithm has proved extremely successful on a wide range of problems (see e.g. [75]).

ACO has been inspired from the natural behaviour of real ant colonies. Ants are social insects which co-operate using indirect communication to find the shortest path between food sources and the nest. Hence, ACO can also be thought of as a population-based algorithm. Although the most well-known population algorithm is the genetic algorithm, we choose to evaluate ACO instead for several reasons. One of these reasons is that there is little evidence from the literature that genetic algorithms have proved successful on the static FAP. This may be because combining parts of two different high quality FAP solutions may not lead to a new high quality solution and indeed, may not even produce feasible solutions. Moreover, ACO may be well suited to a dynamic environment because it contains a natural learning process.

HH represents a different characteristic of heuristic algorithms which work at a higher level. It is based on the idea that each heuristic has strengths and weaknesses, and therefore combining several heuristics may lead to an improved algorithm capable of solving a wide range of problems.

In this study, TS, ACO and HH for the static FAP are compared and the best performing ones are used to construct various approaches to solve the dynamic FAP. This problem has received little attention so far in the literature compared with the static FAP. The main feature of the dynamic FAP is that new requests become known over a period of time and frequencies need to be assigned effectively and promptly. Therefore, the dynamic FAP can be considered as a set of sub-problems, where each sub-problem is considered in turn. Furthermore, a novel approach based on the concept of the dynamic FAP is proposed to solve the static FAP through modelling it as a dynamic FAP. It may be more profitable to portion the static FAP into smaller sub-problems to be solved consecutively than solving it as a whole. This study investi-

gates whether using this novel approach to solve the static FAP leads to better results compared with other algorithms considered in this study and in the literature.

1.3 Overview of Time Complexity and Computational Complexity

Time complexity of an algorithm shows the relationship between the computing time and the problem size [136]. Commonly, the time complexity of an algorithm is expressed using a big *O* notation, which is a mathematical representation for asymptotic upper bounds. Let $f: \mathbb{N} \to \mathbb{N}$ and $g: \mathbb{N} \to \mathbb{N}$ be functions, we say that $f(n) = O(g(n))$ if and only if there exists a constant $c > 0$ and a non-negative integer n_0 such that for all $n \geq n_0$, we have $f(n) \leq c\, g(n)$. Hence, when $f(n) = O(g(n))$, this means f grows no faster than g. When an algorithm takes a maximum of $O(g(n))$ time to solve an instance of a problem for some polynomial $g(n)$, we say this algorithm has polynomial time complexity. For more information about this topic, we refer the reader to [99, 136].

In computational complexity theory, computational problems can be classified as P, NP, NP-hard and NP-complete (among other classes). A problem is classified as P if it can be solved in polynomial time. In other words, the class P consists of problems that can be solved by an algorithm in time $O(n^k)$ for some constant k, where n is the size of the input to the problem. The class NP refers to non-deterministic polynomial time, which involves a non-deterministic computer. A non-deterministic computer is a theoretical tool that makes a non-deterministic (probabilistic) choice at each point in the computation, while a deterministic computer gives the same result for the same input. The class NP consists of decision problems, which can be solved in polynomial time using a non-deterministic computer and their solution can be verified for correctness in polynomial time on a deterministic computer. Since P problems take polynomial time to be solved, P is a subset of NP. A problem is classified as NP-hard if solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time. In other words, a problem is classified as NP-hard if every problem in NP can be reduced to it in polynomial time. Some NP-hard problems are also in NP. Such problems are called NP-complete problems. Therefore, the class NP-complete consists of the most difficult problems in the NP class. For more information about this topic, we refer the reader to [66, 136, 144].

1.4 Overview of the Frequency Assignment Problem

The main concept of the FAP is assigning a frequency to each request while satisfying a set of constraints and optimizing a given objective function. In fact, the FAP is not a single problem. Rather, there are variants of the FAP that are encountered in practice. Overall, the FAP can be defined formally as follows: given

- a set of requests $R = \{r_1, r_2, \dots, r_{NR}\}$, where *NR* is the number of requests,
- a set of frequencies $F = \{f_1, f_2, \dots, f_{NF}\} \subset \mathbb{Z}^+$, where *NF* is the number of frequencies,
- a set of constraints related to the requests and frequencies,
- an objective function,

the goal is to assign one frequency to each request so that the given set of constraints are satisfied and the objective function is optimized. In this thesis, the frequency that is assigned to request r_i is denoted as f_{r_i}.

In the next subsections, different types of constraints of the FAP are introduced. This is followed by the description of the two variants of the FAP, namely the static and the dynamic FAPs.

1.4.1 Constraints of the FAP

There are four main constraints in the FAP, which can be hard or soft depending on the variant of the FAP. Hard constraints must be satisfied while soft constraints are not required to be satisfied, but should be if possible. These constraints can be described as follows:

1) Bidirectional constraints: this type of constraint forms a link between each pair of requests $\{r_{2i-1}, r_{2i}\}$, where $i = 1, \dots, NR/2$. In these constraints, frequencies $f_{r_{2i-1}}$ and $f_{r_{2i}}$ that are assigned to requests r_{2i-1} and r_{2i}, respectively, should be distance d_c apart, where d_c is a given constant. In the datasets considered here (see Section 1.5), d_c is always equal to a constant value (238). These constraints can be written as follows:

$$\left| f_{r_{2i-1}} - f_{r_{2i}} \right| = d_c \qquad \text{for } i = 1, \dots, NR/2 \tag{1.1}$$

2) Interference constraints: this type of constraint forms a link between a pair of requests $\{r_i, r_j\}$, where the frequencies f_{r_i} and f_{r_j} that are assigned to the requests r_i and r_j, respectively, should be more than distance $d_{r_i r_j}$ apart, where $d_{r_i r_j}$ is a given constant. These constraints can be written as follows:

$$\left|f_{r_i} - f_{r_j}\right| > d_{r_i r_j} \qquad \text{for } 1 \leq i < j \leq NR \tag{1.2}$$

3) Domain constraints: the set of available frequencies for each request r_i is denoted by the domain $D_{r_i} \subset F$, where $\cup_{r_i \in R} D_{r_i} = F$. Hence, the frequency which is assigned to r_i must belong to D_{r_i}. This type of constraints is always hard in all of the variants of the FAP.

For the datasets considered in this thesis, there are 7 available domains and hundreds of requests, which mean more than one request share the same domain. Moreover, each pair of requests $\{r_{2i-1}, r_{2i}\}$, where $i = 1, \ldots, NR/2$, has the same domain. Some frequencies belong to more than one domain and so can be assigned to requests which belong to different domains. Table 1.1 shows the 7 domains that are used in all of the benchmark datasets considered in this study.

Domain	No. of frequencies in the domain	Frequencies
1	44	16 30 44 58 72 86 100 114 128 142 156 254 268 282 296 310 324 338 352 366 380 394 414 428 442 456 470 484 498 512 526 540 554 652 666 680 694 708 722 736 750 764 778 792
2	22	30 58 86 114 142 268 296 324 352 380 414 442 470 498 526 554 652 680 708 736 764 792
3	36	30 44 58 72 86 100 114 128 142 268 282 296 310 324 338 352 366 380 428 442 456 470 484 498 512 526 540 666 680 694 708 722 736 750 764 778
4	24	16 30 58 86 114 142 254 268 296 324 352 380 414 442 470 498 526 554 652 680 708 736 764 792
5	6	142 170 240 380 408 478
6	42	30 44 58 72 86 100 114 128 142 156 268 282 296 310 324 338 352 366 380 394 414 428 442 456 470 484 498 512 526 540 554 652 666 680 694 708 722 736 750 764 778 792
7	22	16 30 44 58 72 86 100 114 128 142 156 254 268 282 296 310 324 338 352 366 380 394

Table 1.1: The domains in the datasets considered in this thesis.

4) Pre-assignment constraints: certain requests r_i have already been pre-assigned to given frequencies g_{r_i}. These constraints can be written as follows:

$$f_{r_i} = g_{r_i} \tag{1.3}$$

Example 1.1 clarifies the general concept of the FAP and the different types of constraints.

Example 1.1:

Consider an FAP instance that consists of 10 requests, 10 frequencies and 3 domains as shown in Table 1.2.

Domain	No. of frequencies in the domain	Frequencies
1	4	16 254 100 338
2	4	114 352 428 666
3	6	428 666 100 338 540 778

Table 1.2: The domains considered in Example 1.1.

The domain and the pre-assignment constraints for each request are given in Table 1.3. In this example, there are pre-assignment constraints for only requests r_7 and r_8.

Request	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
Domain constraints	2	2	1	1	3	3	1	1	2	2
Pre-assignment constraints	-	-	-	-	-	-	16	254	-	-

Table 1.3: The domain and pre-assignment constraints considered in Example 1.1.

The bidirectional and the interference constraints are given in Table 1.4.

Bidirectional constraints	Interference constraints
$\lvert f_{r_1} - f_{r_2} \rvert = 238$	$\lvert f_{r_1} - f_{r_3} \rvert > 9$
$\lvert f_{r_3} - f_{r_4} \rvert = 238$	$\lvert f_{r_3} - f_{r_5} \rvert > 7$
$\lvert f_{r_5} - f_{r_6} \rvert = 238$	$\lvert f_{r_4} - f_{r_7} \rvert > 20$
$\lvert f_{r_7} - f_{r_8} \rvert = 238$	$\lvert f_{r_8} - f_{r_{10}} \rvert > 80$
$\lvert f_{r_9} - f_{r_{10}} \rvert = 238$	

Table 1.4: The bidirectional and the interference constraints considered in Example 1.1.

Figure 1.1 presents the FAP instance considered in Example 1.1. Each request is represented by a node, and each bidirectional or interference constraint is represented by an edge. Additionally, each request is given a colour which indicates the domain constraint of that request.

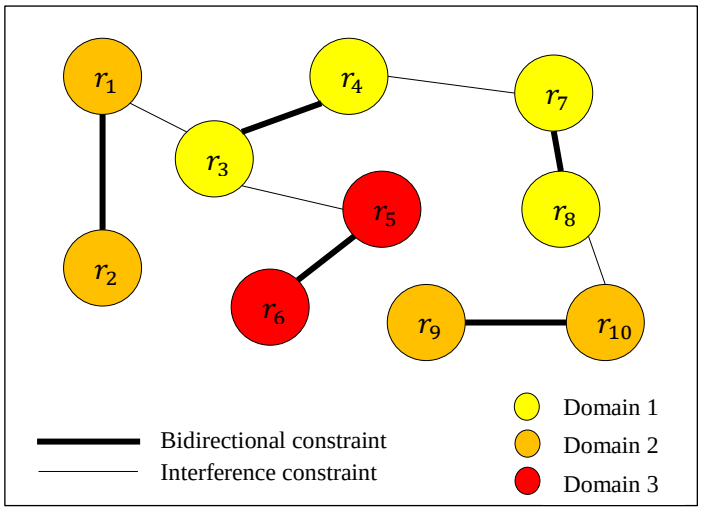

Figure 1.1: The FAP instance considered in Example 1.1.

A feasible solution of the considered FAP instance, which uses 8 frequencies, is given in Table 1.5. Note that a bold number means a pre-assignment constraint.

Request	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
Assigned frequency	666	428	100	338	778	540	**16**	**254**	428	666

Table 1.5: A feasible solution for the problem in Example 1.1

1.4.2 The Static FAP

In the static FAP, all features are known at the beginning and do not change over time. This problem has three main variants, namely the minimum order FAP (MO-FAP), the minimum span FAP (MS-FAP) and the minimum interference FAP (MI-FAP). These variants have different objectives and also differ in terms of whether the constraints (see Section 1.4.1) are considered hard or soft. The formal definitions for each of these are given below as presented in [101]. To help define these problems, the set $T_{r_i r_k} \subset \mathbb{Z}$ is defined as the set of invalid distances between the frequencies f_{r_i} and f_{r_k} that are assigned to the requests r_i and r_k, respectively, based on bidirectional or interference constraints, as follows:

$$T_{r_i r_k} = \begin{cases} \mathbb{Z} \setminus \{d_c\} & if\ r_i\ and\ r_k\ are\ linked\ by\ a\ bidirectional\ constraint \\ \{0, 1, \dots, d_{r_i r_k}\} & if\ r_i\ and\ r_k\ are\ linked\ by\ an\ interference\ constraint \end{cases}$$

Moreover, let C be the set of pairs of requests $\{r_i, r_k\}$ for which there exist bidirectional or interference constraints.

i) The Minimum Order FAP: all types of the constraints are hard in the MO-FAP, and the objective is to minimize the number of used frequencies. The following integer linear programming formulation for this problem is given in [1]. For every request r_i and available frequency f_j , a binary variable $x_{r_i f_j}$ is given by:

$$x_{r_i f_j} = \begin{cases} 1 & if\ frequency\ f_j \in D_{r_i}\ is\ assigned\ to\ request\ r_i \\ 0 & otherwise \end{cases}$$

Moreover, another binary variable y_{f_j} indicates the use of frequency f_j as follows:

$$y_{f_j} = \begin{cases} 1 & if\ frequency\ f_j \in F\ is\ used \\ 0 & otherwise \end{cases}$$

Then, the problem can be formulated as follows:

$$min \sum_{f_j \in F} y_{f_j} \tag{1.4}$$

$$s.t. \sum_{f_j \in D_{r_i}} x_{r_i f_j} = 1 \qquad \forall\, r_i \in R \tag{1.5}$$

$$x_{r_i f_j} + x_{r_k f_l} \leq 1 \qquad \forall\, \{r_i, r_k\} \in C, f_j \in D_{r_i}, f_l \in D_{r_k}: \ |f_j - f_l| \in T_{r_i r_k} \tag{1.6}$$

$$x_{r_i f_j} \leq y_{f_j} \qquad \forall\, r_i \in R, f_j \in D_{r_i} \tag{1.7}$$

$$x_{r_i f_j} \in \{0,1\} \qquad \forall\, r_i \in R, f_j \in D_{r_i} \tag{1.8}$$

$$y_{f_j} \in \{0,1\} \qquad \forall\, f_j \in F \tag{1.9}$$

The objective (1.4) counts the number of used frequencies. The constraint (1.5) means each request has one frequency. The constraint (1.6) gives the bidirectional, the interference and the domain constraints. The constraint (1.7) ensures that if a frequency is assigned to a request, then the corresponding y variable is set to one.

ii) The Minimum Span FAP: all types of the constraints in the MS-FAP are hard, and the objective is to minimize the difference between the maximum and minimum used frequencies, where this difference is called the span. An integer linear programing formulation for this problem is given in [3] as follows:

$$\min\ (z_{max} - z_{min}) \tag{1.9}$$

$$s.t. \sum_{f_j \in D_{r_i}} x_{r_i f_j} = 1 \qquad \forall\, r_i \in R \tag{1.10}$$

$$x_{r_i f_j} + x_{r_k f_l} \leq 1 \qquad \forall\, \{r_i, r_k\} \in C, f_j \in D_{r_i}, f_l \in D_{r_k}: \left|f_j - f_l\right| \in T_{r_i r_k} \tag{1.11}$$

$$x_{r_i f_j} \leq y_{f_j} \qquad \forall\, r_i \in R, f_j \in D_{r_i} \tag{1.12}$$

$$z_{max} \geq f_j\, y_{f_j} \qquad \forall\, f_j \in F \tag{1.13}$$

$$z_{min} \leq f_{max} - (f_{max} - f_j)\, y_{f_j} \qquad \forall f_j \in F \tag{1.14}$$

$$x_{r_i f_j} \in \{0,1\} \qquad \forall\, r_i \in R, f_j \in D_{r_i} \tag{1.15}$$

$$y_{f_j} \in \{0,1\} \qquad \forall\, f_j \in F \tag{1.16}$$

$$z_{max}, z_{min} \in \mathbb{Z}^+ \tag{1.17}$$

Here, z_{max} and z_{min} are the maximum and minimum used frequencies, respectively, and f_{max} is the maximum frequency in the set *F*. The constraints (1.13) and (1.14) guarantee that z_{max} and z_{min} are set to the right values. Other constraints are defined as for the MO-FAP.

iii) The Minimum Interference FAP: there is a combination of hard and soft constraints in the MI-FAP. All the bidirectional constraints are hard, whereas all the interference constraints are soft. In contrast, the pre-assignment constraints can be hard or soft depending on the instances. The soft constraints are given weights which indicate the cost of breaking those constraints (also called violation). The violation cost is weighted by a given penalty value $p_{r_i r_k f_j f_l} \in \mathbb{Z}^+$ for each broken soft constraint, where $\{r_i, r_k\} \in C$, $f_j \in D_{r_i}$ and $f_l \in D_{r_k}$. The objective of the MI-FAP is to minimize the weighted sum of violation costs.

To give an integer liner programing formulation for the MI-FAP as in [3], a new binary variable $z_{r_i r_k f_j f_l}$ is introduced for all $\{r_i, r_k\} \in C$, $f_j \in D_{r_i}$, $f_l \in D_{r_k}$ with $\left|f_j - f_l\right| \in T_{r_i r_k}$.

$$z_{r_i r_k f_j f_l} = \begin{cases} 1 & if\ both\ x_{r_i f_j} = 1\ and\ x_{r_k f_l} = 1 \\ 0 & otherwise \end{cases}$$

Then, this problem can be formulated as follows:

$$min \sum_{\{r_i,r_k\}\in C} \sum_{\substack{f_j\in D_{r_i}, f_l\in D_{r_k} \\ |f_j-f_l|\in T_{r_ir_k}}} p_{r_ir_kf_jf_l}\ z_{r_ir_kf_jf_l} \tag{1.17}$$

$$s.t. \sum_{f_j\in D_{r_i}} x_{r_if_j} = 1 \qquad \forall\, r_i \in R \tag{1.18}$$

$$x_{r_if_j} + x_{r_kf_l} \leq 1 + z_{r_ir_kf_jf_l} \qquad \forall \{r_i, r_k\} \in C, f_j \in D_{r_i}, f_l \in D_{r_k}: \ |f_j - f_l| \in T_{r_ir_k} \tag{1.19}$$

$$x_{r_if_j} \in \{0,1\} \qquad \forall\, r_i \in R, f_j \in D_{r_i} \tag{1.20}$$

$$z_{r_ir_kf_jf_l} \in \{0,1\} \qquad \forall \{r_i, r_k\} \in C, f_j \in D_{r_i}, f_l \in D_{r_k}: \ |f_j - f_l| \in T_{r_ir_k} \tag{1.21}$$

Constraint (1.19) states that the variable $z_{r_ir_kf_jf_l}$ is equal to 1 if and only if both of the frequencies f_j and f_l are assigned to the requests r_i and r_k, respectively, which consequently adds a further penalty $p_{r_ir_kf_jf_l}$ to the sum in the objective (1.17).

1.4.3 The Dynamic FAP

The dynamic FAP changes over time as new requests gradually become known and frequencies need to be assigned to those requests effectively and promptly while satisfying a set of constraints (see Section 1.4.1), which are all hard in the dynamic FAP considered in this study. This problem can be considered as a set of sub-problems to be solved consecutively. The objective of the dynamic FAP is to find a feasible solution with the minimum number of re-assigned requests.

1.5 Overview of the Datasets

The heuristic algorithms considered in this thesis are assessed using two types of datasets, which are static and dynamic FAP datasets. The static FAP datasets are publically available, which can be found on the static FAP website[1]. However, no dynamic FAP datasets are publically available, so for the purpose of this study, new dynamic FAP datasets have been generated from the static FAP datasets. Moreover, the new dynamic FAP datasets have been made available for other researchers, which can be

[1] http://fap.zib.de/problems/CALMA/ (last accessed 25 February 2015).

found on the dynamic FAP website[1]. In this section, information about the static FAP datasets is presented, while the generated dynamic FAP datasets are discussed in more detail in Chapter 6.

The static FAP datasets are denoted by CELAR and GRAPH. CELAR was provided by the Centre d'Electronique de l'Armement in France. This dataset is based on real life problems and has 11 instances. In contrast, GRAPH (Generating Radio Link Frequency Assignment Problem Heuristically) was randomly generated in [146] and has 14 instances. Overall, CELAR and GRAPH are widely used to test different algorithms in the literature, and are widely accepted as benchmarks for the static FAP. The numbers of requests and constraints for CELAR and GRAPH instances are given in Table 1.6.

Instance	Variant of the static FAP	No. of requests	No. of bidirectional constraints	No. of interference constraints	No. of domain constraints	No. of pre-assignment constraints	Total no. of constraints
CELAR 01	MO-FAP	916	458	5,090	916	0	6,464
CELAR 02	MO-FAP	200	100	1,135	200	0	1,435
CELAR 03	MO-FAP	400	200	2,560	400	0	3,160
CELAR 04	MO-FAP	680	340	3,627	400	280	4,647
CELAR 11	MO-FAP	680	340	3,763	680	0	4,783
GRAPH 01	MO-FAP	200	100	1,034	200	0	1,334
GRAPH 02	MO-FAP	400	200	2,045	400	0	2,645
GRAPH 08	MO-FAP	680	340	3,417	680	0	4,437
GRAPH 09	MO-FAP	916	458	4,788	916	0	6,162
GRAPH 14	MO-FAP	916	458	4,180	916	0	5,554
CELAR 05	MS-FAP	400	200	2,398	400	0	2,998
GRAPH 03	MS-FAP	200	100	1,034	200	0	1,334
GRAPH 04	MS-FAP	400	200	2,044	400	0	2,644
GRAPH 10	MS-FAP	680	340	3,567	680	0	4,587
CELAR 06	MI-FAP	200	100	1,222	200	0	1,522
CELAR 07	MI-FAP	400	200	2,665	400	0	3,265
CELAR 08	MI-FAP	916	458	5,286	916	0	6,660
CELAR 09	MI-FAP	680	340	3,763	94	586	4,437
CELAR 10	MI-FAP	680	340	3,763	94	586	4,437
GRAPH 05	MI-FAP	200	100	1,034	200	0	1,334
GRAPH 06	MI-FAP	400	200	1,970	400	0	2,570
GRAPH 07	MI-FAP	400	200	1,970	98	302	2,570
GRAPH 11	MI-FAP	680	340	3,417	680	0	4,437
GRAPH 12	MI-FAP	680	340	3,677	168	512	4,697
GRAPH 13	MI-FAP	916	458	4,815	916	0	6,189

Table 1.6: Details of the CELAR and the GRAPH datasets.

In these datasets, the number of available frequencies (*NF*) is 48 and the number of requests varies between 200 and 916. Considering the computational complexity of these problems and the large numbers of requests and constraints, it is clear that this

[1] https://dynamicfap.wordpress.com/

problem can be difficult to solve. For example, for a moderate size instance of, say, 40 frequencies and 600 requests, there would be up to 40^{600} possible solutions.

1.6 Aims and Structure of this Thesis

The aims of this thesis are as follows:

1) To identify an appropriate solution method for the static FAP.
2) To investigate whether the solution method can prove effective on different variants of the static FAP without significant changes.
3) To determine an appropriate approach to solve the dynamic FAP using the best performing heuristic algorithms in this thesis.
4) To ascertain whether the static FAP can be solved effectively using a novel approach which models the static FAP as the dynamic FAP.

Each of these aims is discussed in more detail as follows:

The first and second aims: in order to identify an appropriate solution method for the static FAP, three different heuristic algorithms are selected to be developed and evaluated. Each of them represents different characteristics of heuristic algorithms. The selected heuristic algorithms in this thesis are tabu search (TS), ant colony optimization (ACO) and hyper heuristic (HH). Several novel and existing techniques are used to improve the performance of these heuristic algorithms to solve the static FAP. The selected heuristic algorithms are mainly designed to solve the MO-FAP. Then, it is of interest to investigate whether these heuristic algorithms can be effective on the other variants of the static FAP without significant changes.

In this thesis, TS is considered first as it is one of the most popular meta-heuristic algorithms and has achieved competitive performance on a variety of problems. ACO is considered next as it is also a meta-heuristic algorithm but it represents a different class of heuristic algorithms. Additionally, many ACO implementations include a local search, so the findings from the work on TS may be helpful. Finally, we consider HH, which involves combining several heuristics, so findings from the previous algorithms may influence this work.

In terms of TS, several novel and existing techniques are used to improve the performance of this algorithm and make it different from other TS algorithms in the litera-

ture. One of the techniques is hybridising TS with multiple neighbourhood structures, one of which is used as a diversification technique. Another novel technique aims to determine a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model. These lower bounds ensure that the search focuses on parts of the solution space that are likely to contain feasible solutions. Moreover, TS is compared in two configurations, where one relaxes the interference constraints, and the other relaxes both the bidirectional and the interference constraints. Some research questions are raised for TS as follows:

- *Is TS an effective solution method for the static FAP?*
- *Is it beneficial to hybridise TS with multiple neighbourhood structures?*
- *Can TS without significant changes be effective on different variants of the static FAP?*

In terms of ACO, some of the key factors in producing a high quality ACO implementation are examined such as different definitions of visibility and trails, and the values of numerous parameters. Moreover, the concept of a well-known graph colouring algorithm, namely recursive largest first, is combined with ACO in order to improve the performance of this algorithm. We also attempt to improve ACO by combining it with a local search. Several research questions are raised for ACO as follows:

- *Can ACO perform better than TS on the static FAP?*
- *Is it beneficial to combine ACO with a local search?*
- *Is ACO an appropriate solution method for the static FAP?*

In terms of HH, simple and advanced low level heuristics (LLHs) associated with an independent tabu list for each LLH are applied in this study. Moreover, a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model, is also used. Several selection mechanisms of the LLHs are discussed and compared. Some research questions are raised for HH as follows:

- *Can HH perform better than TS and ACO on the static FAP?*
- *What is the best mechanism for selecting the LLHs?*
- *Is HH an appropriate solution method for the static FAP?*

The third aim: in order to determine an appropriate solution method for the dynamic FAP, various approaches are designed and constructed using the best performing heuristic algorithms on the static FAP considered in this study. It is interesting to investigate whether heuristic algorithms which work well on the static FAP also prove efficient on the dynamic FAP. Furthermore, several novel and existing techniques are used to improve the performance of the various approaches for solving the dynamic FAP. A novel technique, called the *Gap* technique, aims to identify a good frequency to be assigned to a given request. A research question is raised in this section as follows:

- *Can TS, ACO and HH for the static FAP be successful on the dynamic FAP?*

The fourth aim: in this thesis, a novel approach is proposed to solve the static FAP by modelling it as a dynamic FAP through breaking it down into smaller sub-problems, which are solved consecutively in a dynamic process using the best heuristic algorithm in this study. Moreover, several techniques are applied to improve the performance of this approach such as a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist and the *Gap* technique. Our aim here is to investigate whether using this approach to solve the static FAP leads to better results compared with the heuristic algorithms considered in this study and other algorithms in the literature which solve the static FAP as a whole. A research question is raised in this section as follows:

- *Can the proposed approach that models the static FAP as a dynamic FAP be an effective method for the static FAP?*

Structure of this thesis: this thesis is organized as follows: Chapter 2 provides some background information on the static and the dynamic FAPs and the selected heuristic algorithms in this study. This is followed by Chapter 3, which investigates the TS algorithm for the static FAP. The ACO algorithm is investigated in Chapter 4 and Chapter 5 investigates the HH algorithm for the static FAP. In Chapter 6, the best of these heuristic algorithms are selected to construct various approaches to solve the dynamic FAP, and a novel approach is proposed to solve the static FAP by modelling it as a dynamic FAP. Finally, this thesis is closed with conclusions and ideas for future work.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides background to the study presented in this thesis and puts it in a broader context. Two problems are considered in this thesis: the static and the dynamic frequency assignment problems (FAPs). The static FAP has received more attention so far in the literature compared with the dynamic FAP. The static FAP (defined in Section 1.4.2) has several variants such as the minimum order FAP (MO-FAP), the minimum span FAP (MS-FAP) and the minimum interference FAP (MI-FAP). These problems have been solved in the literature using a variety of solution methods. In this study, three heuristic algorithms are used to solve the static FAP, namely tabu search (TS), ant colony optimization (ACO) and hyper heuristic (HH). Overviews of these algorithms as well as their variations and extensions to solve these problems are given. Additionally, overviews of these algorithms to solve other problems in the literature are presented.

The remainder of this chapter is organised as follows: the next section gives an overview of the static FAP. Section 2.3 provides a general description of the graph colouring problem, which is related to the static FAP. Then, the three heuristic algorithms

are presented in Sections 2.4, 2.5 and 2.6. After that, the dynamic FAP is reviewed in Section 2.7. In Section 2.8, this chapter is closed with some conclusions.

2.2 The Static Frequency Assignment Problem

The static FAP (also known as the static channel assignment problem) has applications in many types of wireless communication networks such as mobile phones, TV broadcasting, radio and military operations. This problem began through radio waves in the late 1800s, which was introduced in [111]. After that, the importance of this technology grew as a result of the increase in the number of wireless applications in the beginning of the 1990s. Since then, a variety of solution models and techniques for variants of the static FAP have been suggested.

More recently, the literature on the static FAP has increased rapidly. This reflects the rapid growth of implementation of satellite communication projects and wireless mobile phone networks. Moreover, important applications of this problem such as military communication and TV broadcasting push the wheel of research quickly. These different applications lead to different models and different instances. Nevertheless, all of them have two common features: a set of requests must be assigned frequencies, and interference occurs when frequencies that are close to each other on the spectrum are assigned to two requests which are linked by a constraint [3].

In the literature, variants of the static FAP were solved using a variety of heuristic algorithms such as tabu search [15, 145], general network algorithm [16], genetic algorithm [94], potential reduction [151], nonlinear approach [150], evolutionary search [34], simulated annealing [145], branch and cut algorithm [1], self-organizing neural network approach [138] and dynamic programming algorithm [101]. Some of these algorithms were applied to the same static FAP datasets considered in this study.

2.3 The Graph Colouring Problem

The graph colouring problem (GCP) is a well-known and widely studied classical optimization problem which has many practical applications. In the literature, the GCP has been solved using a variety of algorithms (see e.g. [30, 49, 56]). The aim of this section is to introduce this type of problem and to show the relationship between the GCP and the static FAP.

The GCP can be described as follows: given a set of vertices *V* and a set of edges *E*, the objective is to allocate a colour to each vertex in such a way that the minimum number of colours is used and no adjacent vertices have the same colour. The minimum number of colours to feasibly colour a graph is known as the chromatic number.

The relationship between the static FAP and the GCP was discussed in [35, 83], which showed that the static FAP is equivalent to a generalization of the GCP. In order to illustrate how the static FAP can be modelled as the GCP, Figure 2.1 presents a sample of a static FAP instance which contains 14 requests (denoted as r_i, $i = 1, \ldots, 14$), 3 frequencies (denoted as f_1, f_2 and f_3) and a set of constraints.

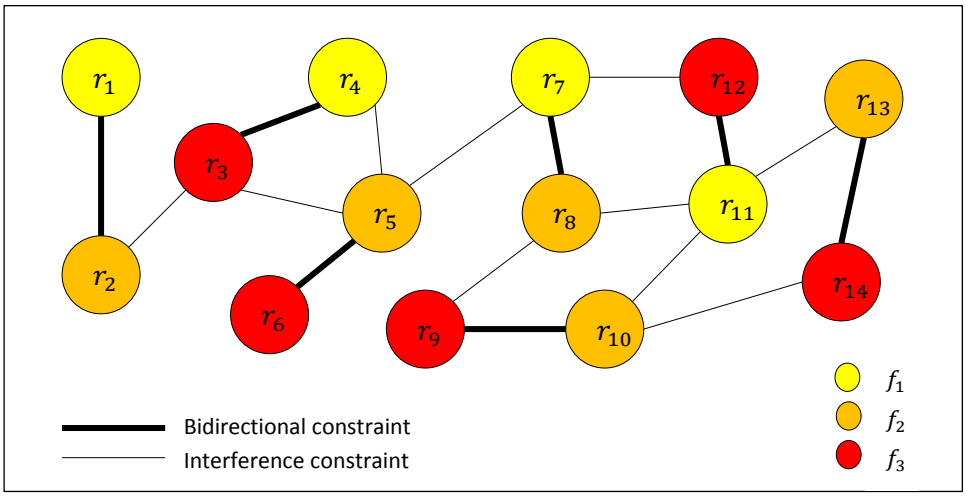

Figure 2.1: A sample of a static FAP instance modelled as a GCP.

In Figure 2.1, each colour represents a frequency and each vertex represents a request. The edges show bidirectional and interference constraints between the requests. For example, there is a bidirectional constraint between r_1 and r_2, and an interference constraint between r_2 and r_3. An edge joining vertices means that the requests associated to those vertices should be assigned frequencies with a certain distance apart. This is similar to the constraints of the GCP (with frequencies viewed as colours).

As a result of this relationship, several existing solution methods for the GCP can also be used to solve the static FAP [131] such as TS in [87], a heuristic algorithm originally developed for the k-colouring problem (KCP)[1] in [123], and a stochastic local search algorithm (a generalization of a local search algorithm) in [30]. In fact, this thesis applies a number of techniques for the GCP in the literature. One of these is the

[1] The KCP can be defined as colouring all vertices in a graph with *K* colours such that the total weight on the edges joining vertices with the same colour is minimized.

recursive largest first (RLF) algorithm, which was proposed in [104]. The RLF algorithm generates a solution as follows: first, a colour is selected, then vertices are sequentially added to this colour until no more can be added feasibly. After that, a different colour is selected to colour as many remaining vertices as possible. This process is repeated until all vertices are coloured. In this study, RLF is hybridised with ACO to improve its performance and makes it different from other ACO implementations for the static FAP in the literature. Moreover, a trail[1] definition of ACO for the GCP in [49] is also applied in our ACO algorithm for solving the static FAP (see Section 4.2.4). Furthermore, the cost function definition of ACO for the GCP in [49] and the solution space definition of TS for the GCP in [87] are used in our TS algorithm for solving the static FAP.

2.4 Tabu Search

Tabu search (TS) is an extension to local search that allows the search to escape from local optima. TS was proposed in [71] as a general meta-heuristic algorithm to solve difficult combinatorial problems. This algorithm has been widely used with much success on a large variety of problems. Therefore, over the last few decades, hundreds of researchers have implemented TS for a variety of combinatorial problems such as [75,120, 130,149]. TS usually finds solutions that are close to the optimal solution, if not the optimal, making it an extremely popular heuristic algorithm.

Built on the idea of steepest descent, TS starts from an initial solution, which may be constructed either randomly or using a greedy heuristic. Then, one of the neighbour solutions (normally the best one) is accepted as the new solution. Unlike steepest descent, TS may accept a neighbour which is worse than the current solution. This may lead to the search cycling through a small group of solutions. Hence, the tabu list is used as a memory structure to record previously visited solutions and to ensure that the search does not return to them. Sometimes, the tabu list is too restrictive by forbidding some attractive moves even when there is no harm of cycling. Therefore, it is essential to use the aspiration criteria to escape from this situation by allowing some neighbours in the tabu list to be accepted when that move leads to a better result than the best result so far. More complicated aspiration criteria in the literature have been proposed and successfully implemented, but are not popular (see e.g. [36, 88]). Unlike

[1] A pheromone trail reflects how good a move is based on the history of successful moves.

other meta-heuristic algorithms such as ACO, TS requires relatively few parameters. The key parameters are the length of the tabu list and the total number of iterations. For more information about TS, the reader is referred to [74, 75].

2.4.1 Tabu Search for the Static FAP

The static FAP is one of the problems which have been solved in several studies using TS. To the best of my knowledge, only two papers [15, 145] applied TS to the same static FAP datasets considered in this study (CELAR and GRAPH). In this section, these two papers and others that use TS for different static FAP datasets are discussed.

The MO-FAP and the MS-FAP (variants of the static FAP, see Section 1.4.1) were solved in [15] using TS. The solution space was defined as the set of solutions which satisfy pre-assignment and domain constraints, which means interference and bidirectional constraints are relaxed. This solution space creates a sub-problem: minimizing the number of violations with a fixed number of used frequencies. TS is used to reduce the number of violations using the best neighbourhood move in each iteration. If a feasible solution is found, then the number of used frequencies is reduced, which may lead to violations. TS is then used again to reduce the number of violations. This process is repeated until one of the stopping criteria is satisfied.

This TS algorithm requires three parameters: the tabu tenure, the patience parameter and the ratio parameter. The tabu tenure is the number of iterations for which a move stays on the tabu list. Although the authors stressed the importance of the tabu tenure, they did not state the value they actually used. The patience parameter is defined as the number of iterations in this algorithm. The ratio parameter (N) is used to reduce the search space to be more efficient. Therefore, this parameter can be seen as a way to restrict a neighbourhood by considering only the set of requests which contribute to the violations by at least $N\%$ of the maximum number of violations. To clarify the ratio parameter, consider Example 2.1.

Example 2.1:

> Consider a static FAP instance which contains 6 requests and the ratio parameter (N) is used to recommend some requests as good candidates to be reassigned to different frequencies. After selecting a value for the ratio parameter, only the requests which contribute to the violations by at least N % of the

maximum number of violations are considered. Table 2.1 gives the number of violations for each request.

Request	r_1	r_2	r_3	r_4	r_5	r_6
Number of violations	8	10	6	0	4	7

Table 2.1: Example of the ratio parameter.

Different values of the ratio parameter suggest different groups of requests to sample as follows:

- If $N = 100$, then only the requests with the maximum number of violations are considered, which is request r_2.
- If $N = 70$, then the requests with at least 70% of the maximum number of violations are considered, which are r_1, r_2 and r_6 .
- If $N = 60$, then the requests with at least 60% of the maximum number of violations are considered, which are r_1, r_2, r_3 and r_6 .
- If $N = 0$, then all requests are considered.

For each selected request, all possible moves are attempted. Any prohibited moves, i.e. those on the tabu list, are excluded unless they satisfy the aspiration criteria. The cost function is the number of violated constraints. Another technique, called constraint variation, is also used in their work. The concept of this technique is to start solving the problem by considering only a subset of the constraints in order to make it easier to be solved. Then, other constraints are added gradually until all of the constraints are considered.

The authors found in [15] that cycling may occur and the same request may cycle between different frequencies despite the tabu list. In order to avoid this problem, they suggested using another tabu list based on frequencies, meaning a request cannot be re-assigned to any frequency for a stated number of iterations. This may make all the possible moves tabu. In this case, the neighbourhood is extended by decreasing the ratio parameter by 20. If that does not help, then it is decreased by a further 20. (For example, if the initial value of *N* is 100, it reduces to *N* = 80, and then *N* = 60). If that still does not help, then it is decreased to zero.

The authors also found in [15] that the TS algorithm is very poor in terms of minimizing the number of used frequencies for a feasible solution. Therefore, they presented a

new method to generate an initial solution using the starting point selection strategy. The main aim of this strategy is to start with the minimum possible number of frequencies even if that leads to an infeasible solution with a high number of violations. Since there are two frequencies that are available in all domains in the considered instances, it is possible to assign all requests to these two frequencies. The stopping criteria in [15] are based on two conditions: when it reaches an optimal solution (if known), or when the number of iterations reaches the patience parameter.

The performance of TS in [15] was compared with the performance of the random search (RS) algorithm and the general network algorithm (GENET) in [16]. It was found that TS achieved the optimal solutions in a reasonable time for 5 out of 6 instances of the CELAR dataset. On the other hand, it was found that RS achieved feasible solutions quickly for some instances, whereas TS was better on more difficult instances, but GENET was the fastest.

The MO-FAP, the MS-FAP and the MI-FAP (variants of the static FAP, see Section 1.4.1) were solved using TS, simulated annealing (SA) and variable-depth search in [145]. The differences between TS in [145] and [15] are the relaxation of the interference constraints and the use of a back-tracking mechanism, which was proposed in [116]. The main concept of this mechanism is that when the algorithm could not improve the quality of the solution for a large number of iterations, the search re-starts from the best found solution. The performance of TS for the MO-FAP in [145] is slightly better compared with SA and variable-depth search. However, TS for the MI-FAP was abandoned due to poor performance. On the other hand, variable-depth search was the fastest in [145].

The static FAP was solved using TS in [85] using a dataset provided by the French national research centre for telecommunications. This dataset is not the same as either CELAR or GRAPH and its source was not specified (hence, it is not considered in this thesis). Here, the same solution space in [15] and hence the same sub-problem are considered. An initial solution was randomly generated for TS in [85] and improved using the standard local search. After that, a randomly selected request is re-assigned to a new frequency. The main technique used in [85] was that when any moves have been accepted more frequently than expected, these moves are prohibited for some time to diversify the search. The performance of TS in [85] was compared with con-

straint programming, graph colouring algorithm and SA. It was found that TS performed similarly to SA and better than the others. Additionally, TS proved quicker on most of the instances, but was markedly slower on the more complicated instances.

The reactive tabu search (RTS) algorithm is an extension of TS, which was used in [78] to solve the static FAP. The key improvement of RTS is adapting the tabu list size to properties of the optimization problem. If it seems that the search is cycling, then the tabu tenure is increased. It was found in [78] that RTS performed worse with long term memory (which is used to count the number of times each move is selected and to ensure that moves that are selected too frequently are penalized) than without it. The reason behind that is using the long term memory also increases the tabu tenure for popular moves, which then over-constrains the search space. Overall, RTS improved the performance of TS as shown in [78].

Two algorithms, namely TS and local search (LS), were proposed to solve the static FAP with polarization (FAPP) in [63]. The FAPP is different from the static FAP since polarities need to be assigned to each request and frequencies. Additional constraints govern how these polarities should be assigned. The dataset which was used in their study is based on the topic of the 2001 international challenge organized by the French operations research society ROADEF in collaboration with CELAR. Some techniques were used to reduce the size of the search space such as an adaptive jumping procedure (AJP) and filtering pre-processing.

The AJP is a diversification technique used to overcome the problem of being trapped in a suboptimal area of the search space. AJP is based on two components, which are LS and the jumping operator. The role of the jumping operator is to move from the current position in the search space to a new position by choosing randomly a fixed proportion of the requests and changing their frequencies randomly. The size of the jump can be changed based on the history of the search process. AJP is operated periodically during the search and each time, there are three possible cases. In the first case, when the cost of the current solution is the same as the best solution, then it is necessary to widen the search by increasing the jumping operator. In the second case, if the cost is worse than the best solution, the process is jumping too far, so the jumping operator needs to be decreased. In the last case, when the solution has been improved, then the jumping operator remains the same and the new solution is stored as

the best solution so far. AJP stops as soon as the search achieves the optimal solution (if known) or if it achieves the maximum number of iterations.

Additionally, the filtering pre-processing was used to reduce the size of the search space by eliminating some frequencies. These frequencies are deleted if it is known that these do not belong to any solution which satisfies all constraints. Such deletion is done using a constraint programming technique called arc-consistency [10,113]. It was found that TS for the FAPP in [63] achieved the best known feasible solutions for all the instances of the ROADEF Challenge 2001.

An alternative TS algorithm for the static FAP was introduced in [86] which has several features. These include a candidate list strategy, bidirectional constraints handling strategy, incremental evaluation and dynamic tabu tenure. The search space in [86] relaxes some constraints. The static FAP dataset in [86] was provided by the French national research centre for telecommunications.

The candidate list strategy is a well-known strategy which aims to limit neighbours through updating a list that contains the best moves for requests that are currently involved in violations, which reduces the run time. The bidirectional constraints handling strategy aims to find solutions that satisfy the bidirectional constraints (see Equation 1.1) and ensure they are satisfied throughout the search. Since these constraints are always satisfied, they can be ignored in the cost function (which counts the number of violations). As a result, requests can be considered as pairs (instead of individually) based on the bidirectional constraints. This leads to a reduction in the size of the search space. Moreover, since the number of neighbours considered might be very large, a fast neighbour evaluation was implemented in [86] using an incremental technique proposed in [61]. This technique aims to evaluate each move by considering only the constraints affected by this move. Moreover, a dynamic tabu tenure was applied in [86], where the tabu tenure is adjusted during the search according to the size of the candidate list. Additionally, too large or small tabu tenures are avoided by introducing maximum and minimum bounds.

The performance of TS in [86] was compared with other algorithms such as constraint programming (CP), graph colouring algorithm (GCA) and simulated annealing (SA). It was found that TS and SA performed better than CP and GCA, while TS performed better than SA.

It was found in [155] that TS is one of the best algorithms for the static FAP in terms of solution quality, but not in run time. Hence, an improved TS algorithm was introduced in [155] to improve the run time of this algorithm. One of the factors that help to speed up TS is the way an initial solution is constructed. Therefore, it was found in [155] that a high quality initial solution helps the search find good solutions faster. Moreover, a candidate list strategy was also used in [155] to speed up the search. One of the ideas for speeding up the search is that there is no need to re-calculate the cost function at each iteration. Therefore, after the cost function of the initial solution was calculated, it needed only to be updated by considering the constraints affected by the new move. Also, a single-frequency violation technique was used in [155] to record the violations in order to make the search efficient. Using the single-frequency violation technique with other techniques to improve the run time of the TS algorithm in [155], a solution was obtained in a high speed compared with other algorithms and had almost the same quality as the solutions which have been found in earlier work.

The heuristic manipulation technique (HMT) was used in combination with TS to solve the static FAP in [114]. The key idea of HMT is to change the search space by creating artificial constraints based on the information provided by TS. Hence, during TS implementation, common features of high quality solutions are observed. In particular, requests that tend to be assigned to different frequencies in good solutions are recorded. Artificial constraints are then added to the original problem to force these requests to be allocated to different frequencies. However, these artificial constraints may lead to suboptimal solutions. Hence, artificial constraints are only included for a certain period of time and can be replaced by others or removed. It was found in [114] that HMT improved the performance of TS. Moreover, HMT is not only applicable to TS for the static FAP, but it can be applied to other algorithms and other problems.

2.4.2 Tabu Search for Other Problems

TS has been implemented successfully on a wide range of problems such as facility location problems [5], scheduling problems [117, 153], traveling salesman problems [9] and vehicle routing problems [7]. In this section, some of these problems which are most often solved by TS are chosen to give a flavour of the success of this algorithm for solving other problems.

An examination timetabling problem was solved in [153] using TS associated with long term memory. This problem is categorised as an assignment problem and can also be considered as a GCP. The description of this problem can be given as follows: assume there are *n* exams and *m* timeslots, each exam has to be assigned to a timeslot to satisfy the given constraints while optimizing a given objective function. The long term memory records the number of times each exam is moved and prevents exams from being moved too frequently. The dataset which has been used to test the TS algorithm in [153] was a real dataset taken from the University of Ottawa in Canada. It was found in [153] that TS performed better when using long term memory than without it. Furthermore, a nurse scheduling problem was solved in [39] using TS based on a dataset taken from the School Hospital at the University of Campinas in Brazil. The TS algorithm in [39] was found to be slightly better and required less run time than a genetic algorithm.

A vehicle routing problem (VRP) with split deliveries was solved in [7] using TS. The VRP can be described as finding the shortest routes for vehicles to serve a group of customers. In the split VRP, each customer can be visited by more than one vehicle, whereas in the classical VRP each customer is visited only once. It was found that the TS algorithm in [7] performed better than the heuristic algorithm suggested in [50].

In general, TS has been successfully applied to solve a large number of problems, but many of these are beyond the scope of this research. For more information, the reader is referred to several publications [68, 75, 125].

2.4.3 Summary of the Tabu Search Literature Review

It can be seen from the literature that there are several motivations for selecting TS as one of the heuristic algorithms considered in this study. One of these is that TS is a flexible meta-heuristic algorithm which has been applied successfully to a wide range of problems, including the static FAP, where TS shows competitive performance compared with other algorithms in the literature. To the best of my knowledge, there are only two published papers [15, 145] that applied TS for solving the static FAP using the datasets considered in this thesis. The TS algorithms in [15, 145] were unable to find the optimal solutions in some instances of the static FAP. Hence, it is interesting to investigate whether we can improve the performance of TS using novel and existing techniques. By analysing the literature of TS, it can be found that there are

some interesting techniques when deciding to implement TS for the static FAP. One of these is attempting to create a feasible initial solution with high quality, which should lead to a more efficient solution method [155]. Another idea is using multiple neighbourhood structures, which make our TS algorithm more efficient and different from existing TS algorithms in the literature. These ideas and others are applied to design an improved TS algorithm in this thesis, which is described in more detail in Chapter 3.

2.5 Ant Colony Optimization

Ant colony optimization (ACO) is inherited from the process of ants seeking a short path between their colony and a source of food. In the 1940s, the first researcher who investigated the social behaviour of insects was Pierre-Paul Grassé [42]. He discovered that these insects are able to react with indirect communication, which was called "significant stimuli", and can be divided into two main types:

- Physical information, which belongs to the natural environment.
- Local information, which belongs to the insects in that area.

In 1992, ACO was proposed by Dorigo [40], where the aim was to find the shortest path between two given points in a graph. Therefore, ACO is considered a young meta-heuristic compared with other algorithms such as tabu search, simulated annealing and evolutionary computation.

ACO belongs to a class of constructive meta-heuristic algorithms. ACO can be viewed as a probabilistic greedy construction algorithm, where the probabilities are adjusted according to the results of previous constructions. ACO is based on a fixed number of ants, where each ant produces a solution. Each construction step is controlled by two factors: the attractiveness (based on the constraints and the objective of a problem), and the pheromone trail level (based on the history of successful construction steps). After all ants complete their solutions, i.e. one generation is complete, then pheromone trails are updated by increasing the level of moves which lead to good solutions and decreasing the trail for moves leading to poor quality solutions. The updated pheromone trails guide ants in the following generations to produce better solutions. The solution produced by each ant may be further improved by a local search.

It was found in [42] that the most successful variants of ACO are ant system (AS), max-min ant system (MMAS) and ant colony system (ACS). Different variants of ACO update the pheromone trail in different ways. AS is the basic ACO algorithm in which all ants provide trail updates. MMAS was proposed in [143] by making several changes to AS as follows:

- The pheromone trail is updated by only the best ant.
- The maximum and the minimum values of the pheromone are limited.

ACS was proposed in [65] by making several changes to AS as follows:

- The local pheromone update is executed for all ants after each construction step. The aim of that update is to diversify the search to produce different solutions.
- An offline pheromone update is executed at the end of the construction process. At the end of each generation, this update is applied for only the best ant.

Although ACO achieves good results, in some cases it is necessary to combine it with another meta-heuristic to achieve even better results. Combining ACO with other algorithms was applied in several studies such as combining ACO with LS [127], with SA [14] and with GA [6].

2.5.1 Ant Colony Optimization for the Static FAP

A small number of researchers have applied ACO to the static FAP. However, to the best of my knowledge, there is only one published paper [109] that implemented ACO to solve the static FAP using the datasets considered in this thesis. This paper is discussed first, followed by other ACO algorithms for different datasets.

A variant of ACO for the static FAP named ANTS was proposed in [109]. This algorithm is an adaption of the original ant system (AS) proposed in [108]. The ANTS algorithm was constructed by adding several modifications to the original AS. One of these modifications was to change the probability formula of selecting each move. In order to avoid repeatedly constructing the same solution, a technique called stagnation avoidance, which evaluates each solution against the last k solutions produced by ANTS, is used. The trail was defined between each request and frequency and updated at each generation. Moreover, the solution produced by each ant was improved by

local search (LS). It was found in [109] that the performance of ANTS was good across many different data instances and the approach was robust.

An extended ACO algorithm (originally designed for the GCP) was applied in [124] to solve the MO-FAP in a clustered mobile ad hoc network (MANET). The trail was defined between each request and frequency and used to determine the next constructive step based on the probabilistic transition rule. Additionally, two different definitions of the visibility were given: based on the degree of each request, i.e. the number of unallocated neighbours of that request, and based on the maximum number of feasible frequencies. It was found in [124] that ACO showed competitive performance.

The MS-FAP was solved in [115] using ANTS, which was proposed in [108] as a variant of ACO. They fixed the available span to a high value and used ANTS to construct a violation free solution. If a feasible solution is found, then the span of available frequencies is decreased and ANTS is used again to find a new solution. The trail, defined between the requests and frequencies, reflects the quality of each move, and the visibility is defined as the number of feasible allocations for each request. Moreover, LS was used to improve the performance of ANTS. It was found in [115] that the performance of the ANTS algorithm was competitive compared with other algorithms described in [137, 152].

An ACO algorithm for the static FAP was presented in [107] using a real dataset based on the global system for mobile (GSM) network. GSM is a digital mobile telephone system that is widely used and considered as a second generation system. GSM uses a variety of time division multiple access and is one of the most widely used of the digital wireless telephony technologies. ACO in [107] is a MMAS, where the trail was defined between requests and frequencies. It was found in [107] that this algorithm did not perform better than an evolutionary algorithm.

2.5.2 Ant Colony Optimization for Other Problems

ACO has attracted many researchers to develop many models to successfully solve a large number of optimization problems such as multi objective optimization [93], dynamic optimization [80], stochastic problems [82], continuous and mixed-variable optimization [139, 140], discrete optimization [44], vehicle routing problems [19], traveling salesman problems, scheduling problems, facility location problems, trans-

portation problems, set covering problems and network flow problems [32]. In this section, the problems that are most often solved by the ACO algorithm are chosen to show the success of this algorithm for other problems.

One of the most frequently solved problems by ACO is the traveling salesman problem (TSP). In this problem, a salesman has to visit a set of cities and the objective is to find the shortest route to visit all the cities only once and return to the first position. The TSP was the first problem to be solved using ACO in [46], where a novel heuristic algorithm called ant system (AS) was proposed. This was followed by many studies aiming to solve the TSP using different variations of ACO. One of these was proposed in [45], which was based on a process called the ant system paradigm (ASP). The main ideas of that paradigm were positive feedback, distributed computation and the use of a constructive greedy heuristic. The purpose of the positive feedback is to speed up the process of finding solutions, the distributed computation is to avoid premature convergence and the greedy heuristic helps find high quality solutions.

ASP was applied in [45] to solve other problems such as quadratic assignment problems, job-shop scheduling problems and asymmetric travelling salesman problems. Three algorithms were implemented in [45] using the ASP to study their strengths and weaknesses. These algorithms were ant-cycle, ant-density and ant-quantity. The difference between these algorithms is that ant-cycle uses global information, that is, ants lay a pheromone trail, which reflects how good a move is based on the history of successful moves, whereas ant-density and ant-quantity use local information. Also, the difference between the ant-density and the ant-quantity algorithms is the method for updating the trail. From the experiments, ant-density and ant-quantity obtained worse results than ant-cycle because of the kind of information which was used to direct the ants. It was found in [45] that ASP was able to find good solutions even for difficult problems. ASP was considered a very promising algorithm and proved to be as good as TS and superior to SA.

An ant colony system (ACS) was applied in [41] to solve the TSP. It was found that ACS performed better compared with other heuristic algorithms such as SA and evolutionary computation. More details of applying the ACO algorithm to solve the TSP can be found in [142].

The ant local search (ALS) algorithm was proposed in [126] for solving the GCP. In this algorithm, each ant is a local search, rather than constructing an entire solution as is standard. A new technique was suggested to reduce the computational effort for each ant using greedy selection and trails. ALS was inspired from TS which has been proposed to solve the k-colouring problem (KCP). It was found in [126] that ALS achieved a competitive performance.

A new ACO algorithm (called ANTCOL) was proposed in [89] to solve the KCP. The main contribution of the proposed algorithm was that each ant gives a colour to a single vertex. Moreover, a trail system and a greedy force technique were used in this algorithm. In the trail system, several parameters such as the evaporation rate and the reinforcement value are set and different values of these parameters are compared. The greedy force technique is a method to remove conflicts which happen when two ants of the same colour are assigned to two adjacent vertices. ANTCOL was compared with other algorithms, namely TS, DSATUR and a hybrid genetic algorithm. It was found in [89] that ANTCOL was much better that DSATUR, whereas TS and the hybrid genetic algorithm were much better than ANTCOL.

The GCP was solved in [58] using a modified ACO which was built on two main bases, namely MMAS and LS. The modified ACO follows the MMAS structure and involves a new probabilistic decision rule and uses LS to improve the performance. This modified algorithm was called the max-min ant system algorithm for the graph colouring problem (MMGC). It was found in [58] that MMGC did not achieve the optimal solutions in most cases, but did produce superior results compared with the results of ANTCOL.

2.5.3 Summary of the Ant Colony Optimization Literature Review

Although ACO has been used successfully to solve various problems, it can be seen from the literature that ACO is not a popular algorithm for solving the static FAP as there are only few pieces of research into this area. To the best of my knowledge, there is only one published paper [109] that implemented ACO to solve the static FAP using the datasets considered in this thesis. Moreover, the published results did not show that ACO is one of the best algorithms for such a problem. Additionally, it is often a relatively time consuming algorithm, which may make it less appropriate for the dynamic FAP. However, it is interesting to include this algorithm in this thesis to

investigate whether it is possible to improve its performance using several novel and existing techniques. Some of the key factors in producing a high quality ACO implementation are investigated such as the visibility definition, trail definition and optimizing numerous parameters. Additionally, in order to improve the performance of ACO, the concept of a well-known graph colouring algorithm, namely recursive largest first, is combined with our ACO algorithm.

2.6 Hyper Heuristics

In 1997, the term hyper heuristic (HH) was proposed in [37]. However, the original concept of HH was discovered in the 1960s [60]. Since 2001, the term and the concept of HH have been seen clearly in the literature. A hyper heuristic is an algorithm that combines multiple heuristics. Therefore, the concept of HH is based on the idea that, as each heuristic has strengths and weaknesses, combining several heuristics might lead to better performance. These heuristics, which are managed by the HH, are called low level heuristics (LLHs). The criteria for choosing one of these at each step of the HH is usually problem independent. Hence, HH is an iterative process of two stages: heuristic selection and move acceptance. The main difference between HH and meta-heuristics is that the implementation of HH always searches within the search space of the heuristics, whereas meta-heuristics search within the search space of the problem.

HH was classified in [22] into two main categories: heuristic selection methodology and heuristic generation methodology. The first methodology uses combinations of pre-defined heuristics, while the second one generates new heuristic algorithms. Each category is classified into two further categories called construction heuristics and perturbation heuristics. The former constructs new solutions from scratch and the latter modifies an existing solution. Moreover, the source of feedback from the search process can be divided into three classifications: online learning hyper heuristics, offline learning hyper heuristics and no-learning hyper heuristics. The first one learns while solving a problem, whereas the second one is a method which learns from a set of training instances, which can be applied to unseen instances. The third one never uses information from the search process.

It was found in [122] that acceptance criteria significantly affected the performance of HH compared with the heuristic selection mechanisms. Therefore, several acceptance criteria were compared in [122], namely all moves (AM), only improving (OI), improving and equal (IE), great deluge (GD) and Monte Carlo (MC). AM accepts all moves, while OI accepts only improving moves and IE rejects only worse moves. GD accepts all moves which are better than or equal to a level computed at each step during the search. MC accepts all improving moves while non-improving moves are accepted based on a dynamically changing probability function; if no improvement can be achieved over a given number of iterations, then the probability is increased. It was found in [122] that GD, MC and IE were the best acceptance criteria.

Moreover, four different frameworks of HH were compared in [122], three of which were proposed in [122]. These frameworks used two different types of heuristics, namely mutational heuristics and hill climbers. The hill climbers aim to produce a better solution, while the mutational heuristics do not normally produce a better solution because they are based on random perturbation. Each framework has a different way of using the mutational heuristics and the hill climbers. The first framework was a traditional one where at each step a mutational heuristic or a hill climber is chosen. In the other three frameworks, a hill climber is used separately to improve the diversity provided by a mutational heuristic. The second framework selects either a mutational heuristic or a hill climber, and then a predefined hill climber is implemented to the solution. Hence, this means the hill climber is used at each step during the search. The third framework always selects a mutational heuristic before a predefined hill climber is used. The final framework uses two hyper heuristics, one for the mutational heuristics and one for the hill climbers. It was found in [122] that each framework performed differently and the third framework performed the best.

2.6.1 Hyper Heuristics for the Static FAP

Few researchers have used HH to solve the static FAP. To the best of my knowledge, there are no published papers using HH to solve this problem using the datasets considered in this thesis. Hence, this is the first attempt to solve such datasets using HH.

The MS-FAP was solved in [96] using a HH which uses a local search based metaheuristic called the great deluge algorithm. The initial solution is produced by a greedy constructive heuristic and then low level heuristics (LLHs) are used to im-

prove the quality of the solution. The selection of LLHs is based on the choice function which ranks each LLH based on the quality of the solution provided by each of them. Each move of a LLH produces a new solution, which is accepted or rejected by the HH. This decision is made by comparing the objective function with a parameter called *Level*. This parameter is set during the initialization stage and will be reduced slowly by another parameter called *DownRate* at each iteration. Therefore, the performance of this algorithm is based on the values of the two parameters, which are the starting value of *Level* and the selected value of *DownRate.* It was found in [96] that HH showed promising performance especially in the limited computational time.

An alternative HH algorithm for the MS-FAP was proposed in [97]. Simple LLHs were used such as delete, add and swap, which are easy, fast and robust. At each iteration, one of the LLHs is randomly selected, then the solution from the selected LLH is accepted by the HH based on one of the following four different acceptance criteria: all moves (AM), only improving (OI), Monte Carlo (MC) (as described earlier), and record-to-record travel (RRT), which is a variant of the great deluge acceptance criteria proposed in [51]. In the RRT criteria, the possibility of acceptance is increased by adding a small value, called *Deviation*, to the cost of the current solution. It was found in [51] that, among the four acceptance criteria, HH performed the best when the RRT acceptance criterion was used.

A parallel hyper heuristic (PHH) algorithm for the static FAP was proposed in [135]. This algorithm combines several LLHs by allocating the best performing LLHs more computation resources. 30 different configurations of LLHs were applied in [135]. Moreover, two HH algorithms with different choice functions and probability selections strategies are compared. The first HH algorithm evaluates the improvement due to each configuration of LLH. The second HH algorithm evaluates the performance of each LLH when operated in parallel with another. It was found in [135] that the second HH algorithm provided slightly better results.

A proposed algorithm for the static FAP in [29] is based on PHH and uses a set of meta-heuristics as LLHs. These meta-heuristics are local search, genetic algorithm, variable neighbourhood search, greedy randomized adaptive search procedure, scatter search and artificial bee colony. PHH manages the LLHs using a probability vector, which reflects the number of times each LLH is selected. Moreover, a minimum limit

of the probability of selecting each LLH is set to make sure no LLH is ignored. In the beginning, the LLHs are expected to be chosen homogeneously. After a period of time, the best solution obtained by each meta-heuristic is compared and the probability vector is updated with meta-heuristics producing the best solutions being rewarded with more computational resources. This algorithm achieved the best results at that time. Moreover, this algorithm was competitive in terms of the run time compared with other results in the literature.

2.6.2 Hyper Heuristics for Other Problems

HH has attracted the attention of researchers in a wide range of areas including operational research, computer science and artificial intelligence. This algorithm is self-adaptive, which means that it can be applied to a wide range of optimization problem without the need for heavy modification. Hence, HH have been used to solve a variety of problems such as bin packing problems, job shop scheduling problems, traveling salesman problems and vehicle routing problems [23]. Moreover, HH demonstrates the ability to achieve extremely good and robust results on a wide range of problems. In this section, some of the problems which are most often solved by HH are chosen to give a flavour of the success of this algorithm for other problems.

The examination timetabling problem was solved by HH in [95], which aims to design a generic system that is able to select the most appropriate heuristics for the given problems. A tabu list was used to control the selection of the LLHs by making the most often applied LLHs tabu to allow other LLHs to be applied. Moreover, several techniques were applied to choose the LLHs: considering all heuristics, considering only non-tabu LLHs and considering LLHs which lead to improved solutions only. It was found that the HH could not beat the best results in the literature. However, the aim of [95] was not to beat the best solution, but to present the ability of the HH to create a good solution across a wide range of problems. Moreover, this algorithm can be applied easily to other problems by only changing the LLHs and the evaluation function.

The course and exam timetabling problem was solved in [25] using HH which is based on graph colouring heuristics and TS (as a high level search algorithm). The aim of this algorithm was to produce a good sequence of LLHs. Moreover, the study in [25] focused on the two search spaces: the heuristic space and the solution space. It

was found in [25] that HH performed better when a larger number of LLHs were used. However, this increased the search space size, which led to an increase in the run time. In general, HH achieved competitive results.

2.6.3 Summary of the Hyper Heuristics Literature Review

It can be seen that HH is increasingly popular and has been used to solve a variety of optimization problems. Furthermore, this algorithm generally gives promising results on a wide range of optimization problems compared with other algorithms. On the other hand, only few researchers have implemented HH for the static FAP. Moreover, to the best of my knowledge, there are no published papers applying a HH to solve the static FAP using the datasets considered in this thesis (CELAR and GRAPH). Hence, this is the first attempt to solve such datasets using a HH.

2.7 The Dynamic Frequency Assignment Problem

Up until recently, research has focused on static problems, where all the data is known in advance, but many real-life problems can be considered as dynamic problems. Research is growing more popular into dynamic variants of optimization problems such as vehicle routing problems [67, 69, 128], where new customers become known during the working day; job shop scheduling problems [121], where unpredictable real-time events such as machine failure and the arrival of new jobs may cause an existing schedule to no longer be feasible; the bin packing problem [31, 79]; scheduling problems [57] and graph colouring problems [148]. The major difficulties of dynamic problems come from ignorance of how the problem is going to change in the future.

In the dynamic FAP, new requests become known over a period of time and frequencies need to be assigned to those requests effectively and promptly. Hence, this problem can be seen as a set of sub-problems to be considered consecutively. In the literature, few pieces of research discussed the dynamic FAP. This includes [54], which concerned the deployment of a Hertzian communication network for military applications provided by Center Electronique de L'armement. Requests and frequencies are considered as pairs based on the bidirectional constraints (see Equation 1.1). The dynamic FAP was divided into three underlying problems, namely the static problem, the online problem and the repair problem, each of which is solved using a different solution phase. The static problem is solved in the beginning using the initial solution

phase, which contains only the first sub-problem. The online problem contains sub-problems, which become known and are solved consecutively using the online assignment phase. In case a sub-problem could not be solved by this phase, then this problem will be solved using the repair phase. This phase aims to solve this problem by re-assigning some requests that have already been previously assigned.

The approach to solve the dynamic FAP in [54] was extended and discussed in more detail in [55]. In terms of the initial solution phase, the static problem was solved using a classical greedy algorithm called the minimum frequency greedy algorithm. If any request could not be assigned, then this approach applies the consistent neighbourhood in tabu search (CN-tabu), which was presented in [147]. This algorithm is a hybrid tabu search algorithm which aims to feasibly assign a set of partial requests under certain constraints in order to find a complete feasible solution. Hence, instead of dealing with complete infeasible solutions, it deals with incomplete feasible ones.

The online problem is considered after solving the static problem. The minimum frequency greedy algorithm is used to deal with every new sub-problem arriving dynamically to the online assignment phase. Each sub-problem is solved without modification of the previous decisions. Two strategies for selecting a feasible frequency were compared, namely the minimum feasible frequency or the most occupied one. For the minimum feasible frequency strategy, since the requests and frequencies are considered as pairs in [55], a pair of frequencies with the minimum highest value is selected (the highest value of a pair of frequencies is the larger one in the pair). In case of a tie, a pair with the minimum smallest value is chosen. These two strategies aim to maximize the number of unused frequencies. This allows more choices of frequencies for requests that will appear at later time periods.

The repair phase is executed in case some requests in the online problem could not be feasibly assigned. The objective of this phase is to find a feasible solution with a minimum number of re-assigned requests. Changing frequencies which have been assigned previously is technically allowed. However, in practice this can be time consuming and takes up human resources. Therefore, the problem considered here states that changing frequencies of requests should be avoided unless no other means of finding a feasible solution exists. Two types of algorithms were applied and compared to solve the repair problem: the first type is based on CN-tabu only, while the second

type is based on branch and bound followed by CN-tabu, which is used only when branch and bound could not solve the repair problem within a given time period.

Branch and bound is an exact algorithm which is used to solve the repair problem. It optimally assigns all the requests connected to the request currently being considered. During the search, the solution quality was evaluated according to the number of requests which have not been assigned to their original frequencies (called the number of repairs) and back-tracking is applied when a better solution cannot be produced. A disadvantage of this algorithm is that it can require a long run time. Therefore, the branch and bound is terminated after a fixed time period. Hence, there is no guarantee that a feasible solution will be found, and therefore CN-tabu is used to allow other changes to be considered.

If branch and bound fails to solve the repair problem, CN-tabu is applied. First, CN-tabu tries assigning each unassigned request to each available frequency and counts the number of violations. Then, the assignment that leads to the smallest number of violations is selected. After that, all clashing requests become unassigned.

In order to prevent CN-tabu from cycling, requests that just became unassigned and their frequencies are added to the tabu list to prevent these frequencies from being reassigned back to these requests for a certain number of iterations using a dynamic tabu tenure. Say a request r_i and a frequency f_j are added to the tabu list, then their dynamic tabu tenure is given by Formula 2.1.

$$\textit{Dynamic tabu tenure } (r_i, f_j) = \textit{iter} + freq(r_i, f_j) \tag{2.1}$$

where *iter* is the current number of iterations, and $freq(r_i, f_j)$ is the number of times f_j has been assigned to r_i. To clarify CN-tabu, consider Example 2.2.

Example 2.2:

Consider a dynamic FAP instance that consists of 6 requests and 3 frequencies ($f_1 = 10, f_2 = 15, f_3 = 20$). The constraints are given in Table 2.2. Note that the frequency that is assigned to request r_i is denoted as f_{r_i} and a dash "-" means that a feasible assignment could not be found.

Constraints
$\lvert f_{r_1} - f_{r_2} \rvert \geq 10$
$\lvert f_{r_1} - f_{r_3} \rvert \geq 10$
$\lvert f_{r_2} - f_{r_5} \rvert \geq 10$
$\lvert f_{r_3} - f_{r_5} \rvert \geq 10$
$\lvert f_{r_5} - f_{r_6} \rvert \geq 10$

Table 2.2: The constraints considered in Example 2.2.

An initial solution is given in Table 2.3, where there are 2 unassigned requests.

Request	r_1	r_2	r_3	r_4	r_5	r_6
Assigned frequency	15	-	-	10	10	20

Table 2.3: An initial solution in Example 2.2.

The number of violations after assigning each frequency to each unassigned request is given in Table 2.4.

requests	frequencies		
	f_1	f_2	f_3
r_2	2	2	1
r_3	2	2	1

Table 2.4: The number of violations after assigning the unassigned requests.

The assignments with the smallest number of violations are $f_{r_2} = 20$ and $f_{r_3} = 20$. One of them is randomly selected, say $f_{r_2} = 20$, and the clashing request r_1 (due to $\lvert f_{r_1} - f_{r_2} \rvert \geq 10$) is unassigned as given in Table 2.5.

Request	r_1	r_2	r_3	r_4	r_5	r_6
Assigned frequency	-	20	-	10	10	20

Table 2.5: The solution after assigning r_2.

After that, r_1 and f_2 are added to the tabu list with *freq*(r_1, f_2) = 1 and *iter* = 1, so the dynamic tabu tenure is updated using Formula 2.1. Therefore, this assignment is tabu for the next two iterations. After that, the number of violations after assigning each frequency to each unassigned request is updated as shown in Table 2.6.

Request	Frequency		
	f_1	f_2	f_3
r_1	0	1	1
r_3	1	1	0

Table 2.6: The number of violations after assigning the unassigned requests.

The assignments with the smallest number of violations are $f_{r_1} = 10$ and $f_{r_3} = 20$. One of these is chosen randomly, say $f_{r_1} = 10$. The new solution is given in Table 2.7.

Request	r_1	r_2	r_3	r_4	r_5	r_6
Assigned frequency	10	20	-	10	10	20

Table 2.7: The solution after assigning r_1.

Nothing is made tabu as no requests are unassigned. Then, the number of violations after assigning each frequency to each unassigned request is updated as given in Table 2.8.

Request	Frequency		
	f_1	f_2	f_3
r_3	2	2	0

Table 2.8: The number of violations after assigning the unassigned requests.

The best assignment is $f_{r_3} = 20$, which leads to a feasible solution as given in Table 2.9.

Request	r_1	r_2	r_3	r_4	r_5	r_6
Assigned frequency	10	20	20	10	10	20

Table 2.9: A feasible solution in Example 2.2.

Based on the experiments in [55], it was found that the approach for the dynamic FAP performed better when the minimum feasible frequency strategy is applied in the online assignment phase than the most occupied strategy. Moreover, it was found in [55] that using the second type of repair phase (using branch and bound and CN-tabu) performed better than using the first one (using CN-tabu only).

To sum up, it is found that the dynamic FAP has been studied in relatively few studies compared with the static FAP. Therefore, the dynamic FAP will be studied and solved in this thesis (see Chapter 6) using novel and existing techniques and compared with existing approaches in the literature.

2.8 Conclusions

This chapter provided background of the problems and methodologies that are considered and investigated in this thesis. It concerns two main problems, namely the static and the dynamic FAPs. In terms of the number of publications of these problems in the literature, the static FAP has been studied more than the dynamic FAP, for which only few studies exist. Therefore, this warrants further study into the dynamic FAP, which appears to be an interesting and practical problem. Moreover, the relationship between the static FAP and the GCP has been discussed and reviewed in the literature. This relationship shows that the static FAP is a generalization of the GCP, which means many algorithms and techniques which have been used to solve the GCP can also be used to solve the static FAP.

The literature showed several motivations to select the heuristic algorithms considered in this study, namely TS, ACO and HH. One of these motivations is that these heuristic algorithms are very flexible and have been applied successfully to a variety of problems. These heuristic algorithms have been reviewed in this chapter by presenting how they were used to solve the static FAP and other problems. For the static FAP, it was found that TS is amongst the most popular solution method, whereas ACO is not very popular compared with the other two algorithms. However, all the three heuristic algorithms have proven track records of producing good quality solutions across a range of problems. To the best of my knowledge, there are only two published papers that implemented TS and one that implemented ACO to solve the static FAP on the datasets considered in this thesis, which are CELAR and GRAPH. On the other hand, no published papers apply HH to solve the static FAP on the datasets considered in this thesis. Finally, the dynamic FAP and existing approaches for this problem were reviewed.

Chapter 3

Tabu Search for the Static FAP

3.1 Introduction

Tabu search (TS) is a modern meta-heuristic algorithm designed to solve difficult combinatorial optimization problems. This algorithm is an extension to local search that allows the search to escape from local optima. The main concepts of TS are accepting non-improving moves (i.e. changes made to a solution) and using a flexible memory called a tabu list to restrict the next choice of neighbour, thereby preventing the search from revisiting previously visited solutions. In order to implement TS for a particular problem, several decisions must be made such as how to define a solution space, a neighbourhood, a cost function, a tabu list and aspiration criteria.

This algorithm has proved to be an efficient method to find a high quality solution for a variety of optimization problems (see e.g. [75]). However, existing TS algorithms in the literature are unable to find the optimal solutions in some instances of the static frequency assignment problem (FAP) considered in this study.

In this chapter, an improved TS algorithm is applied to solve the static FAP. This algorithm is mainly designed to solve the minimum order FAP (MO-FAP) using several novel and existing techniques. One of the novel techniques is hybridising TS with multiple neighbourhood structures, one of which is used as a diversification technique. In contrast, existing TS algorithms for the static FAP in the literature implemented only a single neighbourhood structure (see e.g. [15, 16, 86, 145]). Another novel technique is determining a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist. These lower bounds are based on the underlying graph colouring model (see Section 3.2) and ensure that the search focuses on parts of the solution space that are likely to contain feasible solutions. Additionally, this chapter investigates whether TS without significant changes can prove effective on other variants of the static FAP, namely the minimum span FAP (MS-FAP) and the minimum interference FAP (MI-FAP). This chapter focuses on the following research questions:

- *Is TS an effective solution method for the static FAP?*
- *Is it beneficial to hybridise TS with multiple neighbourhood structures?*
- *Can TS without significant changes be effective on different variants of the static FAP?*

This chapter is organised as follows: the next section explains how the underlying graph colouring model for the static FAP can be used to provide a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist and how this information can be used to assist the search. In Section 3.3, an overview of the TS algorithm for solving the static FAP is given and Section 3.4 presents the main components of the TS algorithm. Results of this algorithm are given and discussed in Section 3.5 before this chapter finishes with conclusions.

3.2 Graph Colouring Model for the Static FAP

The graph colouring problem (GCP) for a graph $G(V,E)$ consisting of $|V|$ vertices and $|E|$ edges involves allocating a colour to each vertex such that no adjacent vertices are in the same colour class and the number of colours is minimized. The GCP can be seen as an underlying model to the static FAP [83]. In other words, a static FAP instance can be represented as a GCP by representing each request as a vertex, each fre-

quency as a colour and each bidirectional or interference constraint as an edge joining the corresponding vertices.

One useful concept of graph theory is the idea of cliques. A clique in a graph can be defined as a set of vertices in which each vertex is linked to all other vertices. A maximum clique is the largest among all cliques in the graph. Every vertex in a clique has to be allocated to a different colour in a feasible colouring. Therefore, the size (i.e. the number of vertices) of the maximum clique acts as a lower bound on the minimum number of colours in a GCP instance and, by extension, as a lower bound on the number of frequencies for a static FAP instance.

A graph colouring representation of a static FAP instance may contain many cliques with different clique numbers. For instance, one of the cliques in the CELAR 01 instance includes the requests r_1, r_{200}, r_{871}, r_{872} and r_{899}, which are linked to each other by either bidirectional or interference constraints as shown in Figure 3.1.

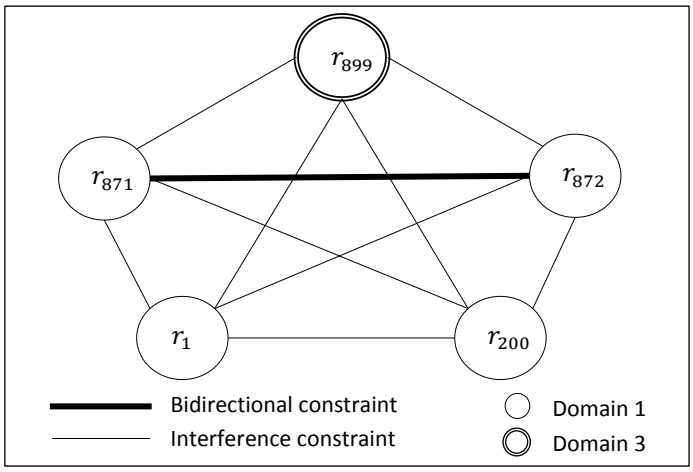

Figure 3.1: An example of a clique in the CELAR 01 instance in the graph colouring model.

Figure 3.1 shows 5 different requests that are linked to each other, hence 5 different frequencies are required to feasibly assign these requests. Additionally, the requests r_1, r_{200}, r_{871} and r_{872} belong to domain 1, while request r_{899} belongs to domain 3. As these requests belong to different domains, the graph colouring model for each domain can be considered separately to calculate a lower bound on the number of frequencies that is required from each domain. For example, the clique shown in Figure 3.1 indicates that the minimum number of frequencies for domain 1 is at least 4. Also, at least 1 frequency is required from domain 3, because request r_{899} belongs to domain 3. In total, at least 5 different frequencies are required.

3.2.1 Computational Time of Lower Bounds

This section considers the effect of the numbers of requests and constraints on the run time to calculate a lower bound of a static FAP instance based on the maximum clique. To assist the analysis, the density of a static FAP is defined by Formula 3.1.

$$Density\ of\ a\ static\ FAP = \frac{Number\ of\ constraints}{Maximum\ possible\ number\ of\ constraints} \quad (3.1)$$

where the maximum possible number of constraints can be calculated by Formula 3.2.

$$Maximum\ possible\ number\ of\ constraints = \frac{NR\ (NR-1)}{2} \quad (3.2)$$

where *NR* is the number of requests.

Experiments were conducted using randomly produced instances with the number of requests varied between 200 and 1,000; and the density varied between 0.05 and 0.4 (i.e. 5% and 40%). A branch and bound algorithm in [17] is used to calculate the maximum clique size for each instance. The run times of this algorithm are shown in Table 3.1.

No. of requests	Density				
	0.05	0.10	0.20	0.30	0.40
200	0.05 sec	0.09 sec	0.34 sec	1.50 sec	8.02 sec
400	0.23 sec	0.76 sec	5.76 sec	46.57 sec	431.67 sec
600	0.84 sec	3.35 sec	37.03 sec	390.38 sec	5,490.33 sec
800	2.22 sec	10.12 sec	141.04 sec	1,880.53 sec	36,250.80 sec
1,000	4.77 sec	25.05 sec	395.18 sec	6,748.14 sec	163,100.00 sec

Table 3.1: Run times for finding the maximum clique size for different numbers of requests and values of density.

Table 3.1 shows that increasing the problem density has a dramatic effect on run time. It can be seen that it required over 45 hours to find the maximum clique size of a problem with 1,000 requests and a density of 0.40.

Figure 3.2 presents the relationship between the log of the run time versus the number of requests.

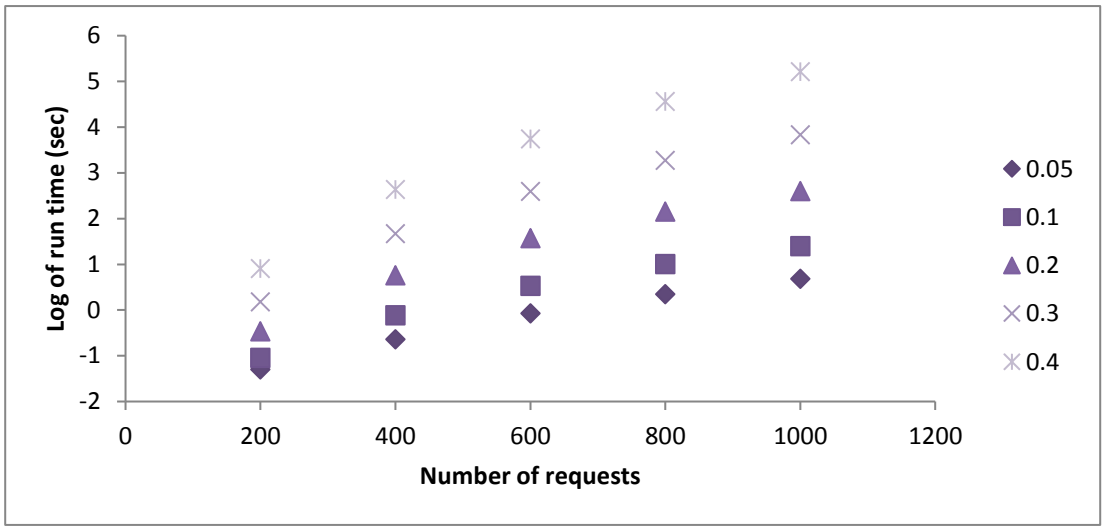

Figure 3.2: The relationship between the log of run time versus the number of requests.

Figure 3.2 demonstrates an exponential increase in run time as the number of requests increases. Table 3.2 shows the densities of the datasets considered in this thesis.

Instance	Variant of the static FAP	Number of Requests	Number of constraints	Max. number	Density
CELAR 01	MO-FAP	916	6,464	419,070	0.015425
CELAR 02	MO-FAP	200	1,435	19,900	0.072111
CELAR 03	MO-FAP	400	3,160	79,800	0.039599
CELAR 04	MO-FAP	680	4,647	230,860	0.020129
CELAR 11	MO-FAP	680	4,783	230,860	0.020718
GRAPH 01	MO-FAP	200	1,334	19,900	0.067035
GRAPH 02	MO-FAP	400	2,645	79,800	0.033145
GRAPH 08	MO-FAP	680	4,437	230,860	0.019219
GRAPH 09	MO-FAP	916	6,162	419,070	0.014704
GRAPH 14	MO-FAP	916	5,554	419,070	0.013253
CELAR 05	MS-FAP	400	2,998	79,800	0.037569
GRAPH 03	MS-FAP	200	1,334	19,900	0.067035
GRAPH 04	MS-FAP	400	2,644	79,800	0.033133
GRAPH 10	MS-FAP	680	4,587	230,860	0.019869
CELAR 06	MI-FAP	200	1,522	19,900	0.076482
CELAR 07	MI-FAP	400	3,265	79,800	0.040915
CELAR 08	MI-FAP	916	6,660	419,070	0.015892
CELAR 09	MI-FAP	680	4,437	230,860	0.019219
CELAR 10	MI-FAP	680	4,437	230,860	0.019219
GRAPH 05	MI-FAP	200	1,334	19,900	0.067035
GRAPH 06	MI-FAP	400	2,570	79,800	0.032206
GRAPH 07	MI-FAP	400	2,570	79,800	0.032206
GRAPH 11	MI-FAP	680	4,437	230,860	0.019219
GRAPH 12	MI-FAP	680	4,697	230,860	0.020346
GRAPH 13	MI-FAP	916	6,189	419,070	0.014768

Table 3.2: The maximum possible number of constraints and the density for the considered datasets.

Table 3.2 shows that the densities of the datasets considered in this thesis varied between 0.013 and 0.076. For densities within this range, the maximum cliques can be determined quickly, generally taking less than 5 seconds for a problem with 1,000 requests. For the considered datasets the larger datasets in terms of the number of re-

quests tended to have a smaller density and vice versa. In practice, when a problem is given, the density can be calculated and then a decision made as to whether it is worthwhile calculating the clique sizes.

3.2.2 Lower Bounds for the Static FAP

A branch and bound algorithm in [17] is used to obtain the set of all maximum cliques for each domain within each instance. This concept can be used for all types of the static FAP except for the MI-FAP because the objective of this problem is different and allows all the frequencies to be used. Table 3.3 gives a lower bound on the numbers of frequencies that are required from each domain for a feasible solution to exist and a lower bound for the whole instance, and the time taken to calculate these lower bounds.

Instance	Variant of the static FAP	Domain							Whole instance	Run time
		1	2	3	4	5	6	7		
CELAR 01	MO-FAP	10	9	10	4	4	7	2	12	1.50 sec
CELAR 02	MO-FAP	10	0	10	0	0	0	2	14	0.02 sec
CELAR 03	MO-FAP	10	0	10	0	2	0	2	12	0.06 sec
CELAR 04	MO-FAP	10	0	10	4	2	0	2	44	0.34 sec
CELAR 11	MO-FAP	20	0	14	4	2	0	2	20	0.34 sec
GRAPH 01	MO-FAP	8	3	6	2	4	4	2	18	0.03 sec
GRAPH 02	MO-FAP	6	2	4	0	2	4	0	14	0.12 sec
GRAPH 08	MO-FAP	10	2	6	2	3	8	3	16	0.28 sec
GRAPH 09	MO-FAP	6	2	10	2	2	8	2	18	0.48 sec
GRAPH 14	MO-FAP	6	2	4	2	0	2	2	8	0.48 sec
CELAR 05	MS-FAP	10	0	10	0	2	0	2	12	0.08 sec
GRAPH 03	MS-FAP	8	0	6	3	2	6	2	12	0.06 sec
GRAPH 04	MS-FAP	6	2	6	2	0	8	3	14	0.12 sec
GRAPH 10	MS-FAP	8	3	6	2	0	10	2	10	0.08 sec

Table 3.3: Lower bounds of the numbers of frequencies required for each domain and for the whole instance, and the time taken to calculate them.

The results in Table 3.3 were obtained using FORTRAN 95 and all experiments were conducted on a 3.0 GHz Intel Core I3-2120 Processor (2nd Generation) with 8GB RAM and a 1TB Hard Drive.

Table 3.3 shows that the run time for all the instances except one is below one second. Note that some frequencies are part of more than one domain, meaning the sum of the lower bound for each domain does not necessarily equal the lower bound for the whole instance. Additionally, CELAR 04 is the only instance here which has pre-assignment constraints, which is used to strengthen the lower bound.

The lower bounds are applied in this study and can be used in two ways: firstly, the search stops as soon as it finds a feasible solution such that the total number of used frequencies is equal to the lower bound of the whole instance (as this is the optimal solution); secondly, the lower bound for each domain are used to ensure that an algorithm never wastes time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bound for each domain since there can be no feasible solutions in this search area.

3.3 Overview of the Tabu Search Algorithm

A key decision when constructing a TS algorithm is how to define the solution space and the corresponding cost function.

3.3.1 Solution Space and Cost Function

It is relatively straightforward to find solutions that satisfy bidirectional, domain and pre-assignment constraints, and to define a neighbourhood operator that moves between such solutions [47]. Here, two configurations are considered. In the first configuration, bidirectional, domain and pre-assignment constraints are enforced while interference constraints are relaxed. Note that interference constraints are relaxed because these are the most difficult constraints to be satisfied [47]. This configuration was previously used in TS for the static FAP [86, 145]. In the second configuration, only domain and pre-assignment constraints are enforced, while bidirectional and interference constraints are relaxed. This configuration was previously used in TS for the static FAP [15, 85]. In both configurations, the cost function is defined as the number of violations.

There are advantages and disadvantages in using each configuration. One of the advantages of using the first configuration is that, in effect, the number of requests is halved because requests are considered as pairs based on the bidirectional constraints. However, this leads to a restriction in the search space which may result in difficulties in the search. In contrast, using the second configuration gives more freedom. Therefore, these two types of configuration are implemented and compared.

The solution space could have been defined as the set of all possible feasible assignments, that is, satisfying all of the constraints, and the corresponding cost function as

the number of used frequencies. However, there are a number of difficulties with this configuration: first, TS has been found to be poor with this configuration [15]. Moreover, it may be difficult to move from one feasible solution to another. Furthermore, a large number of neighbour solutions with the same cost may differ greatly in their quality [49] (clarified by Example 3.1). Hence, this configuration is not considered.

Example 3.1:

Assume a static FAP instance has 50 requests and 2 feasible solutions use 5 frequencies. Table 3.4 shows the number of requests assigned to each used frequency and the corresponding cost function value for each solution.

	f_1	f_2	f_3	f_4	f_5	Total no. requests	Cost function value
Solution 1	10	10	10	10	10	50	5
Solution 2	14	12	12	11	1	50	5

Table 3.4: The number of requests assigned to each used frequency in two different feasible solutions.

Table 3.4 shows that solution 2 is closer to a solution using just 4 frequencies. However, both of the solutions have the same cost function value. This means that using this definition of the cost function may not guide the search towards solutions using fewer frequencies.

3.3.2 Sub-problem in the static FAP

Using the solution space which relaxes some constraints creates the following sub-problem: minimizing the number of violations with a fixed number of used frequencies. If a solution with zero violations (a feasible solution) is found in the improvement phase (see Section 3.4.6), then the number of used frequencies is reduced in the creating violations phase (see Section 3.4.5) and the sub-problem is reconsidered. The process is repeated until a feasible solution can no longer be found. This process is similar to TS for the GCP in [87] and TS for the static FAP in [86, 145].

3.3.3 Structure of the Tabu Search Algorithm

The TS algorithm consists of three phases, namely the initial solution phase, the creating violations phase and the improvement phase. The initial solution phase (see Section 3.4.4) generates an initial solution. Assume the initial solution is feasible. Then,

the creating violations phase (see Section 3.4.5) reduces the number of used frequencies by removing a used frequency. Then, all requests that are assigned to the removed frequency are re-assigned to another used frequency, which may result in some violations. The improvement phase (see Section 3.4.6) aims to reduce the number of violations to zero, using three neighbourhood structures. Three independent tabu lists, one for each neighbourhood structure, are defined in this algorithm. Notice that all of the tabu lists are cleared after the sub-problem is solved. If the improvement phase results in a feasible solution within a specified number of iterations, then the creating violations phase is revisited to remove another used frequency. After that, the process continues until either no feasible solution can be found or the number of used frequencies equal to the lower bound. In case the initial solution is not feasible, the creating violations phase can be omitted and the search moves immediately to the improvement phase. Figure 3.3 illustrates the overall structure of the TS algorithm for the static FAP in this study.

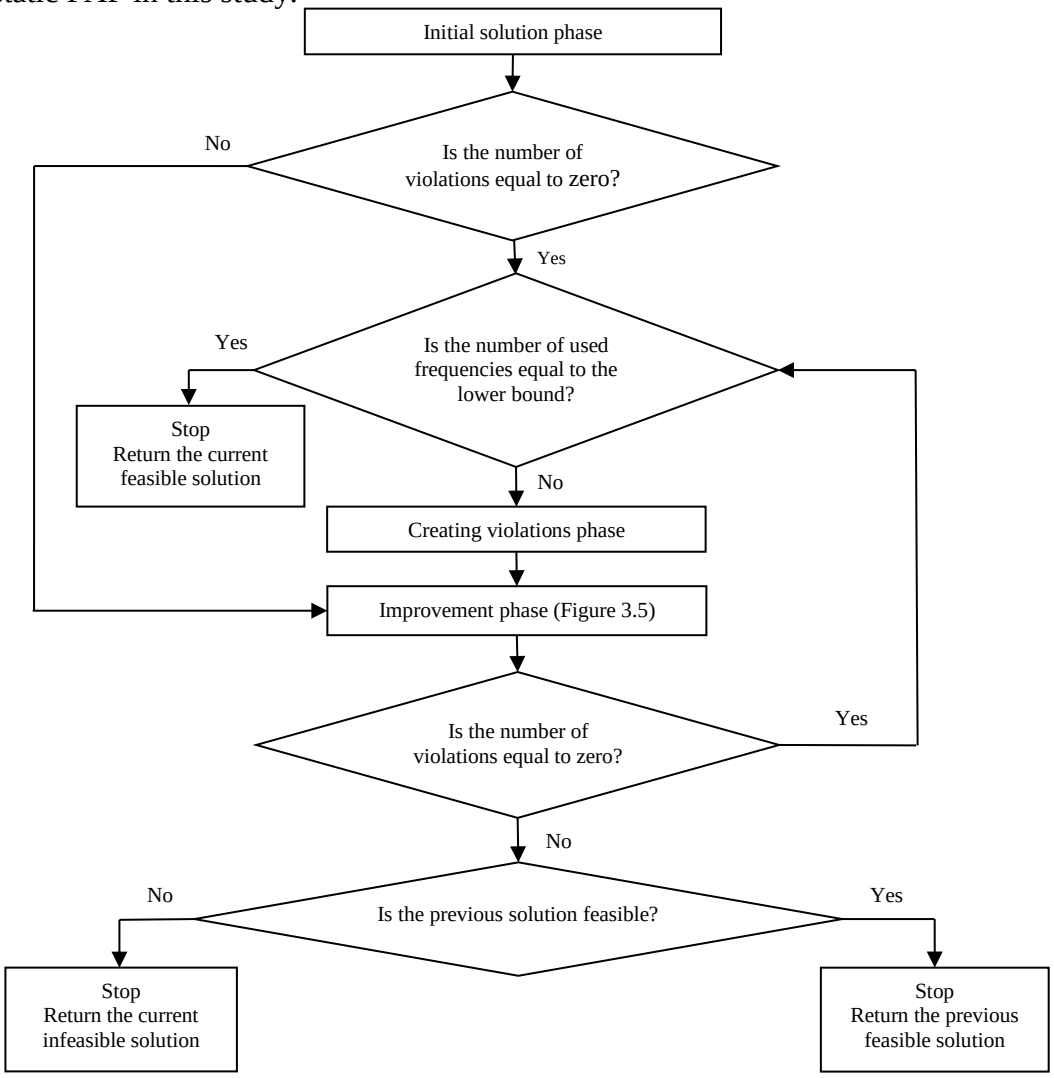

Figure 3.3: Overall structure of the TS algorithm for the static FAP.

3.4 Components of the Tabu Search Algorithm

The main components and the implementation of the TS algorithm are presented and discussed in this section.

3.4.1 Neighbourhood Structures

Hybridising TS with multiple neighbourhood structures is one of the techniques which aim to improve the performance of this algorithm and make it different from existing TS. In fact, existing TS algorithms for the static FAP implemented only a single neighbourhood structure (see e.g. [15, 16, 86, 145]). The concept of using multiple neighbourhood structures is inherited from variable neighbourhood search, which was introduced in [112] and has been proved to be an effective solution method to solve the static FAP in the literature (see e.g. [110]).

In this study, TS is hybridised with three different neighbourhoods, namely a move neighbourhood structure (MNS), a swap neighbourhood structure (SNS) and a diversification neighbourhood structure (DNS). MNS was previously used in TS for the static FAP [15, 16, 86, 145], whereas SNS and DNS are new techniques. The three neighbourhood structures are defined as follows:

i) Move neighbourhood structure: this structure is defined as the set of solutions obtained by selecting a request to be re-assigned to a different used frequency. Therefore, this neighbourhood investigates all the possible moves for all requests and used frequencies (the maximum possible number of such moves is $NR \times n_f$, where n_f is the number of used frequencies). This ensures that the number of used frequencies does not increase.

ii) Swap neighbourhood structure: this structure is defined as the set of solutions obtained by swapping the frequencies of each request with its partner (based on the bidirectional constraints). SNS proves to be quick as it contains a small number of neighbours (at most $NR/2$), yet it can improve the solution quality.

iii) Diversification neighbourhood structure: this structure, unlike the previous structures, is intended to diversify the search, i.e. move to a different part of the solution space. It consists of the set of solutions obtained by replacing a used (old) frequency with an unused (new) frequency. Given an old frequency, another frequency is ac-

cepted if it can be assigned to all requests which were assigned to the old frequency. However, any re-assignment that causes the number of used frequencies to drop below the lower bound for some domains (see Section 3.2) is not considered.

3.4.2 Tabu Lists

Three independent tabu lists, one for each neighbourhood structure, are defined in this algorithm. Notice that all of the tabu lists are cleared after the sub-problem is solved. These tabu lists are described as follows:

i) Move tabu list: when a request is re-assigned to another frequency, then the request and the removed frequency are added to the tabu list and this assignment is classified as forbidden for a given number of iterations (i.e. tabu tenure).

ii) Swap tabu list: when a request is swapped with its partner as a pair (based on the bidirectional constants), then this pair is added to the swap tabu list. This list prevents a pair of requests from being swapped more than once.

iii) Diversification tabu list: when an old frequency is replaced by a new frequency, then both of them are added to the diversification tabu list.

3.4.3 Aspiration Criteria

Sometimes the tabu list is too restrictive by forbidding some attractive moves even when there is no harm of cycling. Hence, it is essential to use a technique to escape from this situation by ignoring the tabu list. This is called the aspiration criteria. Here, the logical and commonly used aspiration criteria is applied, that is, to accept a tabu move if it leads to a better solution than the current best one.

3.4.4 The Initial Solution Phase

It is sensible to produce a good initial solution in order to improve the efficiency of this algorithm [155]. Here, the objective is to produce a feasible initial solution with as few frequencies as possible, although for some problem instances this may be difficult and the initial solution may be infeasible.

This phase consists of three stages, namely the assignment stage, the allowing infeasible solutions stage and the descent method stage. All these stages are described in the following subsections.

3.4.4.1 The Assignment Stage

This stage aims to assign each request a frequency, where the selection of requests and frequencies is based on three arrays as follows:

- *Request Array* (R_A): the elements of this array correspond to the number of feasible frequencies for each request. So, at the start, these elements correspond to the size of the domain for each request. This array is updated each time a frequency is assigned a request.
- *Frequency Array* (F_A): the elements of this array correspond to the number of requests that can be feasibly assigned to each frequency. This array also is updated each time a frequency is assigned a request.
- *Constraint Array* (C_A): the elements of this array correspond to the number of relaxed constraints which are involved for each request. Notice that this array is constant because this reflects the number of relaxed constraints in the static FAP instance, which is fixed.

Selection of Requests and Frequencies: a request which has the minimum value of R_A is chosen. There are two possible cases:

Case A: Min $(R_A) \neq 0$

In case there is more than one request with the same minimum value of R_A, then the one with the maximum value of C_A is chosen. If there is still more than one such request, then one of them is selected at random. After choosing a request, a frequency is selected to be assigned. Therefore, we randomly choose one of the used frequencies that can be feasibly assigned to the selected request. If there is no such frequency, then an unused frequency is selected. In order to choose an unused frequency, we choose the one which can be feasibly assigned to the selected request and has the maximum value of F_A. In case of a tie, one of them is selected randomly. This type of selection is intended to minimize the number of used frequencies. After that, the next request is then considered.

Case B: Min $(R_A) = 0$

If there is more than one request that has no feasible frequencies, then one of them is randomly selected and is called the candidate request. Notice that the candidate request cannot be feasibly assigned. However, experiments showed that in many cases, this infeasibility could be fixed by a simple re-assignment phase. In this phase, the number of attempts to assign the candidate request is counted.

Re-assignment phase: if the number of attempts to feasibly assign the candidate request is not greater than 500 (where this number is chosen based on experiments), then two groups are generated as follows: the candidate request is assigned to all available frequencies in turn. For each assignment, requests that are involved in violated constraints are added to the first group, while the other requests are added to the second group. After that, a request is selected from one of these groups in order to be re-assigned as follows: if the number of attempts to feasibly assign the candidate request is less than 250 (where this number is chosen based on experiments), then a request is randomly selected from the first group; otherwise it is selected from the second group. Then, the selected request is feasibly re-assigned, if possible, in order to allow the candidate request to be feasibly assigned. The process is repeated until a feasible initial solution is found or the number of attempts to feasibly assign the candidate request exceeds 500, then the allowing infeasible assignments stage is executed.

3.4.4.2 The Allowing Infeasible Assignments Stage

This stage allows the candidate request to be assigned to an infeasible frequency which causes the lowest number of violations. If there is still more than one such frequency, then one of them is selected randomly. After assigning all the requests, the descent method stage is executed to attempt to find a feasible initial solution.

3.4.4.3 The Descent Method Stage

The descent method is used to attempt to reduce the number of violations to zero. This method uses MNS and the cost function is defined as the number of violations. Initially, only used frequencies are considered. Steepest descent is used and if there are several moves that lead to equal improvements, one of them is selected at random. Once a local optimum is found, if the number of violations is still greater than zero, any

unused frequencies are considered. Moves that cause the cost to remain unchanged are now accepted and the search terminates when a feasible solution is found or when 10,000 iterations have been completed (this value was set experimentally). Figure 3.4 shows the overall structure of the initial solution phase. The descriptions of the abbreviations in Figure 3.4 are given in Table 3.5.

Abbreviation	Definition
T_{r_k}	The number of attempts to assign the candidate request r_k.
RC	The set of the requests that clash with the candidate request with respect to some constraints when all possible frequencies are assigned to the candidate request.
NRC	The number of requests in *RC*
RS	The set of the requests that do not clash with the candidate request when all possible frequencies are assigned to the candidate request.
NRS	The number of requests in *RS*.

Table 3.5: The definition of the abbreviations in Figure 3.4

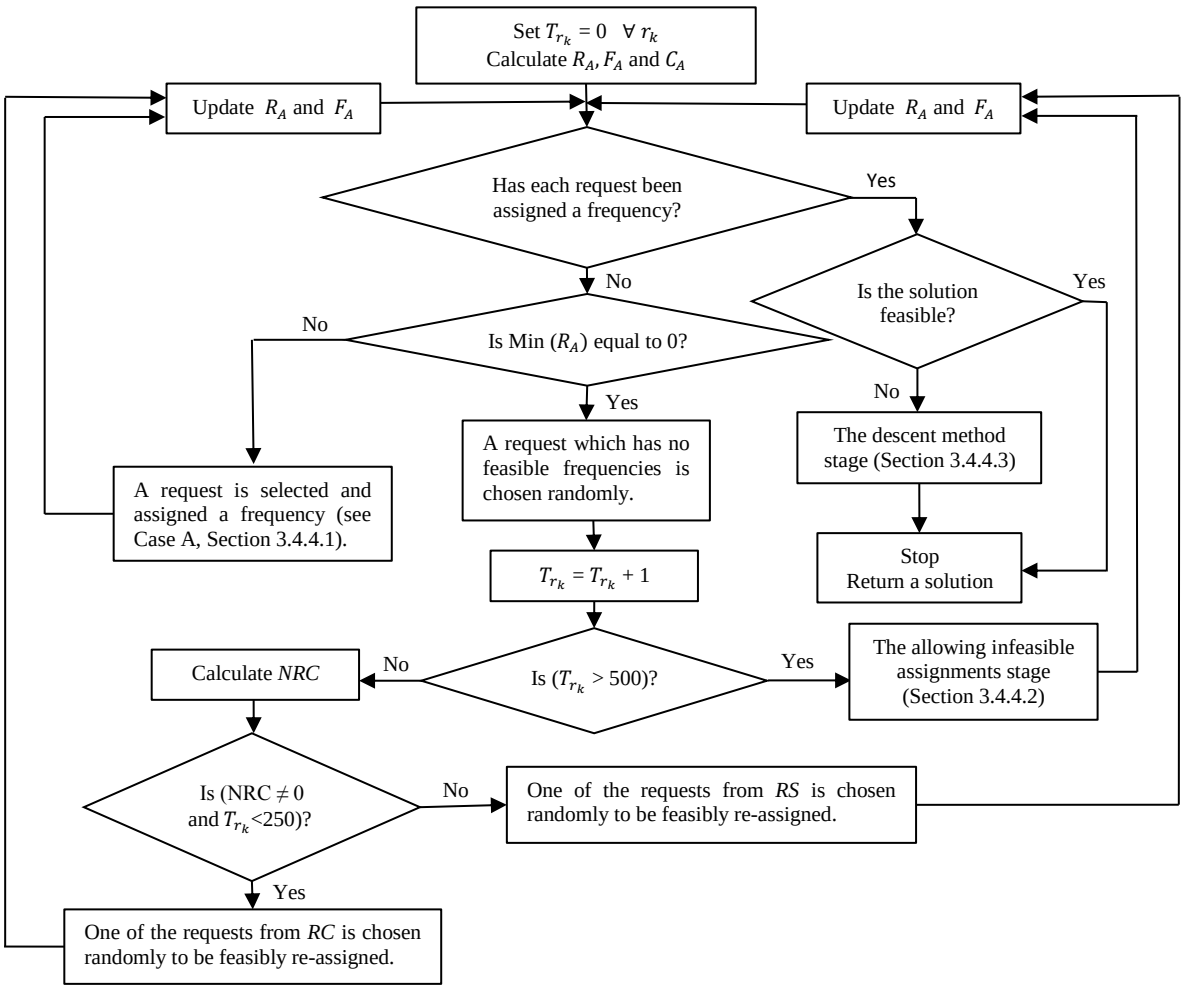

Figure 3.4: Overall structure of the initial solution phase.

3.4.5 The Creating Violations Phase

This phase aims to reduce the number of used frequencies in a feasible solution by removing a used frequency. The frequency that will be removed must satisfy the following conditions: (i) it is not involved in any pre-assignment constraints; (ii) the lower bound on the number of frequencies that are required from each domain based on the underlying graph colouring model (see Section 3.2) is satisfied after removing this frequency. If there is more than one candidate frequency, then the one which is assigned to the least number of requests is selected. If there is still more than one, then one of them is randomly selected. After that, the requests which are assigned to the candidate frequency are re-assigned to another feasible used frequency. The process is repeated until there is no feasible used frequency. In this case, these requests are randomly re-assigned to infeasible used frequencies, and then the improvement phase (see Section 3.4.6) is executed to find a feasible solution. The creating violations phase was previously applied in TS for the static FAP in [86].

3.4.6 The Improvement Phase

This phase starts from an infeasible solution which is usually produced by the creating violations phase. Then, the iterative procedure of TS starts in the improvement phase. The aim of this phase is to solve the sub-problem using three neighbourhood structures, namely the move neighbourhood structure (MNS), the swap neighbourhood structure (SNS), and the diversification neighbourhood structure (DNS). In MNS and SNS, only used frequencies are considered, while DNS considers only unused frequencies. MNS is explored first because it contains a large number of neighbours. SNS, which covers a limited number of neighbours, is then considered to support the MNS. DNS aims to jump from the current position in the solution space to a new position by removing a used frequency and adding a new one from the set of unused frequencies. Therefore, DNS is intended to diversify the search rather than reduce the number of violations, which reflects the reason for leaving it as the last structure.

Implementation of the improvement phase

One of the three neighbourhood structures is executed each iteration, where this phase begins with MNS. If this structure results in a better solution, then it is accepted. Otherwise, it is repeated until MNS is executed for a given number of times consecutively

without improvement. Then, the search enters SNS. If this structure leads to a better or equally good solution, then the search goes back to MNS. Otherwise, it appears there is little prospect of finding a better solution in the current region of the solution space, so the search enters DNS. A solution from DNS is accepted and the search returns to MNS.

It was found that on occasions, no moves in DNS are allowed due to the tabu lists, the pre-assignment constraints and the lower bound for each domain. If this happens, the criterion of selecting a new frequency in DNS is modified, that is, a frequency is accepted as a new frequency if it can be assigned to at least one request (instead of all requests) that were assigned to the old frequency. Although the new frequency is not allowed to be removed because of the diversification tabu list, the old frequency is allowed to return to the solution because of the limited number of neighbours in this structure. So, there are two possible types of diversification neighbourhood structure: structure A (see Section 3.4.1) and structure B (described here).

The output of the improvement phase can be a feasible or an infeasible solution. If it is a feasible, but not optimal solution, then the algorithm returns to the creating violations phase. In contrast, if the output is an infeasible solution, then the algorithm returns to MNS. This continues until one of the stopping criteria is met. Figure 3.5 illustrates the overall structure of the improvement phase.

3.4.7 Stopping Criteria

The TS algorithm has three different stopping criteria as follows: (i) it finds a feasible solution whose number of frequencies is equal to the lower bound (as this is an optimal solution), (ii) the number of iterations reaches a given number without successfully solving the sub-problem (see Section 3.3.2), i.e. a feasible solution could not be achieved (note that the number of iterations is reset to zero each time the sub-problem is solved), (iii) DNS is executed for a given number of times.

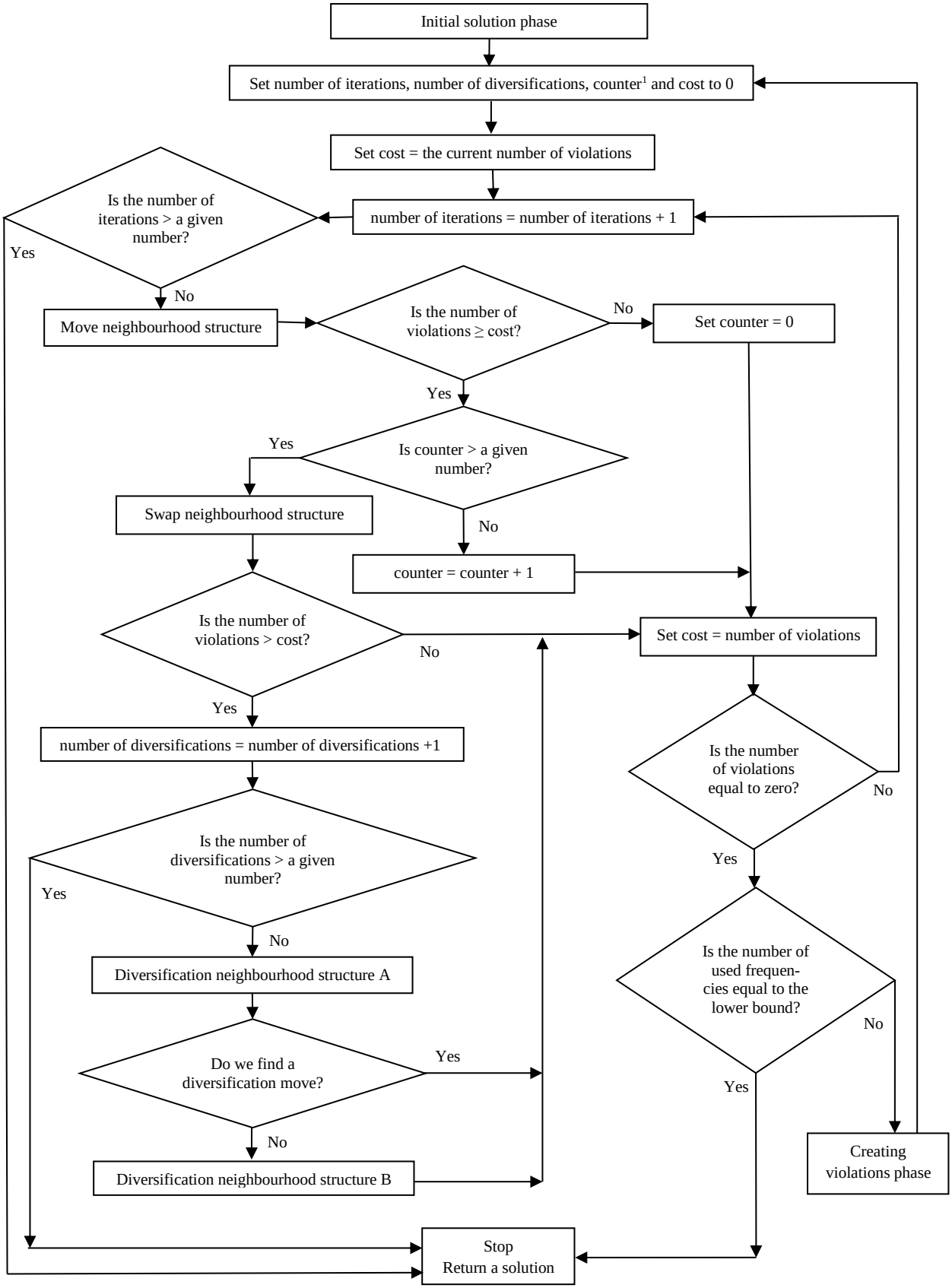

Figure 3.5: Overall structure of the improvement phase.

[1] The counter parameter counts the number of times worse solutions are accepted in the move neighborhood structure.

3.5 Experiments and Results

This section presents the performance of TS for the static FAP using CELAR and GRAPH datasets (available on the FAP website[1]). After that, the process of this algorithm is analysed. Finally, the performance of our TS algorithm is compared with existing algorithms in the literature. The parameters of our TS algorithm are set based on experimentations for solving the sub-problem as follows:

- The maximum number of iterations is 10,000.
- The maximum number of times of accepting worse solutions consecutively in MNS is 100.
- The maximum number of times of executing DNS is 20.
- The tabu tenure of the move tabu list is 100.
- The tabu tenure of the swap tabu list is *NR*/2, where *NR* is the number of requests in the instance.
- The tabu tenure of the diversification tabu list is 20.

In this study, the algorithm was coded using FORTRAN 95 and all experiments were conducted on a 3.0 GHz Intel Core I3-2120 Processor (2nd Generation) with 8GB RAM and a 1TB Hard Drive.

3.5.1 Results of the Tabu Search Algorithm

The results of TS are given for the three variants of the static FAP, namely MO-FAP, MS-FAP and MI-FAP, in two parts. The first part gives the results of the initial solution phase, while the second part compares the results of TS using two types of configurations (see Section 3.3.1). The optimal solutions of these datasets are known and available on the static FAP website[1]. Therefore, the solutions of TS are compared with known optimal solutions.

The results of two different numbers of runs (5 and 20) of TS are compared to investigate whether there is a significant difference. It is found that there is no significant difference using the Wilcoxon signed-rank test. Therefore, for each instance of the static FAP, 5 runs are performed, where each run uses a different random number stream.

[1] http://fap.zib.de/problems/CALMA/ (last accessed 25 February 2015).

Recall that the results of TS for the MO-FAP refer to the number of used frequencies in a feasible solution. Note that a bold number means that the optimal solution was achieved.

3.5.1.1 The Initial Solution Phase

The initial solutions of TS for the MO-FAP are given in Table 3.6.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	20	24	22.4	16	43.33 sec
CELAR 02	**14**	16	14.4	14	0.22 sec
CELAR 03	18	20	18.4	14	18.87 sec
CELAR 04	**46**	**46**	46.0	46	54.43 sec
CELAR 11	44	48	46.7	22	1.50 min
GRAPH 01	22	24	22.4	18	1.43 sec
GRAPH 02	16	20	18.8	14	1.24 sec
GRAPH 08	26	32	29.2	18	3.98 sec
GRAPH 09	28	44	33.6	18	52.50 sec
GRAPH 14	12	14	13.2	8	49.01 sec

Table 3.6: the initial solution of TS for the MO-FAP.

Table 3.6 shows that the optimal solution was obtained using TS for the MO-FAP in only two instances (CELAR 02 and CELAR 04). However, a feasible initial solution was achieved for almost all of the instances, although the algorithm failed to find a feasible solution for 2 out of 5 runs for CELAR 11. It is clear that for some instances, this algorithm used considerably more frequencies than the optimal number of used frequencies. This is expected as we used a relatively simple algorithm to produce the initial solution. It is encouraging that in almost all of the experiments, the initial solution phase managed to produce feasible solutions.

For the MS-FAP, no feasible initial solution was achieved for all of the instances. Additionally, initial solutions of the MI-FAP were poor although some changes were made in this algorithm to suit this problem. This is because the initial solution phase is designed to find a feasible solution, whereas for the MI-FAP there is no feasible solution.

3.5.1.2 Comparison of Different Configurations

The results of the two configurations of our TS algorithm (see Section 3.3.1) are compared. Table 3.7 shows the results of TS for the MO-FAP using the first configuration, which relaxes the interference constraints, while Table 3.8 presents the results of TS for the MO-FAP using the second configuration, where the bidirectional and inter-

ference constraints are relaxed. Both tables of results include the average results of the five runs, the optimal solution and the average run time. Note that the run time of finding the lower bound of the number of frequencies for each domain (see Table 3.1) is included.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	**16**	**16**	16.0	16	3.63 min
CELAR 02	**14**	**14**	14.0	14	0.52 sec
CELAR 03	**14**	16	14.8	14	1.00 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	38	40	38.4	22	8.81 min
GRAPH 01	**18**	**18**	18.0	18	5.43 sec
GRAPH 02	**14**	**14**	14.0	14	2.16 sec
GRAPH 08	**18**	**18**	18.0	18	24.28 sec
GRAPH 09	**18**	**18**	18.0	18	3.01 min
GRAPH 14	**8**	**8**	8.0	8	4.81 min

Table 3.7: Results of TS for the MO-FAP when the interference constraints are relaxed.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	22	18.8	16	5.83 min
CELAR 02	**14**	**14**	14.0	14	0.62 sec
CELAR 03	16	18	17.2	14	2.00 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	38	44	42.0	22	4.21 min
GRAPH 01	**18**	22	19.6	18	24.03 sec
GRAPH 02	**14**	16	14.4	14	42.12 sec
GRAPH 08	26	30	27.6	18	3.40 min
GRAPH 09	20	24	21.6	18	8.81 min
GRAPH 14	10	10	10.0	8	10.81 min

Table 3.8: Results of TS for the MO-FAP when the bidirectional and interference constraints are relaxed.

Table 3.7 shows that the optimal solution was achieved using the first configuration for all the instances except CELAR 11 and for two runs of CELAR 03. Moreover, these results were achieved in a reasonable time, mostly less than 5 minutes. In contrast, Table 3.8 shows that the optimal solution was achieved using the second configuration in almost all runs of GRAPH 02 and in two runs of GRAPH 01. Additionally, some initial solutions (see Table 3.6) were improved. However, for some instances, it used considerably more frequencies than the optimal solution. Moreover, using the second configuration consumed more time.

The average results of TS for the MO-FAP using the two types of configuration is given in Figure 3.6.

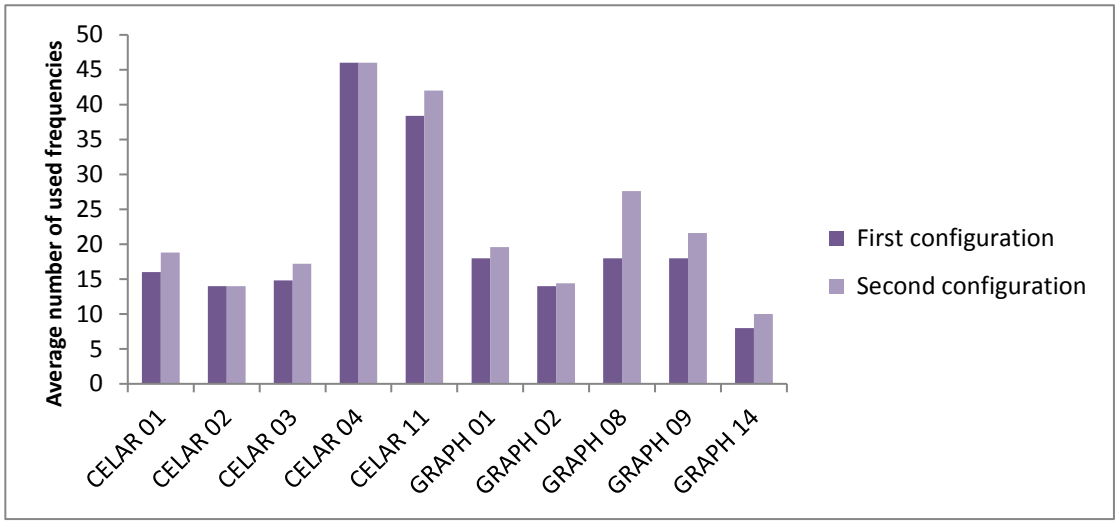

Figure 3.6: Results of TS for MO-FAP using two types of configurations.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is a significant difference between the performances of TS using the two types of configurations. Figure 3.6 shows that the first configuration produces better performance for all the instances except two for which both configurations achieved the optimal solution.

The average run times of TS for the MO-FAP using two types of configurations are shown Figure 3.7.

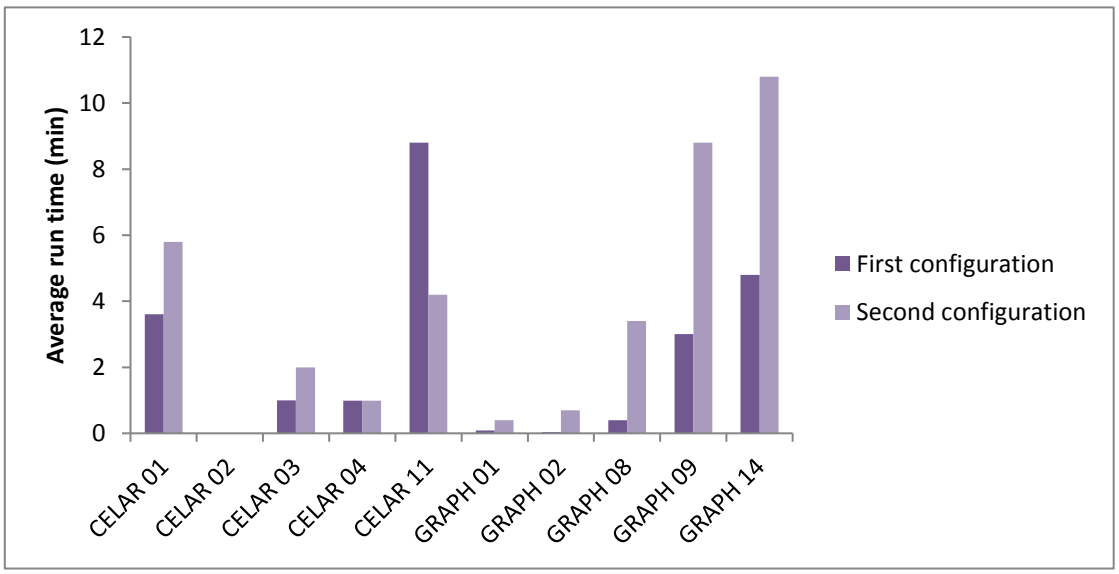

Figure 3.7: Run time of TS for MO-FAP using two types of configurations.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is a significant difference between the run times of TS for the MO-FAP using two types of configurations. Figure 3.7 shows that using the first configuration achieved a better run time for all the instances except CELAR 11. However, the quality of the solutions

which have been found by the first configuration for CELAR 11 was better, although both configurations failed to achieve the optimal solution. Moreover, the total average run time using the first configuration was 22.7 minutes and for the second configuration was 37.1 minutes. Therefore, the best configuration for TS for the MO-FAP is the first configuration, where only the interference constraints are relaxed.

It is of interest to investigate whether TS without significant changes can be successfully applied to other variants of the static FAP (MS-FAP and MI-FAP). Notice that our TS algorithm has been mainly designed to solve the MO-FAP, hence a small number of changes are made. For the MS-FAP, the way of selecting a frequency to be removed in the creating violations phase is changed: the frequency that reduces the maximum value of the used frequencies is selected. For the MI-FAP, the creating violations phase is not required as a zero cost solution does not exist. Hence, TS simply consists of three neighbourhood structures (MNS, SNS and DNS) being searched in turn.

Experimental results show that TS could not achieve feasible solutions for the MS-FAP for all instances except GRAPH 03, where a feasible solution is found, but not the optimal. Furthermore, TS showed poor performance for the MI-FAP. The difficulties in finding good results for this variant of the static FAP using TS which is mainly designed for MO-FAP without significant changes agreed with the findings of [86, 145]. Therefore, this algorithm is not sufficiently effective on all the variants of the static FAP without significant changes. It is likely that more significant changes are required for it to work well on other variants of the static FAP.

3.5.2 Analysis of the Tabu Search Algorithm Process

In this section, different aspects of TS are analysed to investigate four topics: the contribution of each neighbourhood structure, the importance of each neighbourhood structure, the time complexity and the convergence of this algorithm.

3.5.2.1 Contribution of Each Neighbourhood Structure

The contribution of each neighbourhood structure in TS with the first configuration for the CELAR 01 instance is shown in Figure 3.8. The results of each neighbourhood structure during the process of TS are presented as the number of used frequencies and the number of violations in each iteration.

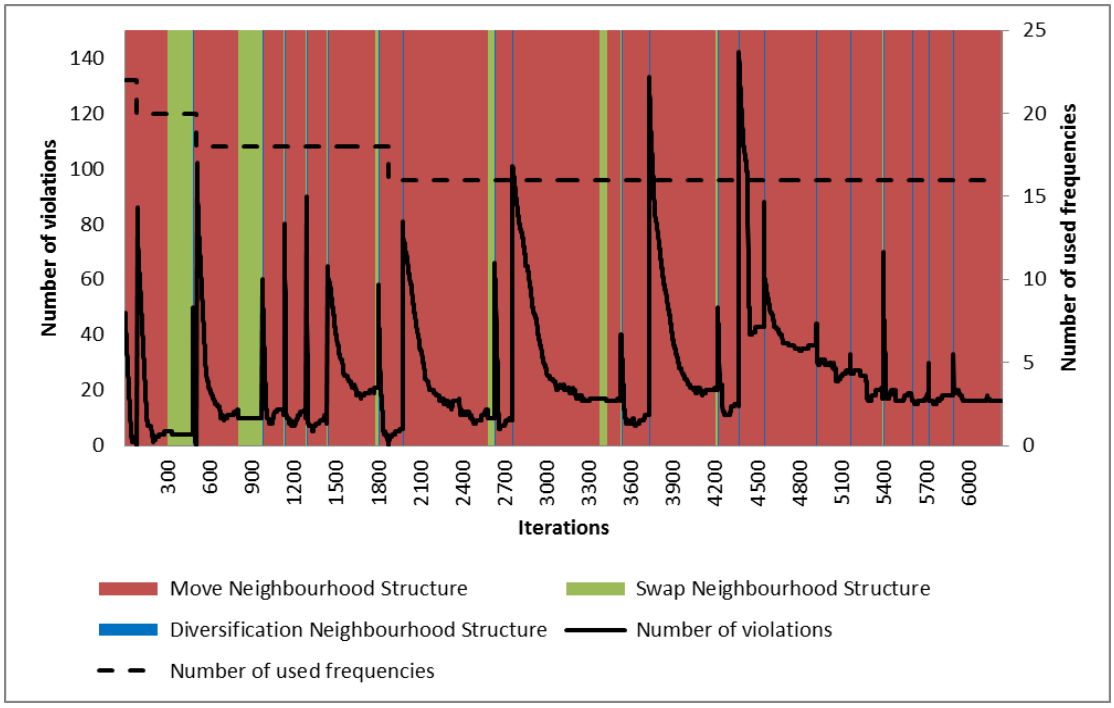

Figure 3.8: The number of used frequencies and violations in each iteration in TS with the first configuration on the CELAR 01 instance.

Figure 3.8 shows that TS started with a feasible initial solution which used 22 frequencies, and then this number was reduced to 16 at the end. Moreover, the most executed neighbourhood structure is MNS, which is represented by the red colour, although all neighbourhood structures have been involved during the process of this algorithm. This reflects the most successful structure in our TS algorithm to reduce the number of violations. Note that MNS is the most commonly used structure in TS for the static FAP in the literature (see e.g. [15, 86, 145]). SNS came as the second structure for reducing the number of violations. This reflects the objective of this structure (i.e. to support MNS) and the limited neighbours of SNS. DNS is executed a very limited number of times and usually results in an increase in the number of violations. This is because this structure aims to diversify the search rather than optimize it. Moreover, it is clear the number of used frequencies has converged. The number of violations tends to increase as the number of frequencies decreases which is to be expected as the problem of finding a feasible solution with fewer available frequencies is more difficult.

3.5.2.2 Importance of Each Neighbourhood Structure

In order to investigate the importance of each neighbourhood structure of TS, 4 different approaches of this algorithm are compared.

- Approach 1: apply the initial solution phase only.
- Approach 2: apply MNS only.
- Approach 3: apply MNS and SNS only.
- Approach 4: apply MNS, SNS and DNS

The performance of the 4 approaches for some of the instances (specifically, CELAR 01, CELAR 03, GRAPH 09 and GRAPH 14) is shown in Figure 3.9. The selected instances are chosen to represent different numbers of requests and constraints.

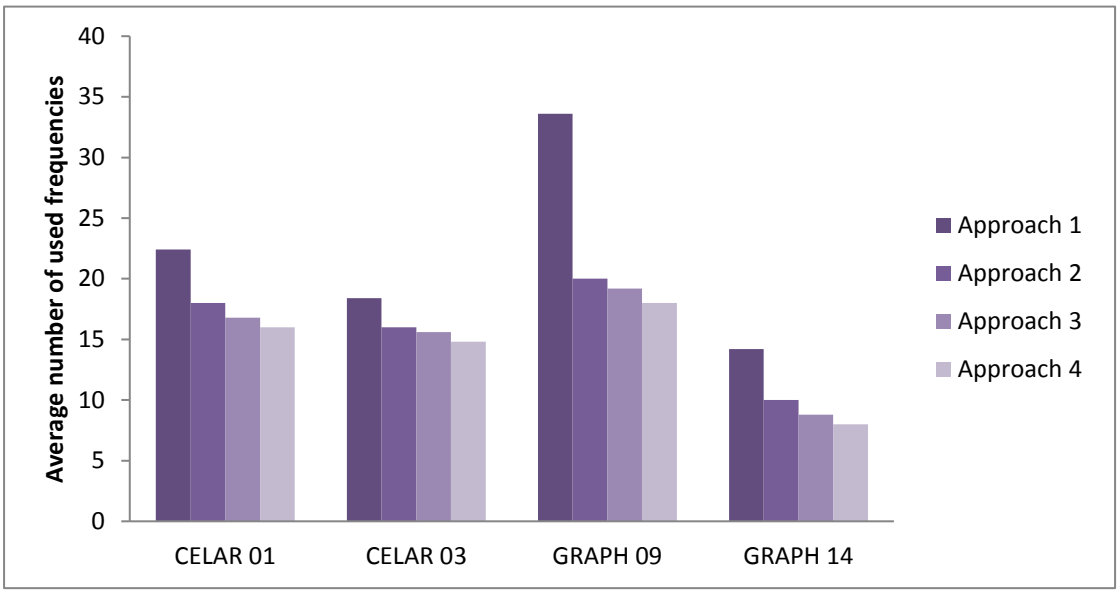

Figure 3.9: Average number of used frequencies for different approaches of the TS algorithm.

Figure 3.9 shows that all the neighbourhood structures play a role. The results of these instances improved after including each neighbourhood structure, which means all the neighbourhood structures are essential to improve solutions.

3.5.2.3 Time Complexity of the Tabu Search Algorithm

The time complexity of TS can be expressed using the big O notation by counting the number of times the key operation, which is assigning a frequency to a request, is performed. Recall that NR is the number of requests and NF is the number of frequencies. In terms of the initial solution phase, the time complexity of the assignment stage is of order $O(NR^2 * NF)$, the allowing infeasible assignment stage is of order $O(NR * NF)$ and the descent method stage is of order $O(NR * NF)$. In terms of the creating violations phase, the time complexity is of order $O(NR * NF)$. The calculation of the initial cost has complexity proportional to $O(NR^2)$. In terms of the improvement

phase, the changes in the cost are calculated efficiently so the time complexity is of order $O(NR)$. The time complexity of MNS is of order $O(NR^2 * NF)$, SNS is of order $O(NR^2)$ and DNS is of order $O(NR^2 * NF^2)$. Hence, the time complexity of our TS algorithm is of order $O(NR^2 * NF^2)$.

3.5.2.4 Convergence of the Tabu Search Algorithm

To investigate the convergence of this algorithm, first note that the number of used frequencies in our TS algorithm never increases. This is because the algorithm consists of reducing the number of used frequencies and seeking for a feasible solution with a fixed number of used frequencies. If a feasible solution is found (i.e. the sub-problem (see Section 3.3.2) is solved), then the number of used frequencies is reduced and the number of iterations is reset to zero. This process is repeated until a feasible solution can no longer be found.

The TS algorithm for the MO-FAP (see Table 3.7) achieved the optimal solution for all the instances except CELAR 11 within 10,000 iterations for each time the sub-problem is considered. Here, TS is run on CELAR 11 for more iterations for each sub-problem (say 50,000 iterations) and the stopping criteria (see Section 3.4.7) are ignored to investigate the convergence of this algorithm. Moreover, TS is executed for five runs, where each run uses different random number streams. Figure 3.10 shows the convergence of TS using the average solutions of the five runs.

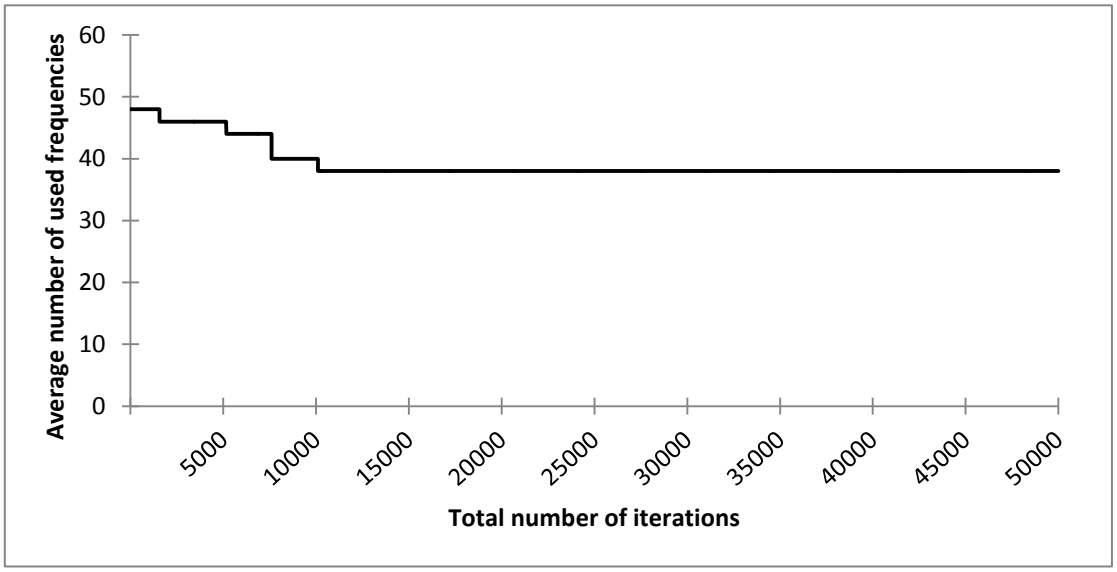

Figure 3.10: The convergence of the TS algorithm on the CELAR 11 instance.

Figure 3.10 shows that TS achieves a feasible solution within 10,000 iterations for each sub-problem. This suggests that the selected number of iterations in this study is an appropriate number based on the convergence experiments.

3.5.3 Results Comparison of the Tabu Search Algorithm

This section compares the performance of our TS algorithm in two subsections. The first subsection compares the performance of this algorithm with existing TS algorithms in the literature. The second subsection compares the performance of our TS algorithm with other algorithms in the literature. Note that a bold number means that the optimal solution is achieved and a dash "-" means that the result is not available.

3.5.3.1 Results Comparison with Existing TS Algorithms

The best found results of our TS algorithm and existing TS algorithms in the literature are given in Table 3.9.

Instance	TS [15]	TS [145]	Our TS	Optimal solution
CELAR 01	18	**16**	**16**	16
CELAR 02	**14**	**14**	**14**	14
CELAR 03	**14**	**14**	**14**	14
CELAR 04	**46**	**46**	**46**	46
CELAR 11	24	**22**	38	22
GRAPH 01	**18**	**18**	**18**	18
GRAPH 02	16	**14**	**14**	14
GRAPH 08	24	20	**18**	18
GRAPH 09	22	22	**18**	18
GRAPH 14	12	10	**8**	8

Table 3.9: Results of TS and existing TS algorithms in the literature.

Table 3.9 shows that our TS algorithm achieved better performance compared with those of TS algorithms proposed in [15, 145]. In fact, our TS algorithm achieved the optimal solution for all the instances except CELAR 11, while the other TS algorithms failed to find the optimal solutions for some instances such as GRAPH 08, GRAPH 09 and GRAPH 14. Additionally, TS proposed in [15] could not achieve the optimal solution for CELAR 11 and GRAPH 02. In contrast, the TS in [145] achieved the optimal solution for both. Overall, our TS algorithm showed competitive performance compared with existing TS algorithms in the literature.

3.5.3.2 Results Comparison with Other Algorithms

This section compares the best found results of our TS algorithm with those of other algorithms in the literature as shown in Table 3.10.

Instance	GENET [16]	Genetic algorithm [94]	Potential reduction [151]	A nonlinear approach [150]	Evolutionary search [34]	Simulating annealing [145]	Variable depth search [145]	Our TS	Optimal solution
CELAR 01	**16**	20	**16**	**16**	-	**16**	**16**	**16**	16
CELAR 02	**14**	**14**	**14**	-	**14**	**14**	**14**	**14**	14
CELAR 03	**14**	16	16	16	**14**	**14**	**14**	**14**	14
CELAR 04	**46**	**46**	**46**	-	-	**46**	**46**	**46**	46
CELAR 11	24	32	-	-	-	24	24	38	22
GRAPH 01	**18**	20	**18**	**18**	**18**	-	-	**18**	18
GRAPH 02	**14**	16	**14**	**14**	**14**	-	-	**14**	14
GRAPH 08	22	-	**18**	**18**	-	-	-	**18**	18
GRAPH 09	22	28	**18**	**18**	-	-	-	**18**	18
GRAPH 14	-	14	10	10	-	-	-	**8**	8

Table 3.10: Results of TS and other algorithms in the literature.

Table 3.10 shows that our TS algorithm achieved competitive performance compared with other algorithms in the literature. Moreover, our TS algorithm is the only algorithm that achieved the optimal solution for GRAPH 14. In contrast, better results for CELAR 11 (not the optimal solution) were found using other algorithms such as simulating annealing and variable depth search in [145]. Note that the performance of the genetic algorithm in [94] is less satisfactory than other algorithms, where this algorithm achieved the optimal solutions for only two instances. Overall, our TS algorithm showed competitive performance compared with other algorithms in the literature.

3.6 Conclusions

This chapter introduced an improved TS algorithm for the static FAP, where this algorithm is mainly designed to solve the MO-FAP. Several novel and existing techniques were used to improve the performance of this algorithm, which are applying the lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model, and hybridising TS with multiple neighbourhood structures, one of which is used as a diversification technique. Moreover, TS was compared in two different types of configurations,

where the first configuration relaxes only interference constraints, while the second configuration relaxes bidirectional and interference constraints.

TS with the first configuration proved effective for the MO-FAP, whereas using the second configuration proved less effective and is therefore rejected. Furthermore, applying TS without significant changes on the other variants of the static FAP was not successful. This finding agrees with what has been found in the literature. It is likely that more significant changes are required for it to work well on other variants of the static FAP.

It was found that our TS algorithm showed competitive performance for the MO-FAP compared with existing TS algorithms and other heuristic algorithms in the literature. In fact, it beat other heuristic algorithms in the literature by achieving the optimal solution for GRAPH 14.

Finally, the research questions which were raised in the beginning of this chapter can be answered as follows:

- *Is TS an effective solution method for the static FAP?*
  Our TS algorithm is an effective solution method for the static FAP. The optimal solution was achieved for all the instances except one (see Table 3.7). Additionally, TS is competitive compared with existing TS (see Table 3.9) and other heuristic algorithms in the literature (see Table 3.10).

- *Is it beneficial to hybridise TS with multiple neighbourhood structures?*
  Each neighbourhood structure in TS plays a role. MNS and SNS aim to improve the quality of the solution using two different techniques, whereas DNS aims to diversify the search. Figure 3.9 shows that the performance of TS improves after including each neighbourhood structure, which means hybridising TS with multiple neighbourhood structures is beneficial.

- *Can TS without significant changes be effective on different variants of the static FAP?*
  TS without significant changes was not successful to solve other variants of the static FAP (MS-FAP and MI-FAP), which agreed with what has been found in the literature. It is likely that more significant changes are required to work well on other variants of the static FAP.

Chapter 4

Ant Colony Optimization for the Static FAP

4.1 Introduction

Ant colony optimization (ACO) is a relatively recent meta-heuristic technique to solve combinatorial optimization problems using indirect communication, which is inspired by how ants cooperate to find the shortest path between their nest and a potential food source. ACO has been successfully applied in the literature to several problems such as traveling salesman problems [41], sequential ordering problems [64], vehicle routing problems [19] and dynamic problems [38].

Although many studies in the literature reflect the success of ACO, it has several shortcomings. One of these is that ACO looks for a better local optimal solution rather than a global optimal solution [18]. Moreover, ACO has a fast convergence rate at the beginning and after a certain number of generations the ants may tend to produce a solution near the local optimum [154]. Finally, ACO is a time consuming method as many elements are used to define the visibility, the trail and parameters and so a lot of computation is needed [33].

There are relatively few papers concerning the application of ACO to solve the static frequency assignment problem (FAP). However, existing ACO algorithms in the literature are unable to find a feasible solution in some instances of the static FAP. Hence, this chapter investigates whether ACO can be improved to be an effective solution method for the static FAP.

In this study, ACO is mainly designed to solve the minimum order FAP (MO-FAP). Several novel and existing techniques are used in this study to improve the performance of ACO. One of these techniques is applying the concept of a well-known graph colouring algorithm, namely recursive largest first (see Section 2.3), which has not been used in ACO for the static FAP in the literature. Furthermore, this study compares ACO using two visibility definitions (see Section 4.2.3). The first definition is based on the number of feasible frequencies, which was previously used in ACO for the graph colouring problem (GCP) [33]. The second one is based on the degree, which was previously used in ACO for the GCP [49]. Additionally, we compare ACO using two trail definitions (see Section 4.2.4). The first one is between requests and frequencies, which was previously used in ACO for the static FAP [109]. Note that ACO in [109] decreases the level of trail for bad solutions, whereas we increase the level of trail for the unassigned requests for all available frequencies in order to be more attractive to be selected. This technique was previously used in ACO for the examination scheduling problem [48]. The second trail definition considered in this study is between requests and requests, which was previously used in ACO for the GCP [49]. Moreover, this chapter investigates whether ACO without significant changes can prove effective on other variants of the static FAP, namely the minimum span FAP (MS-FAP) and minimum interference FAP (MI-FAP). This chapter focuses on the following research questions:

- *Can ACO perform better than tabu search on the static FAP?*
- *Is it beneficial to combine ACO with a local search?*
- *Is ACO an appropriate solution method for the static FAP?*

This chapter is organised as follows: the next subsection gives an overview of ACO. Section 4.2 presents the main components of our ACO algorithm for the static FAP. Results of this algorithm are given and discussed in Section 4.3. Time complexity and

convergence of ACO are discussed in Section 4.4 before the chapter finishes with conclusions.

4.1.1 Overview of Ant Colony Optimization

The basic idea of ACO is inherited from the natural behaviour of real ant colonies. Figure 4.1 shows a group of ants travelling between their nest and some food source.

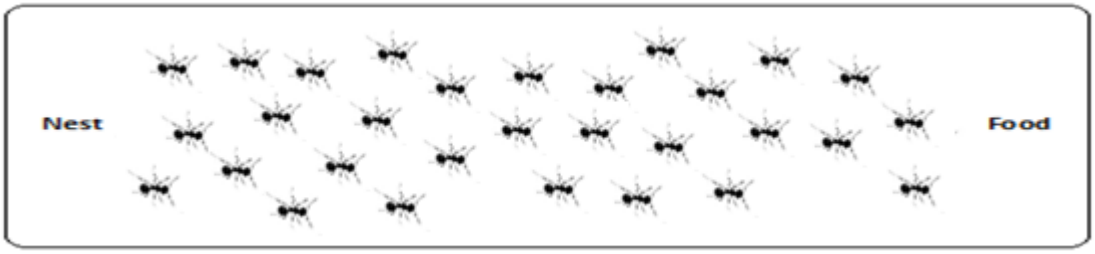

Figure 4.1: Ants in a path between the nest and the food.

If an obstacle appears somewhere in the path between the nest and the food, then when the ants reach the obstacle, at first they randomly choose one way, either right or left, as they are unable to determine which is the shortest route. It is assumed that approximately half of the ants go right and the rest go left, as illustrated in Figure 4.2.

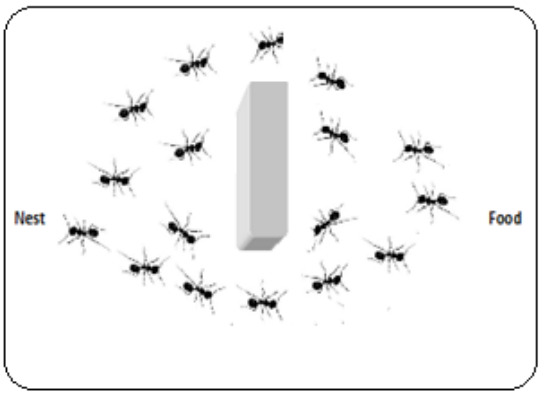

Figure 4.2: Ants can reach the food in two paths.

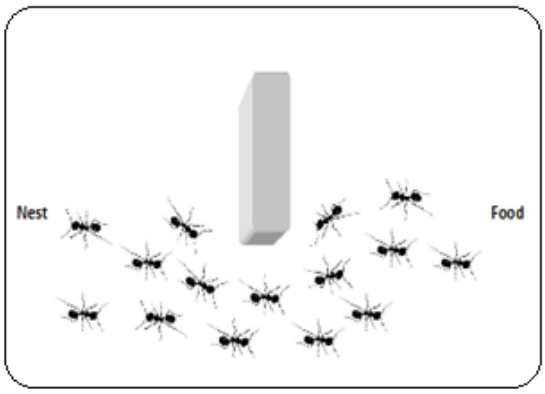

Figure 4.3: Ants find the shortest path.

The group of ants indirectly communicate with one another by leaving trails of pheromone as they travel. As ants find existing trails, they are more likely to follow them depending on the strength of those trails, and they in turn lay down further pheromone, reinforcing the trail. A shorter path is more likely to be followed than a longer path, leading to more trails being laid down along such path. This causes more and more ants to choose the shorter path until eventually all ants have found the shortest path as shown in Figure 4.3.

The behaviour of the ants is exploited in artificial ant colonies for finding the shortest path between two given nodes in a graph, where a path is a sequence of edges. Each artificial ant constructs a solution based on two factors: the visibility and the pheromone trail. The visibility is a measure of the quality of going along each possible edge. The pheromone trail is an indication of the desirability of going along each edge based on the experiences of previous ants. The values of the pheromone trail indicate the strength of the pheromone trail on the corresponding edge based on the experience of previous artificial ants. More formally, moving along the edge (i, j), where i and j are nodes in the graph, during constructing a solution is based on probability p_{ij}, which is given by Formula 4.1.

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} & \text{if } j \in N \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where N is the set of nodes which can be visited by an artificial ant, and τ_{ij} is the pheromone trail between i and j. The visibility η_{ij} is given by Formula 4.2.

$$\eta_{ij} = \frac{1}{L_{ij}}, \tag{4.2}$$

where L_{ij} is the distance of the edge (i, j). The parameters $\alpha, \beta \geq 0$ control the relative significance of the pheromone trail τ_{ij} against the visibility η_{ij}.

After all ants complete their solutions, i.e. one generation is complete, then pheromone trails are updated. The updated pheromone trails guide ants in the following generations to produce better solutions.

There are several variants of ACO. The main difference between them is based on the ways of updating the trail. In the original ACO algorithm in [40], the trail was updated globally after all artificial ants have completed a path between the two given nodes. For each path T^k created by an ant k, the trail is updated by adding $\Delta\tau_{ij}^{k}$, defined by Formula 4.4, to the trail τ_{ij} of edges (i, j) which have been visited in T^k. The trail is updated by Formula 4.3.

$$\tau_{ij} \leftarrow \rho \,.\, \tau_{ij} + \sum_{k} \Delta\tau_{ij}^{k} \tag{4.3}$$

$$\textit{where} \quad \Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{C^k} & \textit{if } T^k \textit{ uses the edge } (i,j) \\ 0 & \textit{otherwise} \end{cases} \tag{4.4}$$

and $\rho \in [0,1)$ is the evaporation parameter, Q is a constant related to the amount of trail laid by ants and C^k is the total distance of T^k. Finally, the solution produced by ACO may be further improved by a local search algorithm.

ACO requires many parameters to be determined. The performance of ACO depends on finding the most appropriate values of these parameters.

4.2 Components of the ACO Algorithm

The components of ACO include solution space and cost function, request and frequency selection, visibility definitions, trail definitions and descent method.

4.2.1 Solution Space and Cost Function

The solution space of ACO is defined as the set of all possible feasible assignments, that is, satisfying all of the constraints. The corresponding cost function is defined as the number of unassigned requests. Note that requests and frequencies in this algorithm are considered as pairs based on the bidirectional constraints (see Equation 1.1) because this configuration showed promising performance (see Section 3.5.1.2)

4.2.2 Request and Frequency Selection

ACO selects a frequency f_j greedily by selecting the one which can be assigned feasibly to the most requests. If there is more than one candidate frequency, then one of them is randomly selected. After that, the frequency f_j is sequentially feasibly assigned to all possible requests until no more can be feasibly assigned. The order of selecting requests from among those that are feasible for f_j is based on probability $p_{r_i f_j}$ given by Formula 4.5.

$$p_{r_i f_j} = \begin{cases} \dfrac{\tau_{r_i f_j}^{\alpha} . \eta_{r_i f_j}^{\beta}}{\sum_{f_k \in \boldsymbol{G}} \tau_{r_i f_k}^{\alpha} . \eta_{r_i f_k}^{\beta}} & \text{if } f_j \in G_{r_i} \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

where G_{r_i} is the set of frequencies which can be feasibly assigned by an artificial ant to the request r_i, The visibility $\eta_{r_i f_j}$ of a request r_i to be assigned a frequency f_j is defined in Section 4.2.3 and the trail $\tau_{r_i f_j}$ is defined in Section 4.2.4.

After that, a different frequency is selected in the same way and this process is repeated until all requests are feasibly assigned, if possible. This process is inherited from a well-known graph colouring algorithm, namely recursive largest first (see Section 2.3). In contrast, ACO for the static FAP in the literature (see e.g. [109, 124]) frequently selects a request based on probability and then assign it to a feasible frequency.

4.2.3 Visibility Definitions

The visibility gives some indication of the desirability of choosing a request based on the experience of previous ants. Hence, the visibility of a request acts as a greedy heuristic. In this study, two types of visibility definition are applied and compared. These two visibilities are defined as follows:

i) Visibility $\eta_{r_i f_j}$ of a request r_i to be assigned a frequency f_j is based on the number of feasible frequencies for r_i (NFF_{r_i}), which is given by Formula 4.6.

$$\eta_{r_i f_j} = \frac{1}{NFF_{r_i}} \quad (4.6)$$

This definition prioritises those requests that have fewer feasible frequencies. This type of visibility definition was previously used in ACO for the graph colouring problem (GCP) [33].

ii) Visibility $\eta_{r_i f_j}$ of a request r_i to be assigned a frequency f_j is based on the degree of r_i (DEG_{r_i}), which is defined as the numbers of unassigned requests that cannot be assigned feasibly to f_j and have a common interference constraint with r_i. This visibility is given by Formula 4.7.

$$\eta_{r_i f_j} = DEG_{r_i} + 1 \tag{4.7}$$

This visibility looks ahead and prioritises requests that have more constraints in common with other requests that cannot be assigned to the frequencies being considered currently. This visibility definition was previously used in ACO for the GCP [49].

Example 4.1 clarifies the probability of selecting a request based on the two different visibility definitions.

Example 4.1:

Assume one of the requests r_1, r_3, r_5 and r_7 needs to be assigned to the selected frequency f_j, where the requests r_9, r_{11} and r_{13} represent unassigned requests that cannot be assigned feasibly to f_j and have a common interference constraint with at least one of the requests r_1, r_3, r_5 and r_7 . The graph colouring model for this problem is shown in Figure 4.4.

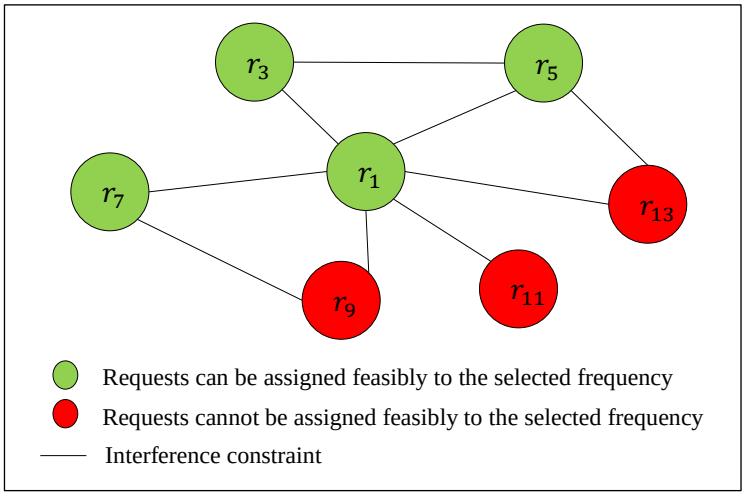

Figure 4.4: Graph colouring model of Example 4.1.

A request from among those that are feasible for the selected frequency f_j is selected based on the probability given by Formula 4.5. Here, assume that the trail and the parameters α and β in Formula 4.5 are set to one. Then, the probability of selecting a request based on the two visibility definitions would be calculated as follows:

i) The probability of selecting each request using the first visibility definition is given in Table 4.1. Note that the number of feasible frequencies of each request (NFF_{r_i}) is invented and cannot be deduced from Figure 4.4.

	r_1	r_3	r_5	r_7	Σ
NFF_{r_i}	1	2	3	4	
$1 / NFF_{r_i}$	1	1/2	1/3	1/4	25/12
$p_{r_i f_j}$	0.48	0.24	0.16	0.12	1

Table 4.1: Requests selection based on probability using the first definition of visibility.

ii) The probability of selecting each request using the second visibility definition is given in Table 4.2. Note that the degree of each request (DEG_{r_i}) can be deduced from Figure 4.4.

	r_1	r_3	r_5	r_7	Σ
DEG_{r_i}	3	0	1	1	
$DEG_{r_i}+1$	4	1	2	2	9
$p_{r_i f_j}$	0.44	0.11	0.22	0.22	1

Table 4.2: Requests selection based on probability using the second definition of visibility.

In both cases, once the probabilities have been calculated, one request is selected probabilistically.

4.2.4 Trail Definitions

The purpose of the trail within ACO is to provide information about previous construction solutions to influence future constructions. In this study, two different trails are defined, where the initial values of these trails are set to one. Moreover, evaporation and updating of these trails are discussed. The definitions of these trails are given as follows:

i) Trail between requests and frequencies ($T_A\ RF$)*:* the key component of a solution is to decide to which frequency each request is assigned. Therefore, the most obvious trail definition is between each request and each frequency, which is also previously used in ACO for the static FAP [124]. The value of the trail indicates the quality of previous solutions when a request is assigned to a frequency.

ii) Trail between requests and requests ($T_A\ RR$)*:* previous work on the graph colouring problem (GCP) in [49] found that a trail between nodes and nodes was more successful than a trail between nodes and colours. This is because the important aspect of a graph colouring solution is not in which colour each node is placed, as the colours are interchangeable. The important aspect is which nodes are placed together in the same colour class. When considering the static FAP, clearly the actual frequency to which each request is assigned is important. However, given the static FAP has the

same underlying model as the GCP, we decided to investigate whether a trail based on which requests are assigned to the same frequencies could be advantageous.

This trail measures the success of previous solutions when requests are assigned to the same frequency using $Average\ T_A\ RR$, which is the average trail between the prospective request and all requests already assigned to the candidate frequency f_j, which is defined by Formula 4.8.

$$Average\ T_A\ RR\{r_i\} = \sum_{r_j \in H, i \neq j} \frac{T_A\ RR\{r_i, r_j\}}{|H|} \tag{4.8}$$

where H is the set of requests already assigned to frequencies f_j.

Example 4.2 clarifies the concept of calculating $Average\ T_A\ RR$, which is given by Formula 4.8.

Example 4.2:

The probability of selecting a request r_i to be assigned a frequency f_j, which is already assigned to three requests, namely r_s, r_t and r_u, and the trail values between r_i and these requests are as follows:

$T_A\ RR\{r_i, r_s\} = 2.0$
$T_A\ RR\{r_i, r_t\} = 1.0$
$T_A RR\{r_i, r_u\} = 0.5$

Based on Formula 4.8, $Average\ T_A\ RR\ \{r_i\} = \frac{2.0 + 1.0 + 0.5}{3} = 1.17$

4.2.4.1 Trail Evaporation

Both types of trail are evaporated after each generation by multiplying the trail by the evaporation parameter, which will be determined experimentally (see Section 4.3.1.4). The trail evaporation can be defined by Formula (4.9).

$$\tau_{r_i f_j} \leftarrow \rho \,.\, \tau_{r_i f_j} \tag{4.9}$$

where the evaporation parameter ρ is in the range [0, 1).

4.2.4.2 Trail Updates

The trails are updated using two reward functions, namely $Cost_1$ and $Cost_2$, which are defined as follows:

> $Cost_1$: counts the number of used frequencies in the current solution. This is appropriate when a solution is feasible.
>
> $Cost_2$: counts the number of unassigned requests in the current solution. This is appropriate when a solution is infeasible.

The values of $T_A\ RF$ could have been updated using Formula 4.10.

$$T_A\ RF\left\{r_i, f_j\right\} = T_A\ RF\left\{r_i, f_j\right\} + \frac{Q}{Cost_1 + Cost_2} \tag{4.10}$$

However, this proved unsatisfactory for two reasons. Firstly, the range of possible amounts added to the trail is relatively small, and therefore fails to distinguish sufficiently between good and bad solutions. So for example, if the optimal number of frequencies is 40, and Q is equal to 10, then the amounts added to the trail for 4 different feasible solutions are given in Table 4.3.

	Solution 1	Solution 2	Solution 3	Solution 4
No. used frequencies	41	42	43	44
Trail update	0.24	0.24	0.23	0.23

Table 4.3: Example of trail update values.

These values are not significantly different, making it difficult for ACO to learn. Additionally, the amount added to the trail differs from instance to instance depending on the typical number of frequencies. In order to avoid this problem, a better way of trail updates is applied and given by Formula 4.11.

$$T_A\ RF\left\{r_i, f_j\right\} = T_A\ RF\left\{r_i, f_j\right\} + \frac{Q}{Cost_1 + Cost_2 - Best + 1} \tag{4.11}$$

where $Best$ is the best minimum number of used frequencies found so far in the search. Note that $Cost_1 + Cost_2 - Best$ can be equal to 0 when $Cost_1 = Best$ and $Cost_2 = 0$. Thus, we add 1 to the denominator of the last term in Formula 4.11. A similar trail update function was previously used in ACO for the GCP [49].

Formula 4.11 has the advantage that any solution that uses a number of frequencies more than the best solution does will have significantly larger trail update values. So

for example, if the optimal number of frequencies is 40, and Q is equal to 10, then the amounts added to the trail for 4 different feasible solutions are given in Table 4.4.

	Solution 1	Solution 2	Solution 3	Solution 4
No. used frequencies	41	42	43	44
Trail update	10	5	3.33	2.50

Table 4.4: Example of trail update values with improved trail update function.

Similarly, the values of $T_A\ RR$ are updated using Formula 4.12.

$$T_A\ RR\{r_i, r_j\} = T_A\ RR\{r_i, r_j\} + \frac{Q}{Cost_1 + Cost_2 - Best + 1} \tag{4.12}$$

Another problem of trail updates is that only requests that have been assigned to frequencies are updated. Therefore, the trail values on any unassigned requests are not increased, meaning such requests are likely to be selected even later in the following construction processes. As we would prefer to consider them earlier in the construction process, the trail is increased on each unassigned request for all available frequencies. This idea was previously used in ACO for the examination scheduling problem [48].

4.2.5 Descent Method

This method is executed only when no feasible solution can be found by all ants in a generation. In such generations, the descent method is executed only for one ant which constructs the infeasible solution with the minimum number of unassigned requests. First, these requests are assigned to the frequencies which lead to the least number of violations. Then, the descent method aims to reduce the number of violations with a fixed number of frequencies to find a feasible solution, if possible. The description of the descent method can be found in Section 3.4.4.3.

4.2.6 The ACO Algorithm Implementation

ACO consists of a given number of generations, each of which contains a given number of ants, where each ant individually constructs a solution. Each ant starts constructing a solution by selecting a frequency to be assigned to all possible feasible requests. The process is repeated until no frequencies can be selected (see Section 4.2.2). After all ants in the current generation construct their solutions, if no feasible solution can be found, then the descent method (see Section 4.2.5) is used to attempt to achieve a feasible solution. Then, the trail is evaporated and updated (see Section

4.2.4.1 and 4.2.4.2). After that, the next generation is executed by the same process. The overall structure of the ACO algorithm is illustrated in Figure 4.5.

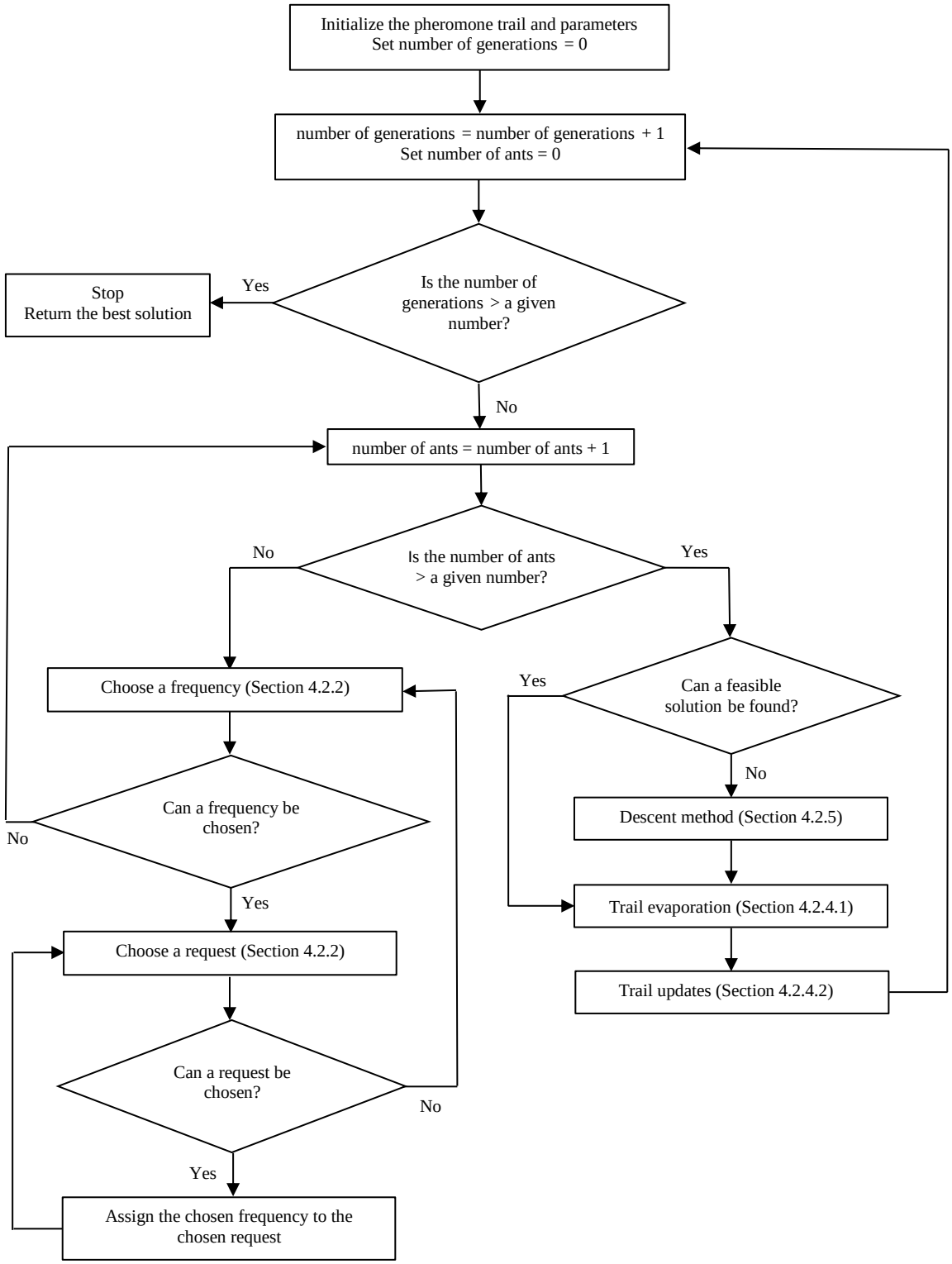

Figure 4.5: Overall structure of our ACO algorithm for the static FAP.

4.3 Experiments and Results

This section presents and compared the performance of ACO in three sections. The first section gives the results of ACO for the static FAP. The second section compares the performance of ACO with existing ACO algorithms in the literature. Finally, the performance of ACO is compared with other algorithms in the literature.

ACO is implemented in FORTRAN 95 and all experiments were conducted on a 3.0 GHz Intel Core I3-2120 Processor (2nd Generation) with 8GB RAM and a 1TB Hard Drive.

4.3.1 Results Comparison of the ACO Algorithm

In this study, the number of generation of ACO is 100, where this number is selected based on experiments. Moreover, the performance of ACO is compared based on several options of the following components:

1. The number of ants,
2. The trail definition,
3. The visibility definition,
4. The parameters $\alpha,\ \beta\ and\ \rho$ (described in Section 4.1.1).

Different values of the number of ants, two options of the trail definition and two options for visibility definition are compared. For the parameters α, β and ρ, three values of each parameter are tested. The values considered for each parameter in this study are given in Table 4.5.

α	β	ρ
1	1	0.80
3	2	0.90
5	3	0.95

Table 4.5: The considered values of the parameters α, β and ρ.

Another parameter that might need to be considered is the parameter Q for trail updates. However, from literature and our experiments, it was found that this parameter has no major effect on the algorithm [45]. Therefore, the value of Q is set to 10 throughout this study.

By considering all these options, there are 756 versions of ACO to be compared. Moreover, each version is tested on 10 instances with 5 runs being performed on each

instance. Therefore, considering all the versions of ACO take excessive time. Hence, the comparison is made for each component while fixing the others; i.e. first, different numbers of ants are compared while fixing the remaining components. After selecting the best number of ants, the two different trail definitions are compared. After that, two definitions of the visibility are compared and finally, different values of the parameters (α, β and ρ) are compared in the same way.

The experiments are run using CELAR and GRAPH datasets for the static FAP. Recall that the results of the MO-FAP refer to the number of used frequencies in a feasible solution. ACO is run on each instance five times and each run uses a different random number stream, where this number is chosen based on experiments. Moreover, the best, the worst and the average solution, and the average run time are calculated.

4.3.1.1 The Number of Ants

This section discusses the effects of the number of ants on the performance of ACO. Different numbers of ants are chosen to be compared, where these numbers are selected based on experiments. The selected numbers of ants are 5, 10, 15, 20, 25, 30 and 35. In order to observe the impact of the number of ants on the performance of ACO, other options are fixed, which are the visibility definition, the trail definition and the values of the parameters α, β and ρ. In this stage, the visibility definition is based on the number of feasible frequencies, the trail definition is $T_A\, RF$ and the parameters α, β and ρ take the default values as given in Table 4.6.

α	β	ρ	Q
5	2	0.90	10

Table 4.6: The default values of the parameters.

The ACO algorithm is tested on a subset of instances (specifically, CELAR 01, CELAR 03, GRAPH 01 and GRAPH 02). The selected subset of instances represents different numbers of requests and constraints. The effect of the number of ants on the performance of this algorithm is shown in Figure 4.6.

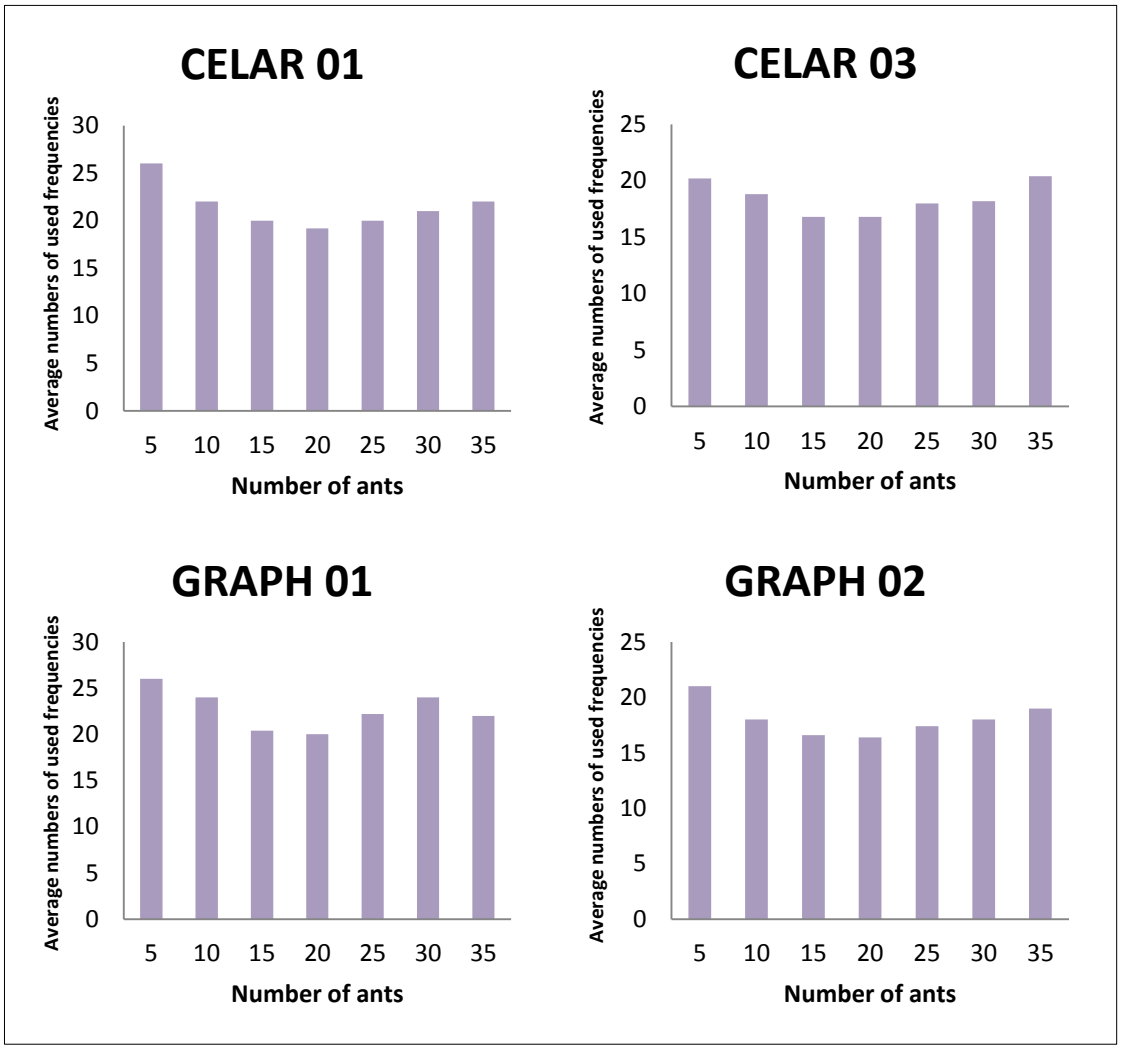

Figure 4.6: The effect of the number of ants on the performance of the ACO algorithm.

Figure 4.6 shows that the best results are found when the number of ants is 20. Hence, the selected number of ants in our ACO algorithm is 20.

4.3.1.2 The Trail Definitions

Two different trails are compared by fixing the number of ants to 20, the visibility to the number of feasible frequencies, and the parameters to the default values (see Table 4.6). The results of ACO using the first trail definition $T_A\ RF$ are given in Table 4.7, while Table 4.8 gives the results of ACO using the second trail definition $T_A\ RR$. The best, the worst and the average solution, and the average run time of these two tables are compared and the better ones are shown in bold. Note that a dash "-" means that a feasible solution could not be found.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	20	**19.2**	16	**1.4 hrs**
CELAR 02	14	14	**14.0**	14	**11.2 min**
CELAR 03	16	18	**16.8**	14	**31.1 min**
CELAR 04	46	46	**46.0**	46	**55.8 min**
CELAR 11	-	-	-	22	-
GRAPH 01	20	20	**20.0**	18	**18.3 min**
GRAPH 02	16	18	**16.4**	14	**29.8 min**
GRAPH 08	24	24	**24.0**	18	**30.1 min**
GRAPH 09	-	-	-	18	-
GRAPH 14	10	12	**11.6**	8	**59.8 min**

Table 4.7: Results of ACO for the MO-FAP using the trail $T_A\ RF$.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	22	24	23.2	16	1.6 hrs
CELAR 02	14	14	**14.0**	14	18.2 min
CELAR 03	16	18	17.6	14	41.8 hrs
CELAR 04	-	-	-	46	-
CELAR 11	-	-	-	22	-
GRAPH 01	20	22	21.2	18	27.8 min
GRAPH 02	18	20	19.6	14	44.2 min
GRAPH 08	-	-	-	18	-
GRAPH 09	-	-	-	18	-
GRAPH 14	10	12	**11.6**	8	1.3 hrs

Table 4.8: Results of ACO for the MO-FAP using the trail $T_A\ RR$.

The performance of ACO using $T_A\ RF$ is better than using $T_A\ RR$ for 6 out of 10 instances. Moreover, using $T_A\ RF$ has better run times for all the instances. The performance of ACO using the two types of trail definitions is shown in Figure 4.7 (for the instances in which feasible solutions are found).

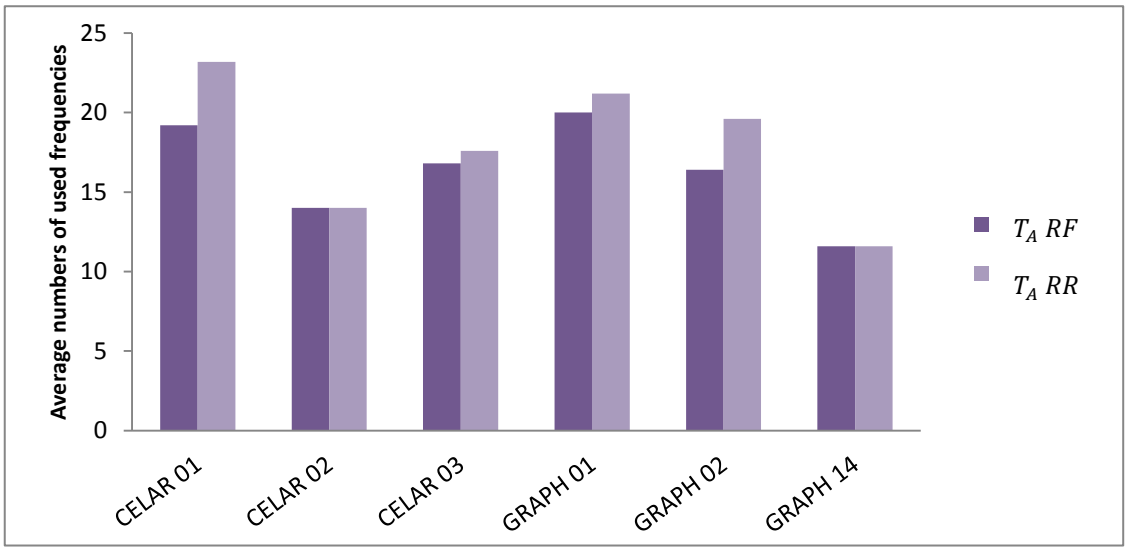

Figure 4.7: The performance of ACO using two types of trail definitions.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is a significant difference between the performances of ACO using $T_A\ RF$ and $T_A\ RR$.

The run time of ACO using these two trails is shown in Figure 4.8 (for the instances in which feasible solutions are found).

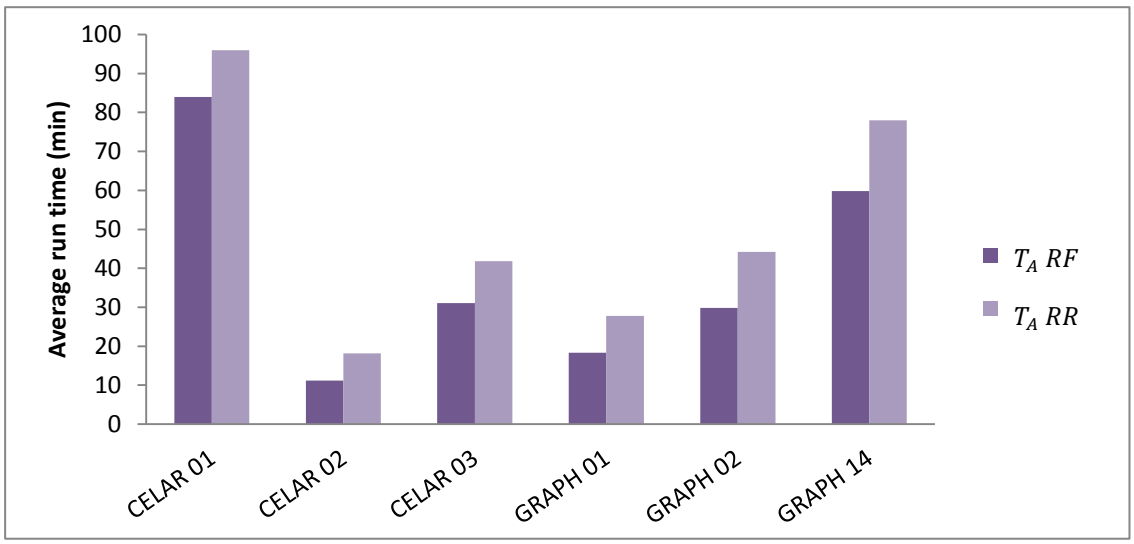

Figure 4.8: The average run time of ACO using two types of trail definitions.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is a significant difference between the run times of ACO using $T_A\ RF$ and $T_A\ RR$. Overall, using $T_A\ RF$ resulted in better performance of ACO, hence it is selected as the definition of trail in this algorithm.

4.3.1.3 The Visibility Definitions

Here, the performance of ACO using the two different visibility definitions is compared. The results of ACO using the first visibility definition were previously shown in Table 4.7, while those for the second visibility definition are given in Table 4.9. Note that a bold number in Table 4.9 means it is not worse than the corresponding one in Table 4.7 and a dash "-" means that a feasible solution could not be found.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	20	22	20.8	16	1.8 hrs
CELAR 02	14	14	**14.0**	14	27.2 min
CELAR 03	18	18	18.0	14	51.1 min
CELAR 04	46	46	**46.0**	46	1.2 hrs
CELAR 11	-	-	-	22	-
GRAPH 01	22	22	22.0	18	29.8 min
GRAPH 02	20	22	20.4	14	59.2 min
GRAPH 08	-	-	-	18	-
GRAPH 09	-	-	-	18	-
GRAPH 14	12	12	12.0	8	1.5 hrs

Table 4.9: Results of ACO for the MO-FAP using the second definition of visibility.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the performance of ACO is significantly better when the first definition of visibility is used.

4.3.1.4 The Parameters Values

Here, different values of each parameter are compared to find the appropriate values. In order to observe the impact of each parameter on the performance of ACO, the values of each parameter are compared as follows: different values of the parameter α are compared while fixing the other parameters (β and ρ) to the default values (see Table 4.6). After that, the best value of the parameter α is fixed to compare different values of the parameter β. Finally, different values of the last parameter ρ are compared. Note that a bold number shows the best result among different values of the parameter being considered and a dash "-" means that a feasible solution could not be found.

The performance of ACO with different values of the parameter α while $\beta = 2$ and $\rho = 0.90$ (their default values) is shown in Table 4.10.

Instance	Average solution		
	$\alpha = 1$	$\alpha = 3$	$\alpha = 5$
CELAR 01	20.4	20.0	**19.2**
CELAR 02	**14.0**	**14.0**	**14.0**
CELAR 03	18.0	18.0	**16.8**
CELAR 04	-	**46.0**	**46.0**
CELAR 11	-	-	-
GRAPH 01	24.0	22.0	**20.0**
GRAPH 02	20.0	18.4	**16.4**
GRAPH 08	-	**24.0**	**24.0**
GRAPH 09	-	-	-
GRAPH 14	12.1	12.4	**11.6**

Table 4.10: Results of ACO using different values of the parameter α.

Table 4.10 shows that the best value of the parameter α is 5. The performance of ACO with different values of the parameter β while $\alpha = 5$ (its best value) and $\rho = 0.90$ (its default value) is shown in Table 4.11.

Instance	Average solution		
	$\beta = 1$	$\beta = 2$	$\beta = 3$
CELAR 01	20.0	**19.2**	20.4
CELAR 02	14.2	**14.0**	**14.0**
CELAR 03	18.8	**16.8**	18.8
CELAR 04	**46.0**	**46.0**	**46.0**
CELAR 11	-	-	-
GRAPH 01	22.0	**20.0**	**20.0**
GRAPH 02	18.4	**16.4**	16.8
GRAPH 08	-	**24.0**	-
GRAPH 09	-	-	-
GRAPH 14	12.2	**11.6**	12.4

Table 4.11: Results of ACO using different values of the parameter β.

Table 4.11 shows that the best value of the parameter β is 2. The performance of ACO with different values of the parameter ρ while $\alpha = 5$ and $\beta = 2$ (their best values) is shown in Table 4.12.

Instance	Average solution		
	$\rho = 0.80$	$\rho = 0.90$	$\rho = 0.95$
CELAR 01	**19.2**	**19.2**	19.4
CELAR 02	**14.0**	**14.0**	**14.0**
CELAR 03	**16.8**	**16.8**	18.4
CELAR 04	**46.0**	**46.0**	**46.0**
CELAR 11	-	-	-
GRAPH 01	**20.0**	**20.0**	20.4
GRAPH 02	18.0	**16.4**	16.8
GRAPH 08	**24.0**	**24.0**	**24.0**
GRAPH 09	-	-	-
GRAPH 14	12.6	**11.6**	12.8

Table 4.12: Results of ACO using different values of the parameter ρ.

Table 4.12 shows that the best value of the parameter ρ is 0.90.

The results of ACO using the best values of the parameters ($\alpha = 5, \beta = 2, \rho = 0.90$) are shown in Table 4.13, where a bold number means the optimal solution is found.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	20	19.2	16	1.4 hrs
CELAR 02	**14**	**14**	14.0	14	11.2 min
CELAR 03	16	18	16.8	14	31.1 min
CELAR 04	**46**	**46**	46.0	46	55.8 min
CELAR 11	-	-	-	22	-
GRAPH 01	20	20	20.0	18	18.3 min
GRAPH 02	16	18	16.4	14	29.8 min
GRAPH 08	24	24	24.0	18	30.1 min
GRAPH 09	-	-	-	18	-
GRAPH 14	10	12	11.6	8	59.8 min

Table 4.13: The best results of the ACO algorithm for the MO-FAP.

Table 4.16 shows that ACO managed to achieve a feasible solution for all the instances except two (CELAR 11 and GRAPH 09). In fact, ACO achieved the optimal solutions for only two instances (CELAR 02 and CELAR 04).

Furthermore, it is of interest to investigate whether ACO without significant changes can be successfully applied to other variants of the static FAP (MS-FAP and MI-FAP). Notice that ACO is mainly designed to solve the MO-FAP. Therefore, a small number of changes are made to ACO. For the MS-FAP, the same ACO algorithm is used, but in the frequency selection, the selected frequency is changed to be the one that has the minimum value and is feasible for the most requests. It is found that ACO has poor performance for MS-FAP and MI-FAP, which agrees with what has been found by the tabu search algorithm in Chapter 3. It is likely that more significant changes are required to work well on other variants of the static FAP.

4.3.1.5 The Descent Method

Here, we investigate whether it is beneficial to combine ACO with a descent method by implementing ACO without it. After that, the performance of ACO with and without the descent method is compared. The performance of ACO without the descent method is shown in Table 4.14.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	20	18.7	16	1.2 hrs
CELAR 02	**14**	**14**	14.0	14	9.8 min
CELAR 03	16	18	16.8	14	26.8 min
CELAR 04	**46**	**46**	46.0	46	46.1 min
CELAR 11	-	-	-	22	-
GRAPH 01	20	20	20.0	18	12.2 min
GRAPH 02	16	18	16.4	14	22.4 min
GRAPH 08	-	-	-	18	-
GRAPH 09	-	-	-	18	-
GRAPH 14	10	12	11.6	8	52.8 min

Table 4.14: Results of ACO for the MO-FAP without using the descent method.

Table 4.14 shows that ACO without the descent method struggled to find a feasible solution for CELAR 11, GRAPH 08 and GRAPH 09. Using the descent method helped ACO to achieve feasible solutions for GRAPH 08 (see Table 4.13). Overall, combining ACO with the descent method led to a better performance compared with ACO without the descent method.

4.3.2 Results Comparison with Existing ACO Algorithms

The performance of our ACO is compared with existing ACO in the literature. To the best of my knowledge, only one published research [109] (described in Section 2.5.1) applied ACO for the MO-FAP using CELAR and GRAPH datasets. Table 4.15 shows the results in the form given in [109], i.e. in the form of (*y*) where *y* is the number of violations. Note that *y* is equal to 0 means a feasible solution is found.

Instance	ACO [109]	Our ACO
CELAR 01	(0)	(0)
CELAR 02	(0)	(0)
CELAR 03	(0)	(0)
CELAR 04	(8)	(0)
CELAR 11	(2)	(6)
GRAPH 01	(0)	(0)
GRAPH 02	(0)	(0)
GRAPH 08	(0)	(0)
GRAPH 09	(0)	(12)
GRAPH 14	(0)	(0)

Table 4.15: Results of ACO and existing ACO algorithm in the literature.

Table 4.15 shows that both of the algorithms struggled to find a feasible solution for CELAR 11. Moreover, ACO in [109] could not achieve a feasible solution for CELAR 04, whereas our ACO could. In contrast, our ACO could not achieve a feasible solution for GRAPH 09, whereas ACO in [109] could. Overall, both of the ACO algorithms performing equally well.

4.3.3 Results Comparison with Other Algorithms

This section compares the best found results of our ACO algorithm with those of other algorithms in the literature and our tabu search algorithm (see Chapter 3) as shown in Table 4.16, where a bold number means that the optimal solution is achieved and a dash "-" means that the result is not available.

Instance	Tabu search [15]	GENET [16]	Genetic algorithm [94]	Potential reduction [151]	A nonlinear approach [150]	Evolutionary search [34]	Tabu search [145]	Simulating annealing [145]	Variable depth search [145]	Our tabu search algorithm	Our ACO	Optimal solution
CELAR 01	18	**16**	20	**16**	**16**	-	**16**	**16**	**16**	**16**	18	16
CELAR 02	**14**	**14**	**14**	**14**	**-**	**14**	**14**	**14**	**14**	**14**	**14**	14
CELAR 03	**14**	**14**	16	16	16	**14**	**14**	**14**	**14**	**14**	16	14
CELAR 04	**46**	**46**	**46**	**46**	**-**	-	**46**	**46**	**46**	**46**	**46**	46
CELAR 11	24	24	32	-	-	-	**22**	24	24	38	-	22
GRAPH 01	**18**	**18**	20	**18**	**18**	**18**	**18**	-	-	**18**	20	18
GRAPH 02	16	**14**	16	**14**	**14**	**14**	**14**	-	-	**14**	16	14
GRAPH 08	24	22	-	**18**	**18**	-	20	-	-	**18**	24	18
GRAPH 09	22	22	28	**18**	**18**	-	22	-	-	**18**	-	18
GRAPH 14	12	-	14	10	10	-	10	-	-	**8**	10	8

Table 4.16: Results of ACO and other algorithms in the literature.

Table 4.16 shows that our ACO algorithm achieved competitive results for CELAR 02, CELAR 04 and GRAPH 14, while it achieved reasonable results for other instances. In contrast, our ACO algorithm struggled to achieve a feasible solution for CELAR 11 and GRAPH 09. Additionally, our ACO algorithm and the genetic algorithm in [34] are less satisfactory than the other algorithms as these achieved the optimal solutions for only two instances. Note that our tabu search algorithm found the optimal solution for the highest number of instances in Table 4.16.

4.4 Time Complexity and Convergence of ACO

The time complexity of ACO can be expressed using the big *O* notation by counting the number of times the key operation, which is assigning a frequency to a request, is performed. The first step of ACO is selecting a frequency and this requires $O(NR^2 * NF)$, while selecting requests to be assigned the selected frequency is of order $O(NR^2 * NF)$. For each ant, the cost function must be calculated from scratch and this has complexity of $O(NR^2)$. The trail update requires $O(NR * NF)$ and the descent method requires $O(NR^2 * NF)$. Hence, the time complexity of ACO is of order $O(NR^2 * NF)$.

To investigate the convergence of ACO, this algorithm is executed for 500 generations for the GRAPH 01 instance. It is run with five different random streams and the average of the solutions is shown in Figure 4.10.

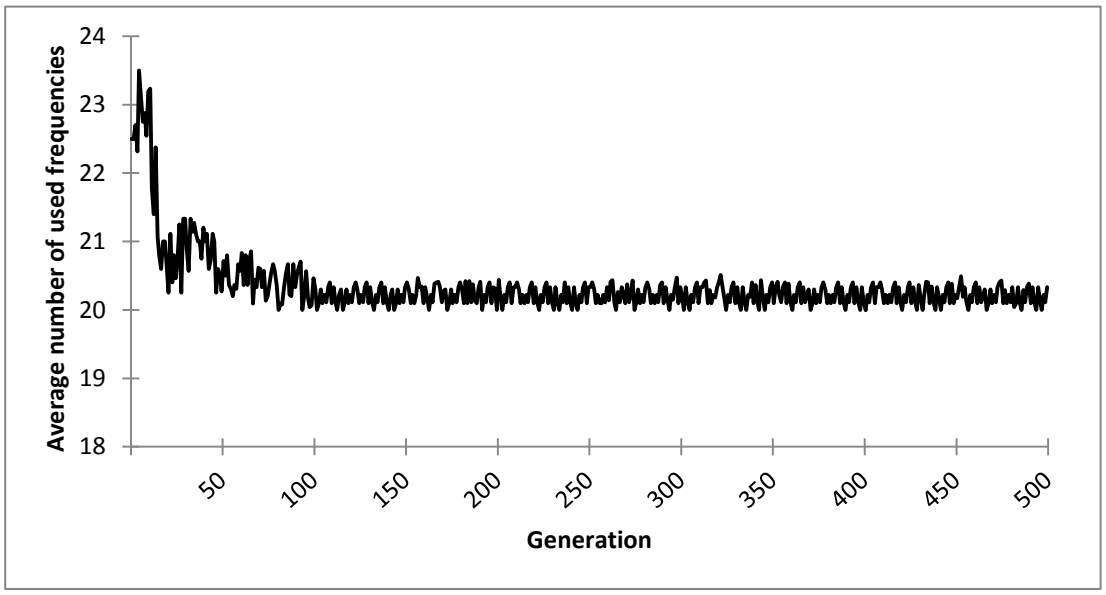

Figure 4.9: The convergence of the ACO algorithm on the GRAPH 01 instance.

Figure 4.9 shows the ACO algorithm converged within 100 generations. Furthermore, similar convergence has been found for other instances. Therefore, this convergence experiment suggests that the appropriate number of generations in our ACO for the static FAP is 100.

4.5 Conclusions

In this study, an improved ACO was introduced for the static FAP, where ACO is mainly designed to solve the MO-FAP. One of the techniques which was applied to improve the performance of ACO is the recursive largest first (RLF). In fact, applying RLF aims to improve the performance of selecting frequencies and requests to be assigned. Moreover, ACO was compared using two trail definitions and two visibility definitions. It was found that using the trail between requests and frequencies led to better performance than the other trail definition. Moreover, using the visibility definition based on the number of feasible frequencies (Formula 4.6) resulted in better performance than another visibility definition. Furthermore, several values for the parameters α, β, ρ were compared and the best values are 5, 2 and 0.90, respectively.

ACO is combined with a descent method to achieve better results when no feasible solution can be found in a generation. In such generations, the descent method is executed for only one ant which constructs the infeasible solution with the minimum number of unassigned requests. Overall, our ACO algorithm performed similarly to ACO in the literature, whereas it showed poor performance compared with other algo-

rithms in the literature. Hence, other heuristic algorithms need to be considered to achieve the optimal solution or better results. In chapter 5, one of the popular heuristic algorithms called hyper heuristics is considered to solve the static FAP.

Finally, the research questions of this chapter can be answered as follows:

- *Can ACO perform better than tabu search on the static FAP?*
  ACO cannot perform better than tabu search on the static FAP (see Section 4.3.3). Indeed, ACO struggled to find a feasible solution for some instances (see Table 4.13), so this algorithm is not strong enough.

- *Is it beneficial to combine ACO with a local search?*
  Combining ACO with a local search improves the results of our ACO algorithm (see Section 4.3.1.5).

- *Is ACO an appropriate solution method for the static FAP?*
  The comparison of the performance of ACO with other algorithms in the literature (see Table 4.16) suggests that ACO is not the appropriate solution method for the static FAP due to the poor performance of this algorithm.

Chapter 5

Hyper Heuristic for the Static FAP

5.1 Introduction

Hyper heuristic (HH) can be thought of as an algorithm that combines multiple heuristics to solve hard combinatorial optimization problems. The concept of HH is based on the idea that, as each heuristic has strengths and weaknesses, combining several heuristics may lead to an improved performance. Such heuristics are called low level heuristics (LLHs) and are managed by HH. The criteria to select one of these at each step is usually problem independent. This algorithm is an iterative process of two stages: heuristic selection and move acceptance.

There are two main types of HH, namely constructive and improvement algorithms [22]. The constructive HH algorithm constructs a solution from scratch, whereas the improvement HH algorithm starts with an initial solution and aims to improve it. In this thesis, an improvement heuristic algorithm was considered in Chapter 3, namely tabu search (TS), whereas a constructive heuristic algorithm was discussed in Chapter 4, namely ant colony optimization (ACO), and the former proved far superior. There-

fore, the improvement approach is adopted for HH in this study. There are relatively few papers that apply HH for the static frequency assignment problem (FAP). However, to the best of my knowledge, no existing papers in the literature apply HH on the static FAP datasets considered in this thesis. Hence, this is the first attempt to solve such datasets using HH.

In this chapter, a HH algorithm is applied to the static FAP and is mainly designed for the minimum order FAP (MO-FAP) using several novel and existing techniques. One of these is using a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model (see Section 3.2). These lower bounds are used to ensure that we never waste time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bounds. Moreover, applying simple and advanced LLHs associated with an independent tabu list for each LLH is aimed to make this algorithm more efficient and different from other HHs for the static FAP in the literature (see e.g. [96, 97]). This chapter focuses on the following research questions:

- *Can HH perform better than TS and ACO on the static FAP?*
- *What is the best mechanism for selecting the LLHs?*
- *Is HH an appropriate solution method for the static FAP?*

This chapter is organized as follows: Section 5.2 presents an overview of our HH algorithm for the static FAP. Section 5.3 gives the main components of our HH algorithm. In Section 5.4, the results of HH are given and analysed. Then, the performance of HH is compared with the algorithms considered in this thesis and other algorithms in the literature. The time complexity and the convergence of our HH algorithm are discussed in Section 5.5. Finally, this chapter is closed with conclusions.

5.2 Overview of the Hyper Heuristic Algorithm

5.2.1 Solution Space and Cost function

It was found in [47] that the interference constraints are the most difficult constraints to be satisfied. Hence, this type of constraint can be relaxed and the solution space is defined here as the set of all possible solutions that satisfy bidirectional, domain and pre-assignment constraints. The cost function is defined as the number of broken in-

terference constraints, also known as the number of violations. This configuration is different from other configurations in HH for the static FAP in the literature where no constraints are relaxed (see e.g. [96, 97]). Note that requests and frequencies are considered as pairs based on the bidirectional constraints (see Equation 1.1) because this configuration showed promising performance (see Section 3.5.1.2).

Using the solution space which relaxes the interference constraints creates the following sub-problem: minimizing the number of violations with a fixed number of used frequencies. If a solution with zero violations (a feasible solution) is found, then the number of used frequencies is reduced using the creating violations phase (see Section 5.3.2) and then the sub-problem is revisited. The process is repeated until a feasible solution can no longer be found. This process is the same as tabu search in Chapter 3.

5.2.2 Structure of the Hyper Heuristic Algorithm

HH starts with the initial solution phase (see Section 5.3.1). Assume the initial solution is feasible. Then, the creating violations phase (see Section 5.3.2) is used to reduce the number of used frequencies. After that, HH is applied using LLHs (see Section 5.3.3) to find a feasible solution with a fixed number of used frequencies. One of the LLHs is selected in each iteration based on the selection mechanism (see Section 5.3.4) to find a new solution. This solution is accepted or rejected based on the move acceptance criteria (see Section 5.3.5), which accepts worse solutions a limited number of times to diversify the search. After that, the process continues until one of the stopping criteria is met (see Section 5.3.6). If the initial solution is infeasible, then the creating violations phase is skipped and all available frequencies are allowed. The overall structure of the HH algorithm considered in this study is illustrated in Figure 5.1.

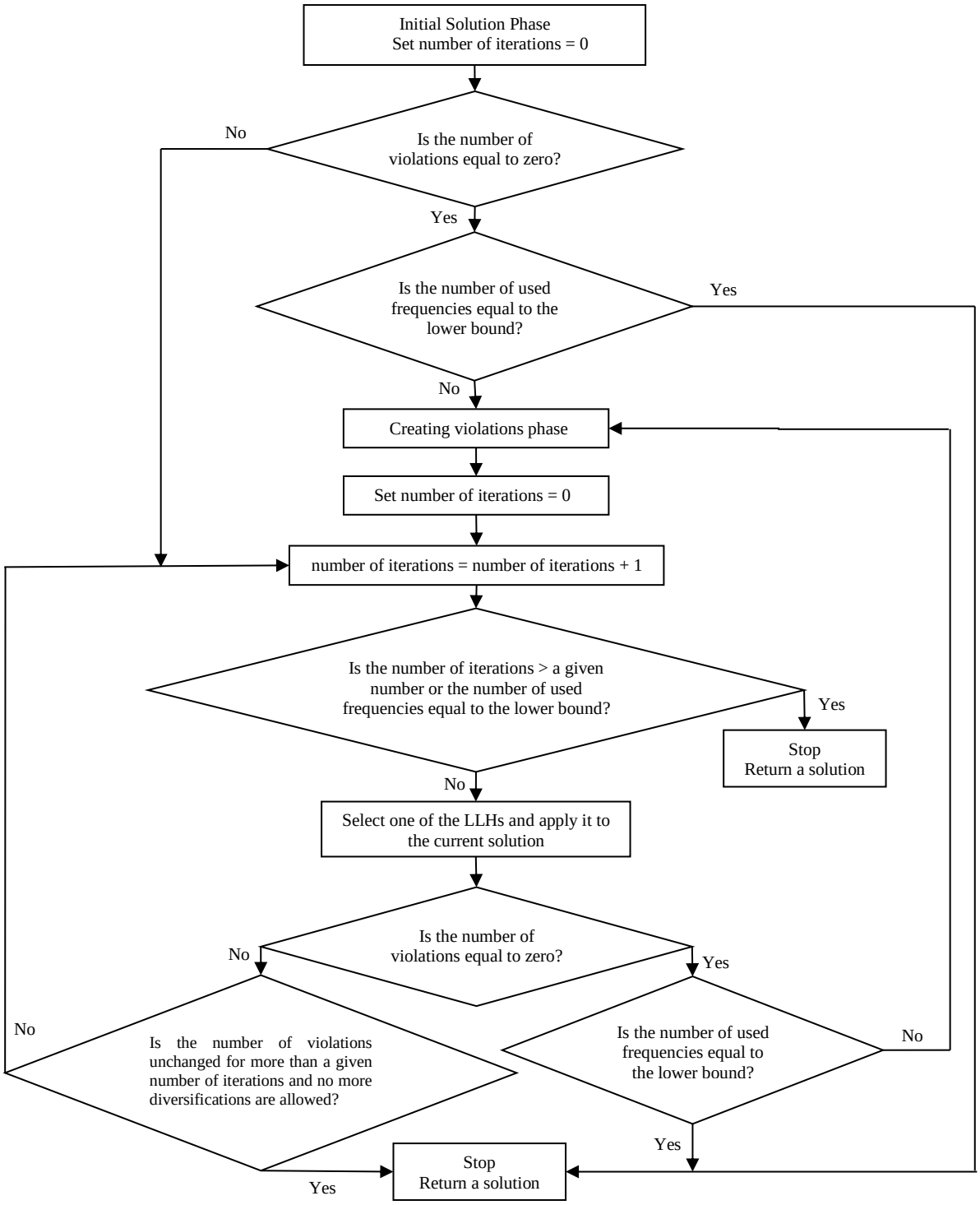

Figure 5.1: Overall structure of the HH algorithm for the static FAP.

5.3 Components of the Hyper Heuristic Algorithm

5.3.1 The Initial Solution Phase

In this phase, the initial solution is generated in the same way as for TS (see Section 3.4.4) to allow a fair comparison between HH and TS. That is, a greedy algorithm

hybridised with a descent method to produce a feasible initial solution, if possible. Similarly, a greedy constructive heuristic was used to generate an initial solution in the HH algorithm for the static FAP in [97].

5.3.2 The Creating Violations Phase

This phase aims to reduce the number of used frequencies in a feasible solution by removing a used frequency. The creating violations phase that we implement here is the same as that in TS (see Section 3.4.5). To the best of my knowledge, this idea has not been previously used in HH for the static FAP.

5.3.3 The Low Level Heuristics

HH involves a set of LLHs, each of which gives a new neighbour solution. In this study, 13 LLHs are applied, where some of them are simple (and previously used, see e.g. [97]) and the others are more advanced. Then, HH accepts or rejects each new solution. In fact, some neighbour solutions are accepted sometimes even if these lead to an increase in the number of violations in order to diversify the search. Hence, the LLHs can be divided into two groups: intensification and diversification LLHs. As this algorithm accepts neighbour solutions that are of the same cost as the current solution, cycling is one of the problems which may be faced in each LLH. In order to avoid this, each LLH has an independent local tabu list.

Each LLH starts by either choosing a frequency f_k to be removed or choosing a request r_i to be re-assigned. In the former type of the LLHs, the chosen frequency f_k should satisfy the following conditions: (i) f_k is not in the local tabu list, (ii) f_k is involved in most violations. If more than one frequency satisfies these conditions, then one of them is chosen randomly.

In the latter type of the LLHs, the chosen request r_i should satisfy the following conditions: (i) r_i is not in the local tabu list, (ii) r_i is involved in most violations. If more than one request satisfies those conditions, then one of them is chosen randomly.

A frequency f_j to be assigned in place of the removed frequency f_k or to be assigned to the chosen request r_i should satisfy the following conditions: (i) f_j is not in the local tabu list, (ii) f_j results in the minimum number of violations. If more than one frequency satisfies these conditions, then one of them is chosen randomly.

The descriptions of the LLHs which start by choosing a frequency f_k are given as follows:

LLH_1: the set of requests that are currently assigned to the chosen frequency f_k is swapped with its partner (based on the bidirectional constants).

LLH_2: all requests assigned to f_k are re-assigned to either the chosen unused frequency f_j or one of the used frequencies. If the assignment to f_j results in zero violations, then this is always made, otherwise each request is assigned to a used frequency that results in the smallest number of violations. In the event of a tie, the requests are assigned randomly.

LLH_3: a request is randomly selected from the set of requests that are currently assigned to the chosen frequency f_k to be re-assigned to the used frequency f_j.

The description of the LLHs which start by choosing a request r_i is given as follows:

LLH_4: the selected request r_i is assigned to the chosen used frequency f_j.

LLH_5: the selected request r_i is swapped with its partner (based on the bidirectional constraints).

LLH_6: it is similar to LLH_2 but the method of choosing the frequency to be removed is different. Here, the frequency of the selected request r_i is chosen to be removed.

The following LLHs are not required to satisfy the condition (ii) for selecting f_k, or r_i, or f_j.

LLH_7: it is similar to LLH_2 but the method of choosing the frequency to be removed is different. Here, the frequency f_k which is assigned to the fewest requests is removed.

LLH_8: a request r_i is chosen randomly to be re-assigned to a used frequency f_j which is also chosen randomly.

LLH_9: a request r_i is chosen randomly to be swapped with its partner (based on the bidirectional constraints).

LLH_{10}: a used frequency f_k is chosen randomly, and then one of the requests that is assigned to f_k, say r_i, is randomly selected. After that, a used frequency f_j which results in the minimum number of violations is assigned to r_i. In case of a tie, one of them is chosen randomly.

LLH_{11}: each request is swapped with its partner (based on the bidirectional constraints) as long as this does not increase the number of violations.

LLH_{12}: for each request r_i, the used frequency f_j that results in the minimum number of violations is chosen. In case of a tie, one of them is chosen randomly.

LLH_{13}: the search is diversified by assigning an unused frequency f_j in place of a used frequency f_k. In other words, the frequency f_j is assigned to all requests that are currently assigned f_k. Here, all the possible choices of f_j and f_k are considered. Then, the choice that results in the lowest number of violations is chosen.

Recall that LLHs can be divided into two groups: intensification and diversification LLHs. Here, the former group contains LLH_1, LLH_3, LLH_4, LLH_5, LLH_8, LLH_9, LLH_{10}, LLH_{11} and LLH_{12}, and the latter group contains LLH_2, LLH_6, LLH_7 and LLH_{13}.

Any change made to the solution by each LLH is added to the local tabu list. All the local tabu lists are cleared when a feasible solution is achieved, i.e. the sub-problem is solved.

5.3.4 LLH Selection Mechanisms

The LLH selection mechanisms can be executed in two ways: a non-adaptive selection method such as random selection or an adaptive selection method such as probabilistic selection [22]. In our HH algorithm, both of these selection mechanisms are considered and compared to select the best one. These two selection mechanisms are discussed in more detail in the following two subsections.

5.3.4.1 Random Selection of the LLHs

The first selection mechanism of the LLHs in this study is based on randomness. Random selection of the LLHs is the oldest, the simplest and the most commonly

used selection mechanism of the LLHs in HH [27]. This type of selection mechanism was previously used in HH for the static FAP in [97].

5.3.4.2 Probabilistic Selection of the LLHs

In this type of selection mechanism, the LLHs are selected probabilistically based on the performance of each LLH. At the beginning, each of the intensification LLHs has an equal chance of being selected. Note that the diversification LLHs are not probabilistically selected because they are intended to take the search to different parts of the solution space, regardless of the effect on a solution cost. Moreover, the diversification LLHs usually lead to worse results, which would result in a lower probability of selecting them, where this is undesirable. Probabilistic selection was used previously in HH for the static FAP in [135].

In order to reflect the performance of each LLH, the probabilities are updated by measuring their contribution in terms of reducing the number of violations. When a LLH reduces the number of violations, the probability of selecting that LLH increases. Three different approaches can be considered to update the probability when a LLH leads to a decrease or an increase in the number of violations. These three approaches are defined as follows (where N is a parameter, which is set to 50 in this study):

Approach 1:

If the selected LLH_j decreases the number of violations, the probability of selecting LLH_j is increased using Formula 5.1.

$$P(LLH_j) \leftarrow P(LLH_j) + \frac{1}{N} \tag{5.1}$$

In contrast, if LLH_j increases the number of violations, the probability of selecting LLH_j is decreased using Formula 5.2.

$$P(LLH_j) \leftarrow P(LLH_j) - \frac{1}{N} \tag{5.2}$$

Approach 2:

If the selected LLH_j decreases the number of violations by M, the probability of LLH_j is increased using Formula 5.3.

$$P(LLH_j) \leftarrow \frac{P(LLH_j) + M}{N} \tag{5.3}$$

Therefore, the probability is increased by a greater amount if LLH_j causes a large improvement in the number of violations.

In contrast, if LLH_j increases the number of violations, the probability of LLH_j is unchanged using Formula 5.4.

$$P(LLH_j) \leftarrow P(LLH_j) \tag{5.4}$$

Approach 3:

This approach is a mixture of the first two approaches and rewards changes that improve a solution and penalises changes that worsen a solution. If the selected LLH_j decreases the number of violations by *M*, then the probability of LLH_j is increased according to Formula 5.3. In contrast, if LLH_j increases the number of violations, the probability of LLH_j is decreased according to Formula 5.2.

After increasing or decreasing the probability of the selected LLH using one of the above approaches, the probabilities are normalised using Formula 5.5.

$$P(LLH_j) \leftarrow \frac{P(LLH_j)}{\sum_{\substack{i=1 \\ i \neq 2,6,7}}^{12} P(LLH_i)} \tag{5.5}$$

Example 5.1 clarifies the concept of the probabilistic selection of the LLHs.

Example 5.1:

> Assume the probabilistic selection of the LLHs is based on Approach 3. In the first iteration, LLH_1 is selected and reduces the number of violations by 6. In the second iteration, LLH_4 is selected and reduces the number of violations by 20. The probability of selecting of each LLH is shown in Table 5.1.

LLHs	Initial probability	M	P(LLH)	First updated probability	M	P(LLH)	Second updated probability
LLH 1	0.1111	6	0.12222	0.12088	-	0.12088	0.0935
LLH 2	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 3	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 4	0.1111	-	0.11111	0.10989	20	0.40219	0.31122
LLH 5	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 6	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 7	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 8	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
LLH 9	0.1111	-	0.11111	0.10989	-	0.10989	0.08503
Total	1	6	1.011102	1	20	1.29230	1

Table 5.1: An example of updating the probability of selecting each LLH.

Table 5.1 shows that the probability of selecting LLH_1 is increased from 0.1111 to 0.12088, reflecting its success and making it more likely to be selected in the future. In the second iteration, the probability of selecting LLH_4 is increased by 0.20133, which reflects its success. It can be seen that a bigger reduction in the number of violations results in a bigger increase in the corresponding probability.

Limitation on probabilities

Probabilistic selection of the LLHs can be implemented in two different ways:

1. Probabilistic selection of the LLHs without a limit.
2. Probabilistic selection of the LLHs with a limit.

In the first type of implementation, the probabilities have freedom to be increased or decreased without a limit. This may lead to some LLHs being ignored because their probabilities reach approximately zero. In order to avoid this, the second type of implementation restricts the probabilities by giving a minimum limit of the probability of selecting each LLH. The reason behind using this limit is to make sure there is a balance between selecting all the LLHs.

In order to avoid probabilities dropping below the limit value, which is set to 0.05 in this study, the probability of LLH_j is updated using Formula 5.6. This may lead to an increase in the probability in some LLHs.

$$P(LLH_j) \leftarrow Max\ (0.05\ , P(LLH_j)) \tag{5.6}$$

Using a limit for the probability of selecting each LLH was previously used in [29], but this technique is implemented differently in our HH algorithm by considering two stages of removing the extra probability, which is given by Formula 5.7.

$$Extra\ probability = 0.05\ -\ P(LLH_j) \tag{5.7}$$

Note that the second stage is used when the first stage fails to remove the extra probability. These two stages are described as follows:

a) Removing the extra probability by equivalent division

The extra probability is equally removed from those LLHs which do not reach the limit of the probability value. Assume that we have *n* LLHs that do not reach the limit of the probability. Then, the extra probability is divided by *n*, which gives the reduction probability as given by Formula 5.8.

$$Reduction\ probability = \frac{Extra\ probability}{n} \tag{5.8}$$

Then, the reduction probability is subtracted from each probability of the *n* LLHs. In order to clarify this, consider Example 5.2.

Example 5.2:

Assume there are five LLHs and their probabilities are given in Table 5.2.

LLHs	P(LLH)
LLH 1	0.40
LLH 2	0.30
LLH 3	0.20
LLH 4	0.05
LLH 5	0.05
Total	1

Table 5.2: An example of probabilities of selecting the LLHs.

Then, assume the probability of LLH_1 is increased by 0.05. After that, the probability is updated using Formula 5.5 and the limit of the probability is applied using Formula 5.6 as shown in Table 5.3.

LLHs	P(LLH)	Updated probability	Applying the limitation
LLH 1	0.45	0.428571	0.428571
LLH 2	0.30	0.285714	0.285714
LLH 3	0.20	0.190476	0.190476
LLH 4	0.05	0.047619	0.050000
LLH 5	0.05	0.047619	0.050000
Total	1.05	1	1.004762

Table 5.3: Applying the limit on the probabilities of selecting the LLHs.

Hence, the extra probability 0.004762 is removed from the other three LLHs which do not reach the limit of the probability. Using equivalent division according to Formula 5.8, each of these is reduced by $\frac{0.004762}{3}$ = 0.00159 as shown in Table 5.4.

LLHs	P(LLH)	Appling the equivalent division
LLH 1	0.428571	0.426984
LLH 2	0.285714	0.284127
LLH 3	0.190476	0.188889
LLH 4	0.050000	0.050000
LLH 5	0.050000	0.050000
Total	1.004762	1

Table 5.4: Applying the equivalent division to the probabilities of selecting the LLHs.

b) Removing the extra probability by proportional division

Occasionally, the previous stage resulted in at least one LLH with probability still being less than the limit. This happens when the probability of a particular LLH is narrowly above 0.05, but becomes smaller than 0.05 by the equivalent division process. In this stage, the extra probability is removed proportionally from the probabilities of LLHs which do not reach the limit. Say there are *m* such LLHs. Therefore, the probability of selecting each of the *m* LLHs is reduced by a different value based on proportional division. The reduction probabilities are given by Formula 5.9.

$$Reduction\ probability(LLH_j) = \frac{P(LLH_j)}{\sum_{i=1}^{m} P(LLH_i)} \times Extra\ probability \qquad (5.9)$$

In order to clarify this concept, consider Example 5.3.

Example 5.3:

Assume removing the extra probability by equivalent division results in an extra probability as shown in Table 5.5. Hence, the extra probability is removed by proportional division. As a result, each probability of LLHs which is higher

than the limit is reduced by different values based on proportional division using Formula 5.9. These results are shown in Table 5.5.

LLHs	P(LLH)	Updated probability	Applying the limitation	Reduction probability	Updated probability
LLH $_1$	0.450000	0.428571	0.428571	0.002255	0.426316
LLH $_2$	0.300000	0.285714	0.285714	0.001504	0.284210
LLH $_3$	0.200000	0.190476	0.190476	0.001003	0.189473
LLH $_4$	0.050000	0.047619	0.050000	0	0.050000
LLH $_5$	0.050000	0.047619	0.050000	0	0.050000
Total	1.050000	1	1.004762	0.004762	1

Table 5.5: Applying the proportional division to the probabilities of selecting the LLHs.

The extra probability in Table 5.5 is removed using proportional division based on Formula 5.9 which is calculated as follows:

- *Reduction probability* (LLH_1) = $\frac{0.428571}{0.904761} \times 0.004762 = 0.002255$
- *Reduction probability* (LLH_2) = $\frac{0.285714}{0.904761} \times 0.004762 = 0.001504$
- *Reduction probability* (LLH_3) = $\frac{0.190476}{0.904761} \times 0.004762 = 0.001003$

Finally, each different reduction probability is removed from each corresponding probability of the LLHs as shown in Table 5.5.

5.3.5 Acceptance Criteria

A combination of two types of acceptance criteria is applied. This concept was previously used in the literature (see e.g. [129]). The first one is applied when one of the intensification LLHs is selected, where only neighbour solutions that are not worse than the current solution are accepted. This type of criteria is commonly used and one of the successful acceptance criteria in the literature [122]. The second types of acceptance criteria is applied when one of the diversification LLHs is selected, where neighbour solutions are accepted regardless of the effect on the solution cost, i.e. even if these lead to an increase in the number of violations. Note that one of the diversification LLHs is selected when no better neighbour solution has been found for a certain number of iterations using the intensification LLHs. This allows the search to diversify by moving into a new area of the search space. Note that each of the diversification LLHs is allowed to give a worse solution for no more than a given number of times.

5.3.6 Stopping Criteria

The HH algorithm has three stopping criteria as follows: (i) a feasible solution whose number of frequencies is equal to the lower bound is found (as this is the optimal solution), (ii) the number of iterations is equal to a given number without successfully solving the sub-problem (see Section 5.2.1), i.e. a feasible solution could not be achieved (note that the number of iterations is reset to zero each time the sub-problem is solved), and (iii) the number of violations remains unchanged for more than a given number of iterations and the number of times each diversification LLH is executed reaches a given number.

5.4 Experiments and Results

This section presents and compares the results of HH for the static FAP in three stages using CELAR and GRAPH datasets (available on the FAP website[1]). The first stage presents and compares the results of HH in this study. The second stage compares the performance of HH with other algorithms in the literature. Finally, the performance of HH is compared with the algorithms considered in this thesis. The parameters of our HH algorithm are set based on experimentations for solving the sub-problem as follows:

- The maximum number of iterations is 2,500.
- The tabu tenure of a local tabu list for each LLH is 5.
- When no better neighbour solution can be achieved using the intensification LLHs for 50 consecutively iterations, a solution is produced using one of the diversification LLHs. This will usually be worse than the current solution. Each of the diversification LLHs is allowed to give a worse solution for no more than 6 times.

In this study, the algorithm was coded using FORTRAN 95 and all experiments were conducted on a 3.0 GHz Intel Core I3-2120 Processor (2nd Generation) with 8GB RAM and a 1TB Hard Drive.

5.4.1 Results Comparison of the Hyper Heuristic Algorithm

This section compares the performance of HH for the static FAP using random and probabilistic selection mechanism for the LLHs. The optimal solutions of these da-

tasets are known (available on the FAP website[1]). Therefore, the results are compared with the known optimal solutions. Note that the initial solutions in this algorithm are the same as given in Section 3.5.1.1.

As HH is mainly designed for the MO-FAP, applying HH to solve this problem is considered first. For each instance, HH is run 5 times, where each run uses different random number streams. The selected number of runs is chosen based on the experiments. The results of HH include the best, the worst and the average solution, and the average run time. Notice that a bold number means that the optimal solution is achieved.

5.4.1.1 Random Selection of the LLHs

Three different versions of HH are examined and compared using different subsets of the LLHs to investigate the importance of each subset. The first version of HH considers all the LLHs (this version of HH is denoted by approach A). The second version of HH considers all the LLHs except the diversification LLHs (this version of HH is denoted by approach B). The third version of HH considers all the LLHs except the LLHs which do not contribute to reducing the number of violations in approach A (this version of HH is denoted by approach C). The results of these approaches are given and compared as follows:

Approach A: this approach considers the performance of HH including all the LLHs. The results of this approach are given in Table 5.6.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	**16**	24	19.2	16	18.03 min
CELAR 02	**14**	**14**	14.0	14	0.62 sec
CELAR 03	16	18	16.8	14	3.20 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	36	48	40.0	22	10.41 min
GRAPH 01	**18**	20	18.4	18	48.03 sec
GRAPH 02	**14**	16	14.8	14	3.00 min
GRAPH 08	20	20	20.0	18	13.20 min
GRAPH 09	20	24	22.0	18	19.21 min
GRAPH 14	10	12	10.8	8	15.61 min

Table 5.6: Results of HH for the MO-FAP using approach A.

[1] http://fap.zib.de/problems/CALMA/ (last accessed 25 February 2015).

Table 5.6 shows that the optimal solution is achieved for CELAR 02 and CELAR 04, and most of the runs for GRAPH 01 and GRAPH 02. In contrast, for some instances, HH achieved solutions that use considerably more frequencies than the optimal.

Three instances are selected to analyse the number of calls (the number of times each LLH is selected) and the number of executions (the number of times each LLH is selected and accepted) of the LLHs. The selected instances represent different instances with different number of requests and constraints. Figure 5.2 shows the total number of calls of the LLHs for the selected instances.

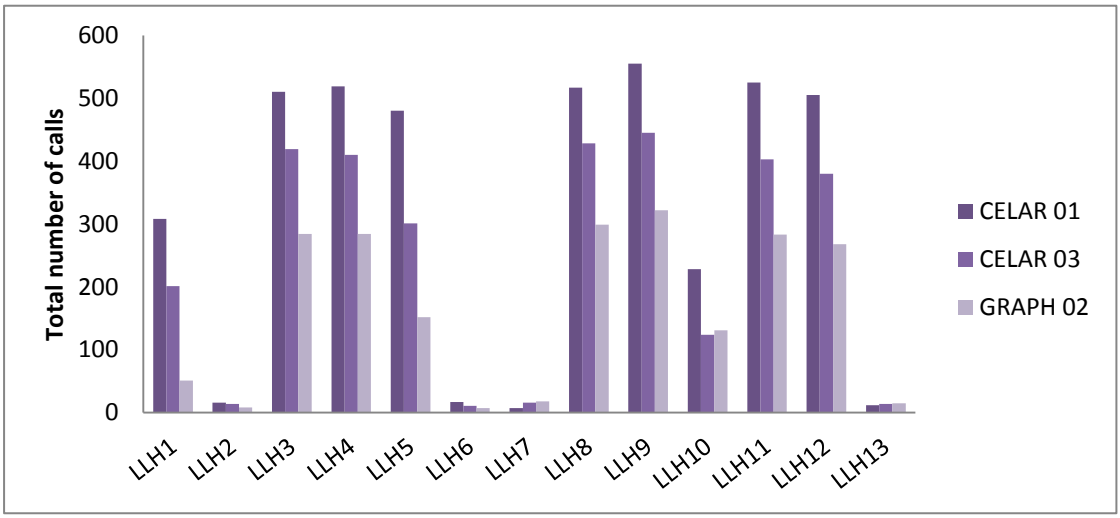

Figure 5.2: The total number of calls of the LLHs for the selected instances.

Figure 5.2 shows that the lowest numbers of calls of the LLHs correspond to LLH_2, LLH_6, LLH_7 and LLH_{13} (the diversification LLHs). The reason behind that is that these LLHs are applied to diversify the search and the use of these is limited. The total number of times each LLH is executed for the same selected instances is shown in Figure 5.3.

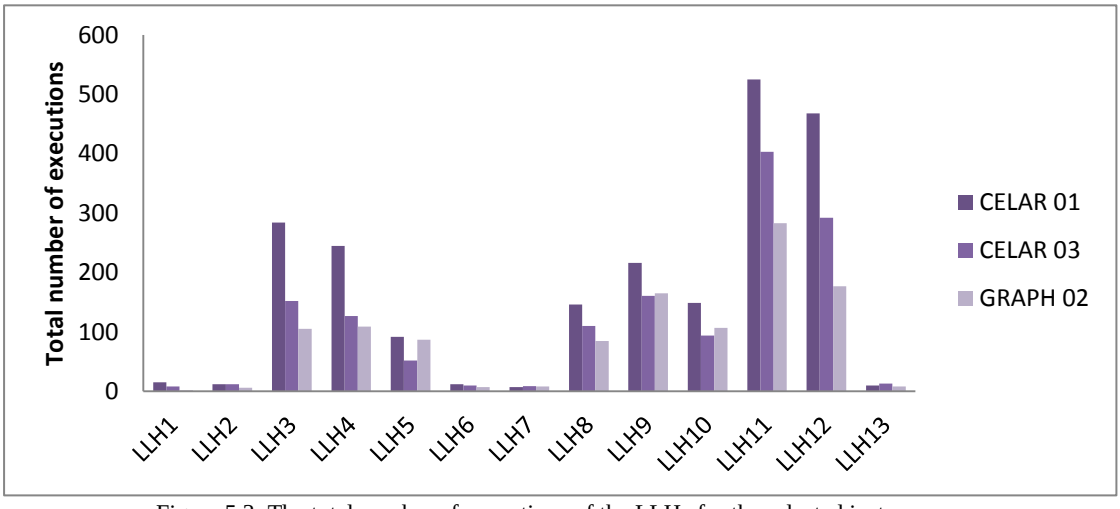

Figure 5.3: The total number of executions of the LLHs for the selected instances.

Figure 5.3 shows that the lowest numbers of executions of the LLHs correspond to the diversification LLHs (LLH_2, LLH_6, LLH_7 and LLH_{13}), as well as LLH_1. This is because LLH_1 involves the swapping of a set of pairs of requests, which is a small change in the solution compared with the other LLHs, so this is less likely to be accepted. However, the importance of LLH_1 is investigated by considering the total and average number of violations which has been reduced by each LLH. Figure 5.4 and Figure 5.5 show the total and the average reduction in the number of violations due to each LLH, respectively.

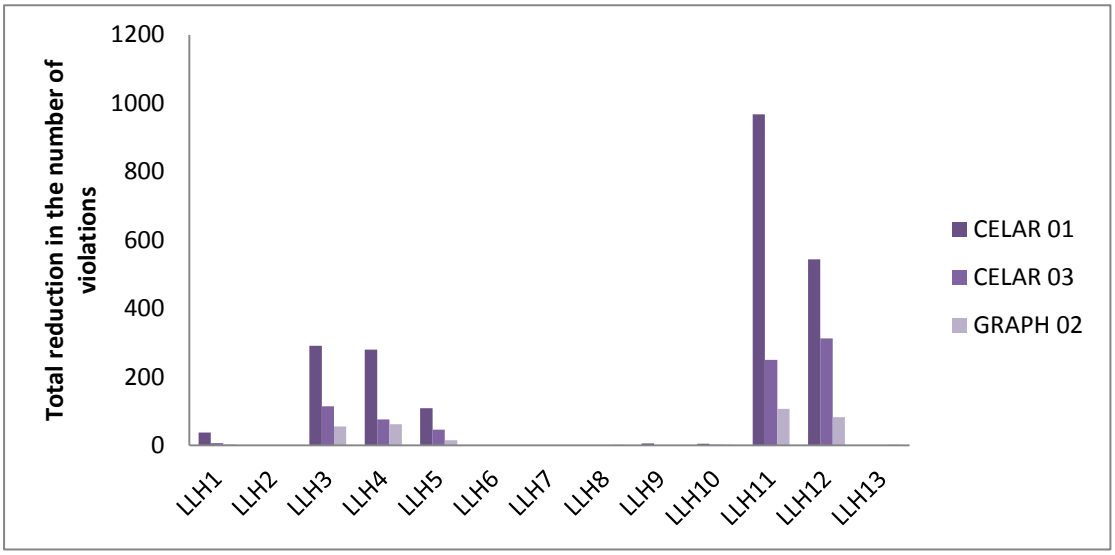

Figure 5.4: Total reduction in the number of violations due to each LLH.

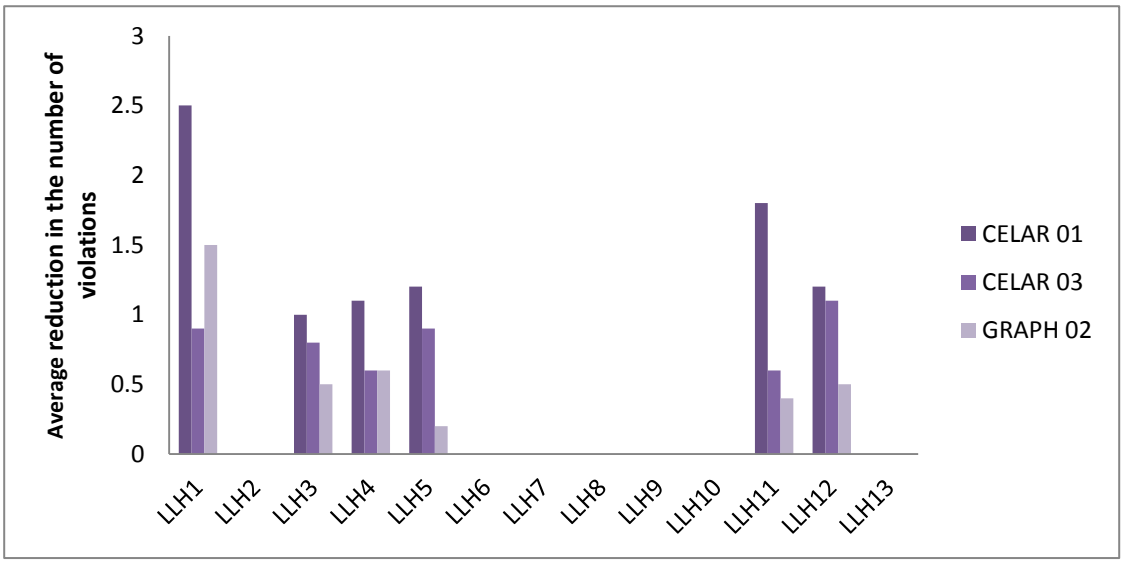

Figure 5.5: Average reduction in the number of violations due to each LLH.

Figure 5.4 and Figure 5.5 shows that LLH_2, LLH_6, LLH_7 and LLH_{13} did not contribute to reducing the number of violations as was explained earlier. Moreover, LLH_8,

LLH_9 and LLH_{10} did not contribute to reducing the number of violations either, but these LLHs are accepted as can be seen in Figure 5.3 because they keep the number of violations unchanged. To investigate whether these LLHs are important for optimizing the solution, approach B excludes the diversification LLHs and approach C excludes LLH_8, LLH_9 and LLH_{10} as shown next.

Approach B: this approach considers the performance of HH including all LLHs except the diversification LLHs (LLH_2, LLH_6, LLH_7 and LLH_{13}). The results of this approach are given in Table 5.7.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	20	24	21.6	16	8.63 min
CELAR 02	**14**	**14**	14.0	14	0.62 sec
CELAR 03	16	20	18.0	14	2.00 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	44	48	44.8	22	7.61 min
GRAPH 01	20	22	21.2	18	1.83 min
GRAPH 02	16	20	17.2	14	3.40 min
GRAPH 08	20	28	23.2	18	7.60 min
GRAPH 09	22	28	24.4	18	13.21 min
GRAPH 14	10	12	11.6	8	10.21 min

Table 5.7: Results of HH for the MO-FAP using approach B.

Table 5.7 shows that this approach struggled to find the optimal solutions in most of the instances, whereas in approach A the optimal solutions are found in almost half of the instances. Moreover, this approach struggled to improve the initial solution in some of the instances (see Table 3.6). Therefore, these results demonstrate the importance of the diversification LLHs to achieve better quality solutions.

Approach C: this approach excludes some LLHs which do not contribute to reducing the number of violations using approach A. In other words, HH is executed without LLH_8, LLH_9 and LLH_{10}. The results of this approach are shown in Table 5.8.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	22	20.4	16	18.03 min
CELAR 02	**14**	**14**	14.0	14	0.62 sec
CELAR 03	16	18	17.2	14	3.60 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	34	48	41.2	22	13.21 min
GRAPH 01	**18**	**18**	18.0	18	25.23 min
GRAPH 02	**14**	16	14.8	14	2.20 min
GRAPH 08	20	24	22.0	18	14.40 min
GRAPH 09	20	24	21.6	18	31.21 min
GRAPH 14	10	12	10.4	8	25.21 min

Table 5.8: Results of HH for the MO-FAP using approach C.

By comparing the average solutions of approach A and approach C, it can be seen that approach C is better than approach A for three instances, but worse for four instances. The run time is generally slower for approach C. In order to give a clear picture about the performance of each approach given earlier, the average solutions of the three different approaches are presented in Figure 5.6.

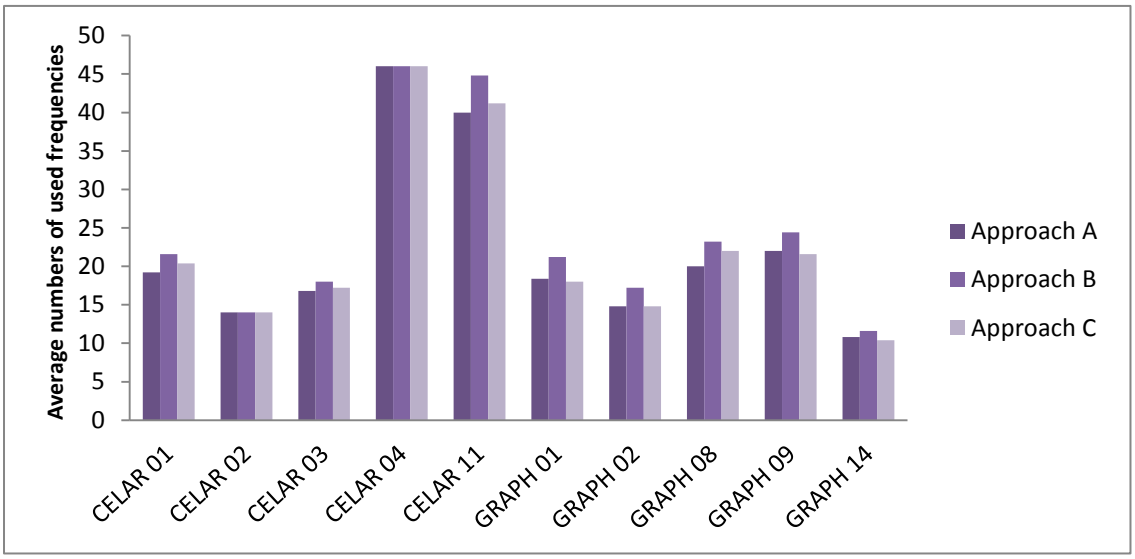

Figure 5.6: Average solutions of the three different approaches of the HH algorithm.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that approach A is significantly better than approach B. Furthermore, there is no significant difference between approach A and approach C. Hence, these two approaches are compared by considering the run time. Figure 5.7 presents a comparison of the run time between approach A and approach C.

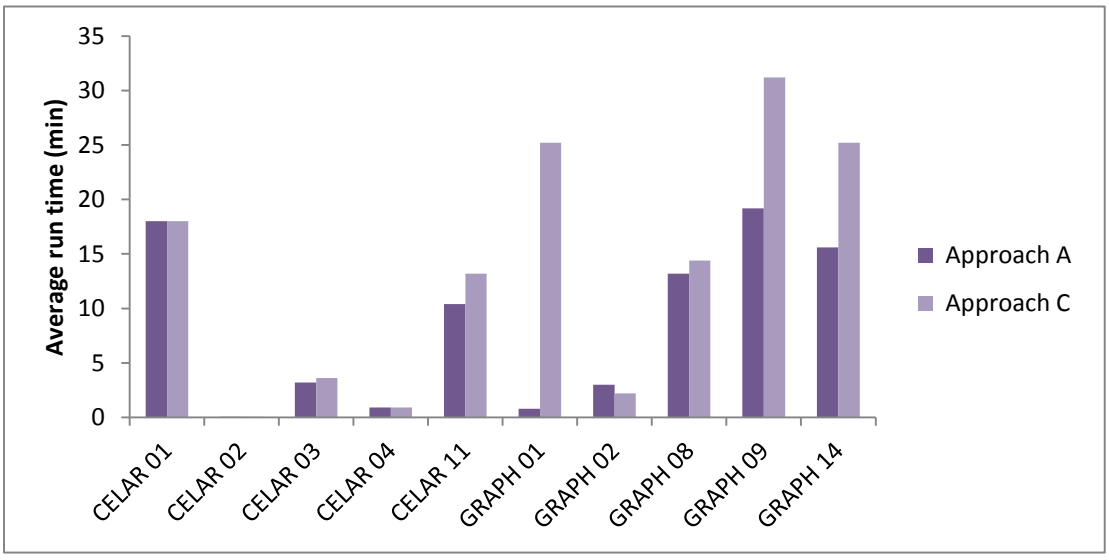

Figure 5.7: Average run time of approach A and approach C.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the average run time of approach A is significantly better than the average run time of approach C. Therefore, approach A is selected in this stage as the best approach.

5.4.1.2 Probabilistic Selection of the LLHs

In this section, the performance of the three different approaches, namely approach 1, 2 and 3 (see Section 5.3.4.2), are compared in two different ways: allowing the probabilities of the LLHs to vary without and with a limit. Adding a limit to the probabilities of the LLHs creates a balance which makes sure no LLH is ignored. Three instances, namely CELAR 01, CELAR 03 and GRAPH 08, are used to test these different approaches. The selected instances represent different instances with different numbers of requests and constraints. The following subsections discuss this comparison in detail.

5.4.1.2.1 Probabilistic Selection of the LLHs without a Limit

Here, the LLHs probabilities have freedom to be increased or decreased without a limit. The results of each approach are given as follows:

Approach 1:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	20	24	21.6	16	50.43 min
CELAR 03	16	18	17.2	14	9.60 min
GRAPH 08	20	28	23.2	18	3.70 min

Table 5.9: Results of approach 1 based on the probabilistic selection of the LLHs without a limit.

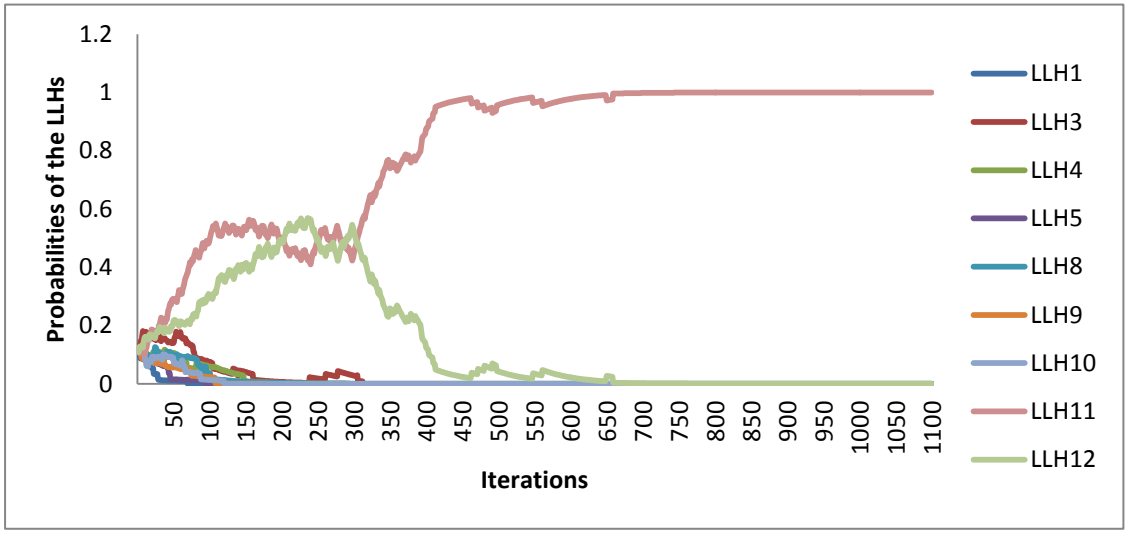

Figure 5.8: Probabilities of the LLHs in CELAR 01 during the iterations using approach 1 without a limit.

Approach 2:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	24	20.4	16	32.43 min
CELAR 03	16	18	16.8	14	9.40 min
GRAPH 08	20	22	20.4	18	37.20 min

Table 5.10: Results of Approach 2 based on the probabilistic selection of the LLHs without a limit.

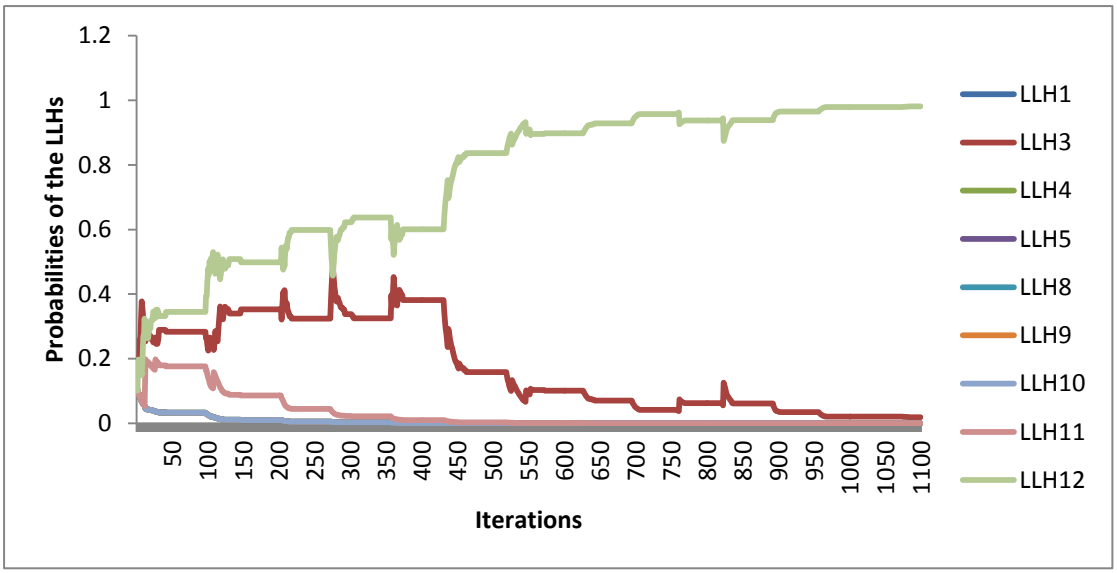

Figure 5.9: Probabilities of the LLHs in CELAR 01 during the iterations using approach 2 without a limit.

Approach 3:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	22	19.2	16	50.43 min
CELAR 03	**14**	18	16.4	14	9.40 min
GRAPH 08	**18**	22	20.4	18	37.20 min

Table 5.11: Results of approach 3 based on the probabilstic selection of the LLHs without a limit.

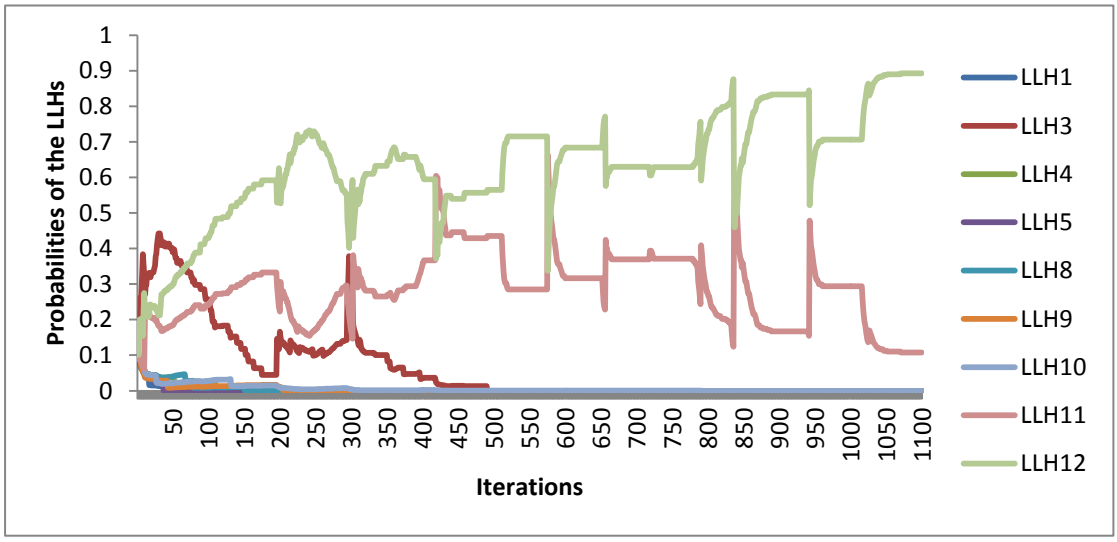

Figure 5.10: Probabilities of the LLHs in CELAR 01 during the iterations using approach 3 without a limit.

It can be seen from Figures 5.8, 5.9 and 5.10 that typically the probability of one LLH becomes close to 1. Interestingly, it is not the same LLH in each case. On the other hand, the probabilities of the majority of the LLHs reach 0. This means there is no chance of them being selected in the future. Hence, it is essential to add a limit for each LLH probability, which is discussed next.

5.4.1.2.2 Probabilistic Selection of the LLHs with a Limit

The probability of each LLH is restricted by a limit, that is, it is not permitted to decrease below 0.05. The results of these approaches are given as follows:

Approach 1:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	22	20.0	16	37.63 min
CELAR 03	16	18	17.2	14	7.20 min
GRAPH 08	**18**	22	19.6	18	26.40 min

Table 5.12: Results of approach 1 based on the probabilistic selection of the LLHs with a limit.

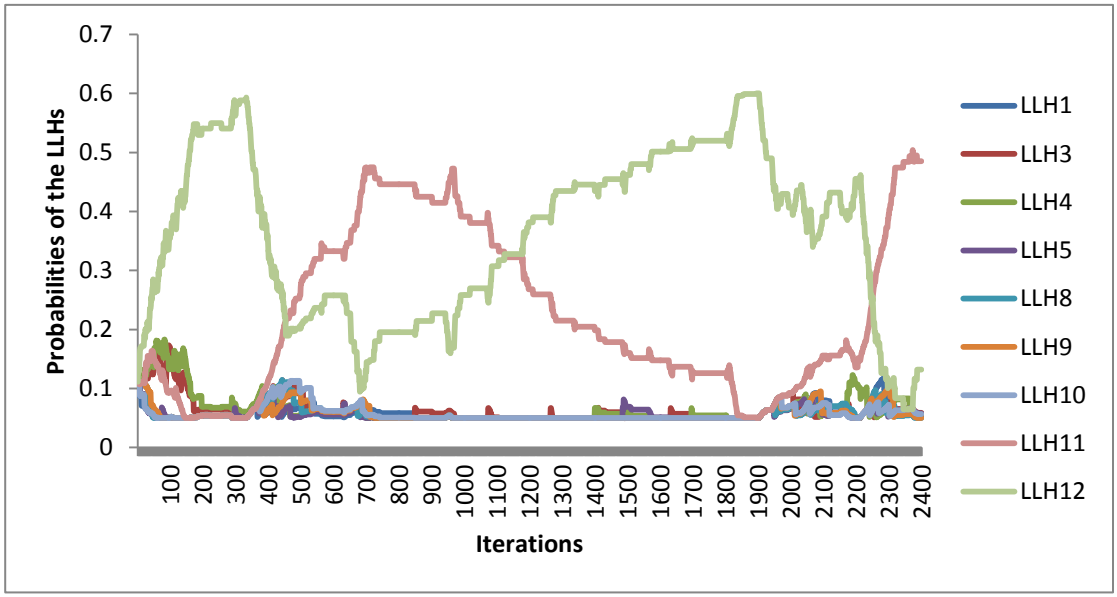

Figure 5.11: Probabilities of the LLHs in CELAR 01 during the iterations using approach 1 with a limit.

Approach 2:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	18	24	20.0	16	33.63 min
CELAR 03	16	18	16.4	14	9.40 min
GRAPH 08	**20**	22	21.2	18	25.20 min

Table 5.13: Results of approach 2 based on the probabilistic selection of the LLHs with a limit.

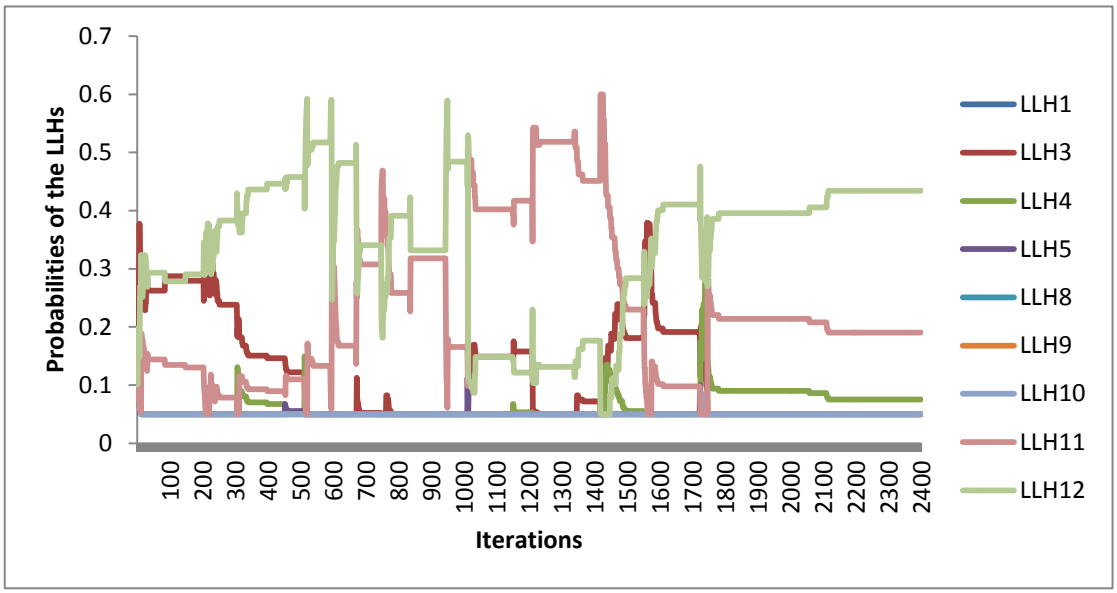

Figure 5.12: Probabilities of the LLHs in CELAR 01 during the iterations using approach 2 with a limit.

Approach 3:

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	**16**	22	18.8	16	26.03 min
CELAR 03	16	18	16.4	14	6.60 min
GRAPH 08	**18**	20	19.2	18	20.80 min

Table 5.14: Results of approach 3 based on the probabilistic selection of the LLHs with a limit.

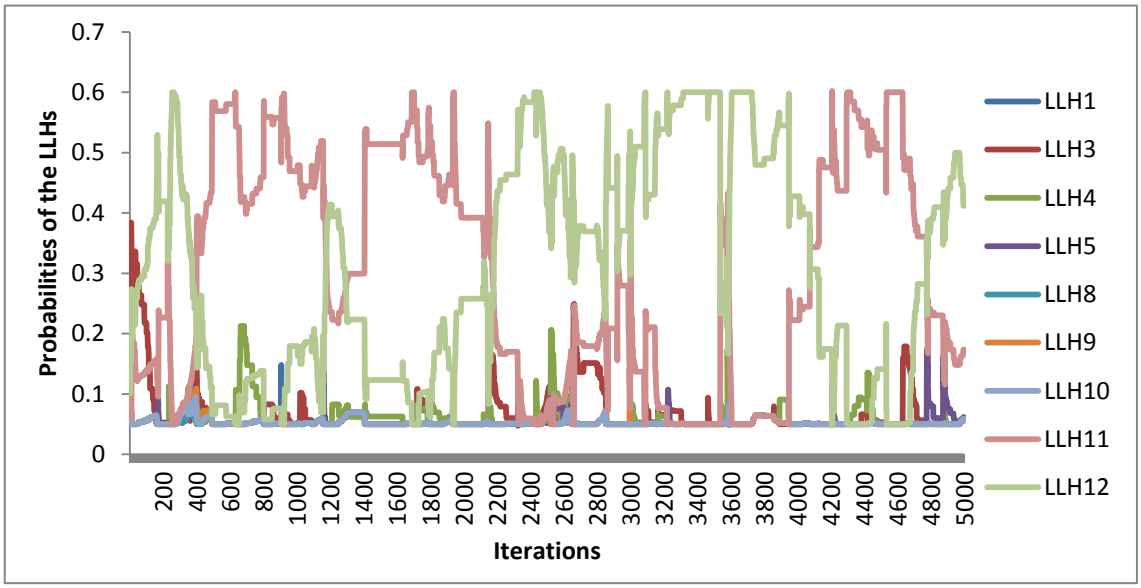

Figure 5.13: Probabilities of the LLHs in CELAR 01 during the iterations using approach 3 with a limit.

It can be seen from Figures 5.11, 5.12 and 5.13 that giving the probability of each LLH a limit is successful to ensure that all LLHs are involved in the search and probabilistically chosen. The LLHs that have high probabilities are the same LLHs as in Section 5.4.1.2.1, although here all LLHs have a reasonable chance of being selected.

5.4.1.2.3 Results Comparison and Analysis

This section compares the performance of HH which is based on the probabilistic selection without and with a limit (see Section 5.4.1.2.1 and Section 5.4.1.2.2, respectively). The total average numbers of used frequencies for all instances in each approach are shown in Figure 5.14.

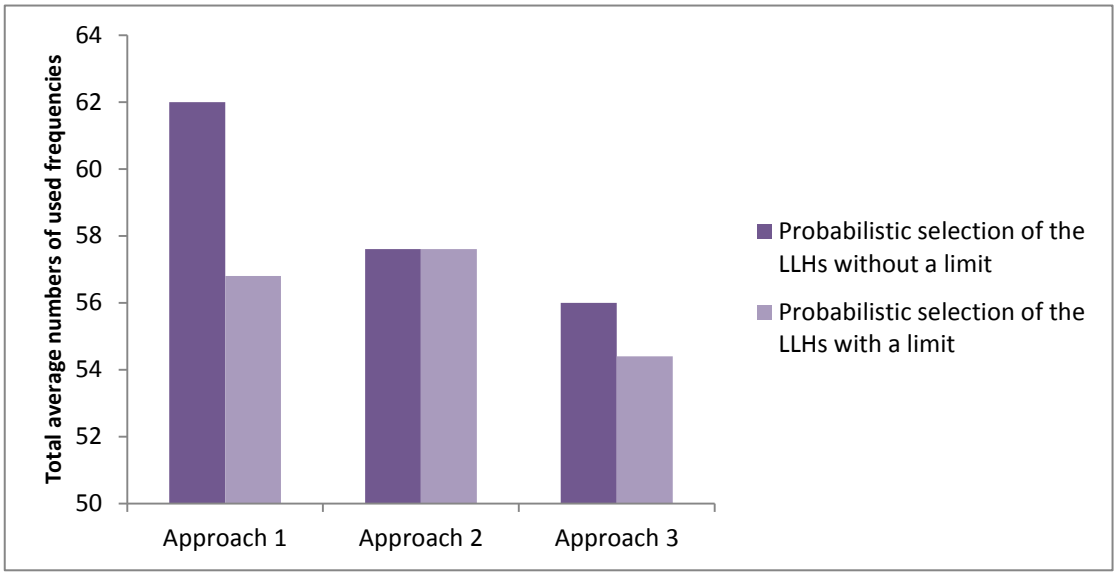

Figure 5.14: Total average number of used frequencies for each approach.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the performance of HH with a limit on the probabilities of LLHs is significantly better than HH without a limit. Hence, the three approaches (see Section 5.3.4.2) of updating the probabilities with a limit are compared as shown in Figure 5.15

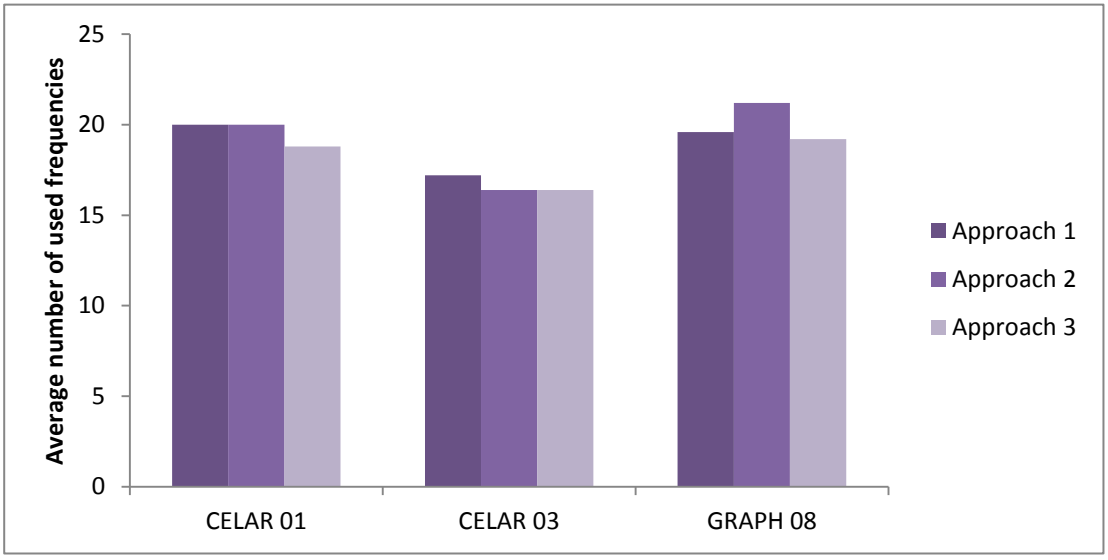

Figure 5.15: The average number of used frequencies in each instance for all approaches based on the probabilistic selection.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is no significant difference between the average results over the selected three instances of these approaches. Hence, these approaches are compared based on the average run time as shown in Figure 5.16.

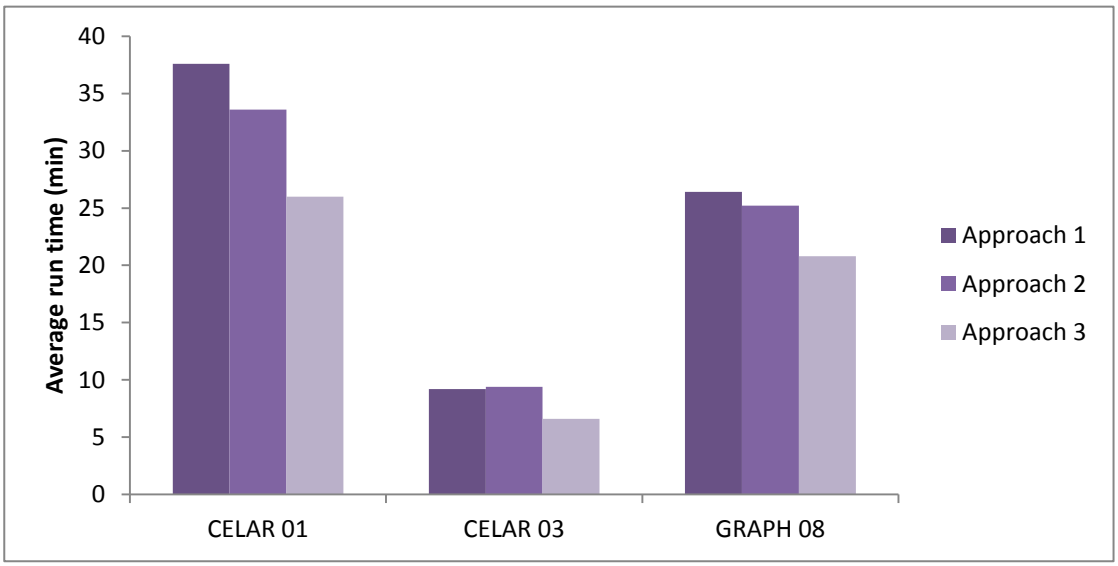

Figure 5.16: The average run time in each instance for all approaches based on the probabilistic selection.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is a significant difference between the average run times over the selected three instances of these approaches. Therefore, the selected approach in this stage is approach 3.

Finally, Table 5.15 presents the best results of the HH algorithm using the LLHs probability selection with a limit and updating the probability using approach 3.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	**16**	22	18.8	16	36.03 min
CELAR 02	**14**	**14**	14.0	14	0.82 sec
CELAR 03	16	18	16.4	14	7.60 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	38	48	43.2	22	19.21 min
GRAPH 01	**18**	**18**	18.0	18	1.13 min
GRAPH 02	**14**	18	14.8	14	4.60 min
GRAPH 08	**18**	20	19.2	18	28.80 min
GRAPH 09	22	24	22.4	18	38.41 min
GRAPH 14	10	12	10.8	8	38.4 1min

Table 5.15: The best results of the HH algorithm for the MO-FAP based on the probabilistic selection.

5.4.1.3 Comparison of the LLH Selection Mechanisms

The results of the HH algorithm based on the LLH selections mechanisms, which are random and probabilistic selection mechanism, are shown in Figure 5.17.

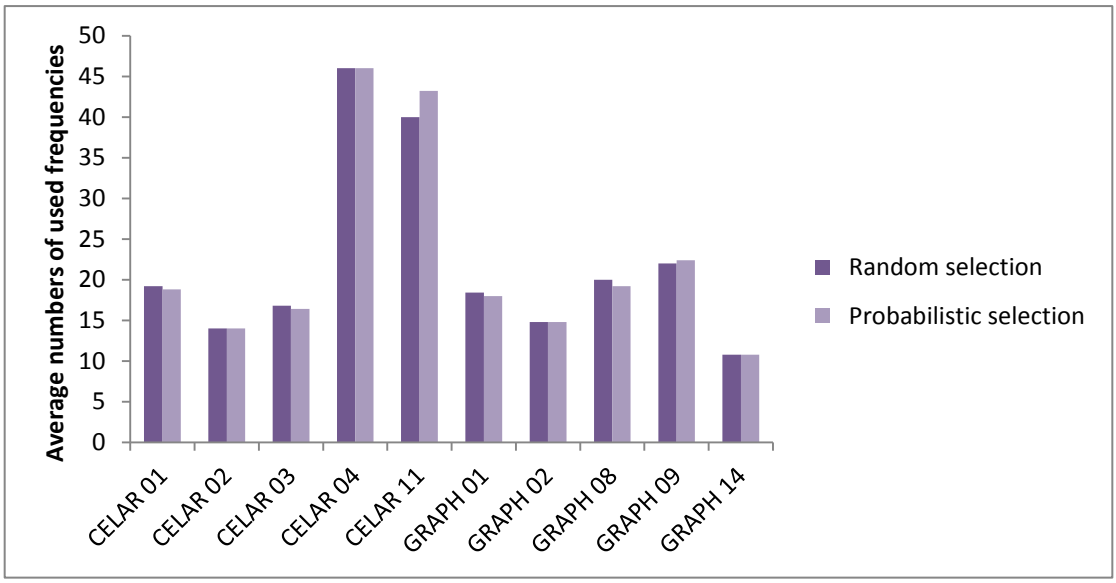

Figure 5.17: The average results of the HH algorithm using two types of the LLH selection mechanisms.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that there is no significant difference between the average results of HH based on the LLHs selection mechanisms. Therefore, these approaches are compared based on the run time as shown in Figure 5.18.

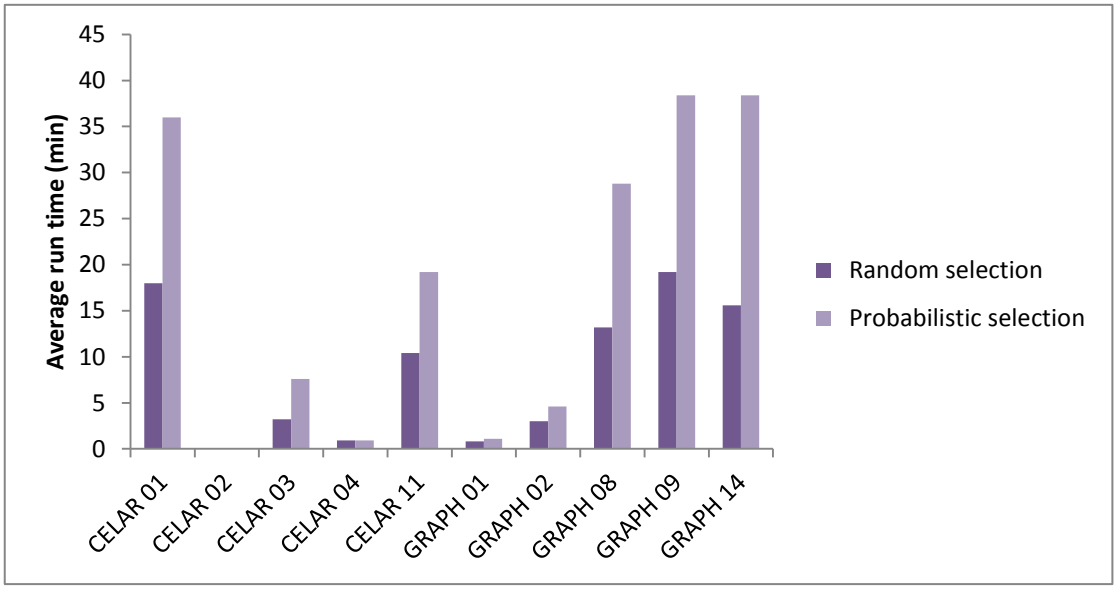

Figure 5.18: The average run time of the HH algorithm using two types of the LLH selection mechanisms.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the average run time using the random selection is significantly better than using the probabilistic selection of the LLHs. Hence, the random selection of the LLHs is selected.

The best results of the HH algorithm are given in Table 5.16. Note that the run time of finding the lower bound of the number of frequencies for each domain (see Table 3.3) is included.

Instance	Best solution	Worst solution	Average solution	Optimal solution	Average run time
CELAR 01	**16**	24	19.2	16	18.03 min
CELAR 02	**14**	**14**	14.0	14	0.62 sec
CELAR 03	16	18	16.8	14	3.20 min
CELAR 04	**46**	**46**	46.0	46	54.34 sec
CELAR 11	36	48	40.0	22	10.41 min
GRAPH 01	**18**	20	18.4	18	48.03sec
GRAPH 02	**14**	16	14.8	14	3.00 min
GRAPH 08	20	20	20.0	18	13.20 min
GRAPH 09	20	24	22.0	18	19.21 min
GRAPH 14	10	12	10.8	8	15.61 min

Table 5.16: The best results of the HH algorithm for the MO-FAP.

Table 5.16 shows that HH achieved the optimal solution for CELAR 01, CELAR 02, CELAR 04, GRAPH 01 and GRAPH 02. However, for some instances, it used considerably more frequencies than the optimal solution. Moreover, these results were achieved in a reasonable time, mostly less than 18 minutes.

Furthermore, it is of interest to investigate whether HH without significant changes can be successfully applied to other variants of the static FAP (MS-FAP and MI-FAP). Notice that our HH algorithm has been mainly designed to solve the MO-FAP. Therefore, a small number of changes are made to the HH algorithm to solve the MS-FAP. For example, in the creating violations phase, the removed frequency is changed to be the frequency that reduces the maximum value of the used frequencies. For the MI-FAP, the creating violations phase is omitted as minimizing the number of used frequencies is not required. It was found that HH showed poor performance for both MS-FAP and MI-FAP, which agrees with the findings for tabu search (TS) and ant colony optimization (ACO) (see Chapters 3 and 4, respectively). It is likely that more significant changes are required for HH to work well on other variants of the static FAP.

5.4.2 Results Comparison with Other Algorithms

This section compares the performance of our HH algorithm with other algorithms in the literature and the algorithms considered in this thesis, namely TS and ACO. The comparison is shown in Table 5.17, where a bold number means that the optimal solution was achieved and a dash "-" means that this result is not available.

Instance	GENET [16]	Genetic algorithm [94]	Potential reduction [151]	A nonlinear approach [150]	Evolutionary search [34]	Simulating annealing [145]	Variable depth search [145]	TS [15]	TS [145]	Our TS algorithm	Our ACO algorithm	Our HH algorithm	Optimal solution
CELAR 01	**16**	20	**16**	**16**	-	**16**	**16**	18	**16**	**16**	18	**16**	16
CELAR 02	**14**	**14**	**14**	-	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	14
CELAR 03	**14**	16	16	16	**14**	**14**	**14**	**14**	**14**	**14**	16	16	14
CELAR 04	**46**	**46**	**46**	-	-	**46**	**46**	**46**	**46**	**46**	**46**	**46**	46
CELAR 11	24	32	-	-	-	24	24	24	**22**	38	-	36	22
GRAPH 01	**18**	20	**18**	**18**	**18**	-	-	**18**	**18**	**18**	20	**18**	18
GRAPH 02	**14**	16	**14**	**14**	**14**	-	-	16	**14**	**14**	16	**14**	14
GRAPH 08	22	-	**18**	**18**	-	-	-	24	20	**18**	24	20	18
GRAPH 09	22	28	**18**	**18**	-	-	-	22	22	**18**	-	20	18
GRAPH 14	-	14	10	10	-	-	-	12	10	**8**	10	10	8

Table 5.17: Results of HH and other algorithms in the literature.

Table 5.17 shows that our HH algorithm achieved the optimal solution for half of the instances and came as the fifth best algorithm (based on the number of optimal solutions achieved by each algorithm). However, the majority of the optimal solutions were achieved by our TS algorithm, which is the best performing algorithm in Table 5.17 and in this thesis. In contrast, the genetic algorithm in [94] and our ACO algorithm showed poor performance compared with the other algorithms. In fact, the optimal solution was achieved for only two instances. Overall, our HH algorithm came as the second best algorithm in this study and shows reasonable results compared with other algorithms in the literature.

5.4.3 Results Comparison with TS and ACO Algorithms

This section compares the performance of the three heuristic algorithms considered in this study, which are TS, ACO and HH, in order to identify an appropriate solution method for the static FAP and to determine the appropriate heuristic algorithms to be used to construct an approach to solve the dynamic FAP, where the run time is important. Table 5.18 shows the results comparison of these heuristic algorithms including the best found solution, the average run time and the optimal solution. Note that a bold number means that the optimal solution was achieved and a dash "-" means that a feasible solution could not be found.

Instance	Best found			Average run time			Optimal solution
	TS	ACO	HH	TS	ACO	HH	
CELAR 01	**16**	18	**16**	3.6 min	1.4 hrs	18.0 min	16
CELAR 02	**14**	**14**	**14**	0.5 sec	11.2 min	0.6 sec	14
CELAR 03	**14**	16	16	1.0 min	31.1 min	3.2 min	14
CELAR 04	**46**	**46**	**46**	54.3 sec	55.8 min	54.3 sec	46
CELAR 11	38	-	36	8.8 min	-	10.4 min	22
GRAPH 01	**18**	20	**18**	5.4 sec	18.0 min	48.3 sec	18
GRAPH 02	**14**	16	**14**	2.2 sec	29.8 min	3.0 min	14
GRAPH 08	**18**	24	20	24.3 sec	30.1 min	13.2 min	18
GRAPH 09	**18**	-	20	3.0 min	-	19.2 min	18
GRAPH 14	**8**	10	10	4.8 min	59.8 min	15.6 min	8

Table 5.18: The best solutions and the average run time of TS, ACO and HH in this study.

Table 5.18 shows that the best performing algorithm in this study is TS while HH came as the second best algorithm. In contrast, ACO is the worst performing among these heuristic algorithms. Figure 5.19 compares the quality of the solution of these heuristic algorithms by showing the number of instances where the optimal solution is achieved and the number of instances where the optimal solution is not achieved.

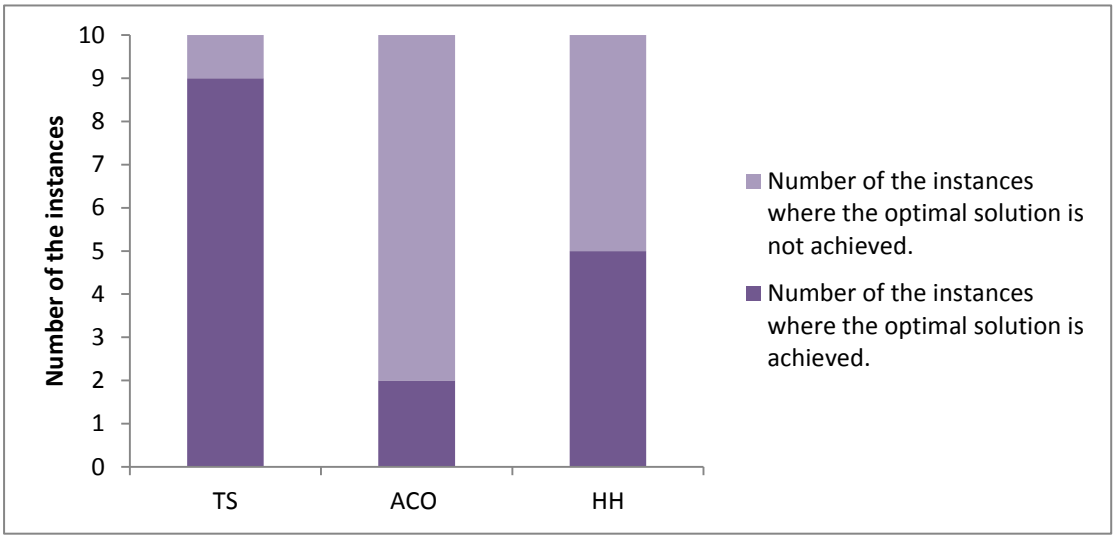

Figure 5.19: The numbers of instances where the optimal solution is achieved by TS, ACO and HH.

Figure 5.19 shows TS is the best performing algorithm for the static FAP, followed by HH and finally ACO. This suggests that local search-based algorithms perform better than population-based algorithms and HH for solving the static FAP.

The run time is considered as the best algorithm will be used to construct an approach to solve the dynamic FAP in the next chapter, where time is limited. Figure 5.20 shows the total of average run times for each heuristic algorithm.

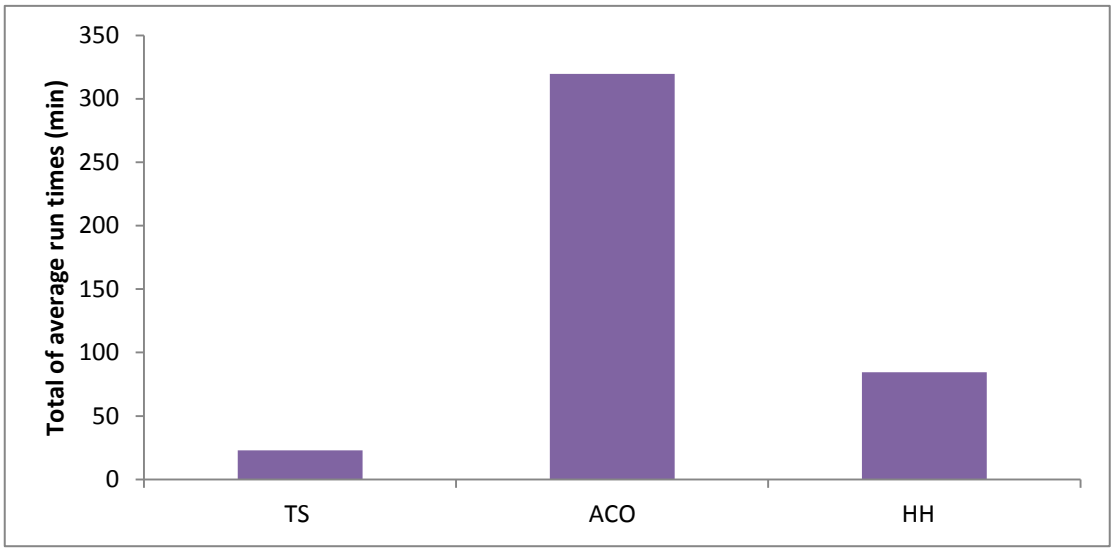

Figure 5.20: Total of average run times for TS, ACO and HH.

Figure 5.20 shows that the best heuristic algorithm in this study in terms of the run time is TS, followed by HH and the worst one is ACO. Overall, TS and HH have the best performance in this study. Hence, these heuristic algorithms are selected to be used in Chapter 6 to construct an approach to solve the dynamic FAP.

5.5 Time Complexity and Convergence of HH

The time complexity of the HH algorithm can be expressed using the big O notation by counting the number of times the key operation are performed, which is assigning a frequency to a request. In terms of the initial solution phase, the time complexity of the assignment stage is of order $O(NR^2 * NF)$, where NR is the number of requests and NF is the number of frequencies, the allowing infeasible assignment stage is of order $O(NR * NF)$ and the descent method stage is of order $O(NR * NF)$. In terms of the creating violations phase, the time complexity is of order $O(NR * NF)$. For each LLH, the initial cost is calculated, which is of order $O(NR^2)$. Then, the cost is updated each step, which requires a number of calculations proportional to $O(NR)$. The highest order of time complexity in the LLHs is LLH_{13}, which have complexity $O(NR^2 * NF^2)$. Hence, the overall time complexity of HH is of order $O(NR^2 * NF^2)$.

To investigate the convergence of this algorithm, first note that the number of used frequencies in our HH algorithm never increases. This is because the algorithm consists of reducing the number of used frequencies and seeking for a feasible solution with a fixed number of used frequencies. If a feasible solution is found (i.e. the sub-

problem (see Section 5.2.1) is solved), then the number of used frequencies is reduced and the number of iterations is reset to zero. This process is repeated until a feasible solution can no longer be found.

Here, HH is run on GRAPH 09 for more iterations for each sub-problem (say 20,000 iterations) and the stopping criteria (see Section 5.3.6) are ignored to investigate the convergence of this algorithm. Moreover, HH is executed for five runs, where each run uses different random number streams. Figure 5.21 shows the convergence of HH using the average solutions of the five runs.

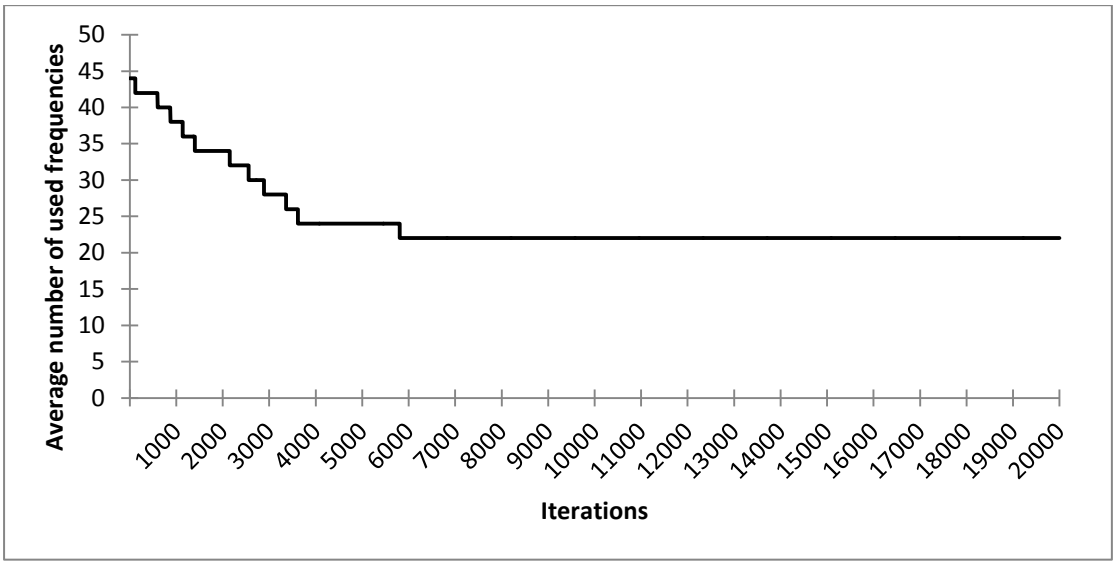

Figure 5.21: The convergence of the HH algorithm on the GRAPH 09 instance.

Figure 5.21 shows that HH converged within 2,500 iterations with a fixed number of used frequencies. Furthermore, similar convergence has been found for other instances. This convergence experiment justifies the selected number of iterations in this study.

5.6 Conclusions

This chapter discussed HH for the static FAP, where this algorithm is mainly designed to solve the MO-FAP. Note that this is the first attempt to solve the static FAP by HH using the datasets considered in this study (CELAR and GRAPH). Several novel and existing techniques have been used to attempt to improve the performance of this algorithm. One of these is applying the lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying

graph colouring model. These lower bounds are used to ensure that we never waste time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bounds. Moreover, our HH includes 13 simple and advanced LLHs, some of which are used for diversification. Furthermore, each LLH has an independent tabu list in order to avoid cycling. Additionally, two different methods of the LLHs selection were compared: random and probabilistic selection. The probabilistic selection gives a higher probability to the LLHs which reduce the number of violations. Moreover, two types of the LLH probabilistic selection were tested: without and with a limit.

It was found that random selection of the LLHs performed better than probabilistic selection. Moreover, the performance of HH does not seem to be as efficient as TS (see Chapter 3), but does perform much better than ACO (see Chapter 4). Hence, this algorithm recorded the second best performance in this thesis. As a result, this suggests that local search-based algorithms are more suitable for the static FAP than population-based algorithms and HH. Furthermore, applying HH without significant changes on other variants of the static FAP (MS-FAP and MI-FAP) were not successful. This finding agrees with what has been found for TS (see Chapter 3). It is likely that more significant changes are required for HH to work well on these problems.

Finally, the research questions which were presented in the beginning of this chapter can be answered as follows:

- *Can HH perform better than TS and ACO on the static FAP?*
  HH does not seem to be as efficient as TS, but does perform much better than ACO (see Section 5.4.3).

- *What is the best mechanism for selecting the LLHs?*
  Based on the investigation in this study, the best selection mechanism for the LLHs is based on random selection (see Section 5.4.1.3).

- *Is HH an appropriate solution method for the static FAP?*
  HH showed competitive performance compared with other algorithms in the literature (see Sections 5.4.2). Moreover, HH came as the second best performing heuristic algorithm in this study (see Section 5.4.3). Therefore, HH is considered as an appropriate solution method for the static FAP after TS.

Chapter 6

Approaches for Dynamic and Static FAPs

6.1 Introduction

There has been an increasing interest in variants of dynamic optimization problems, in which some attributes of the problem change over time. Decisions have to be made at different points of time, and the quality of the solution depends on all the decisions made over time periods. The major difficulties of dynamic problems come from ignorance of how the problem is going to change in the future. Many real-life problems can be considered to be dynamic, but up until recently, research has focused on static problems, where all the data is known in advance. Research into dynamic problems is growing in some areas such as graph colouring problems, scheduling problems and vehicle routing problems [54].

One of the dynamic problems is the dynamic frequency assignment problem (FAP), which was proposed in [55]. In the dynamic FAP, new requests become known over time periods and frequencies need to be assigned to them effectively and promptly while satisfying a set of constraints (see Section 1.4.1). Hence, solving the dynamic

FAP needs to deal with uncertain data as new data arrive in a dynamic process. There are two possible types of uncertain data: entirely accessible data and partially accessible data. The first type belongs to the area of robust optimization [102], where we need to find solution methods able to accommodate different realizations of data. The second type, which is considered in this study, corresponds to dynamic optimization [84], which has three features: 1) new decisions are made one by one; 2) the decisions are non-adjustable unless necessary; 3) no information about the future is accessible.

During solving the dynamic FAP using uncertain data, if no feasible solution can be found, it is essential to change the previous decisions to improve the solution with the minimum number of changes. Although changing frequencies that have been assigned previously is technically allowed, in practice this can be time consuming and takes up human resources. Hence, the dynamic FAP states that changing frequencies of requests that are previously assigned should be avoided unless no other means of finding a feasible solution exists. Therefore, the objective of the dynamic FAP is to find a feasible solution with the minimum number of re-assigned requests.

In order to clarify the dynamic FAP, a dynamic FAP instance over 3 time periods is illustrated in Figure 6.1, where each node represents a request, each edge represents a bidirectional or an interference constraint (see Equations 1.1 and 1.2, respectively) and each colour represents a time period in which a request becomes known for the first time.

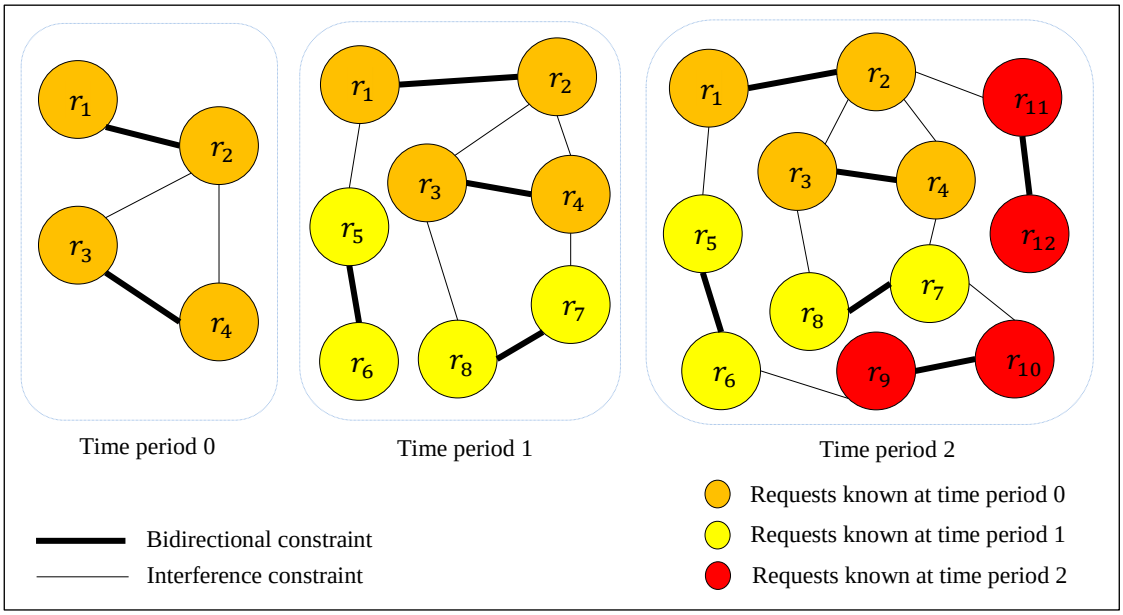

Figure 6.1: A dynamic FAP instance over 3 time periods.

Figure 6.1 shows that requests are partitioned into three sub-problems based on 3 time periods, where each sub-problem is considered in its respective time period.

In this chapter, various approaches are designed to solve the dynamic FAP. The best performing heuristic algorithms for the static FAP in this thesis (tabu search (TS) and hyper heuristic (HH)) are used to construct these approaches for the dynamic FAP. It is interesting to investigate whether heuristic algorithms which work well on static FAP also prove efficient on the dynamic FAP. Several novel and existing techniques are applied to improve the performance of these approaches. These include a novel technique called the *Gap* technique, which aims to identify a good frequency to be assigned to a given request. To assess these approaches, new dynamic FAP datasets are generated from the static FAP datasets (CELAR and GRAPH). Moreover, the new dynamic FAP datasets have been made available for other researchers, which can be found on the dynamic FAP website[1].

Furthermore, this chapter proposes a novel approach to solve the minimum order FAP (MO-FAP), which is a variant of the static FAP. The objective of the MO-FAP is to find the optimal solution with no restriction on the number of re-assigned requests. The idea of this approach is inherited from the dynamic concept, where the computational time is important. It can be extremely time consuming to solve a large static problem and it may be a better strategy to break it into smaller parts and solve these in turn, rather than try to solve it in its entirety. Hence, the static FAP is modelled as a dynamic FAP through dividing this problem into smaller sub-problems, which are then solved consecutively. This novel approach is constructed using TS (see Chapter 3) because it was the best performing heuristic algorithm in this thesis. Several techniques are applied to make this approach more efficient. These include applying the lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model (see Section 3.2), and the *Gap* technique.

In this chapter, we focus on the following research questions:

- *Can TS and HH for the static FAP be successful on the dynamic FAP?*
- *Can the proposed approach that models the static FAP as a dynamic FAP be an effective method for the static FAP?*

[1] https://dynamicfap.wordpress.com/

This chapter is organised as follows: Section 6.2 describes how to generate the new dynamic FAP datasets. In Section 6.3, an overview of approaches for the dynamic FAP is presented. Section 6.4 presents the main components of approaches for the dynamic FAP. In Section 6.5, the results of these approaches are given and compared. In Section 6.6, a novel approach for solving the static FAP that models it as a dynamic FAP is proposed and discussed. Finally, this chapter is closed with some conclusions.

6.2 Generating the Dynamic FAP Datasets

For the purpose of the present study, new dynamic FAP datasets are generated from the static FAP datasets (CELAR and GRAPH) to assess the proposed approaches. The dynamic FAP instances are generated by breaking down the static FAP instances into smaller sub-problems, where each sub-problem is considered at a specific time period. To achieve this, each request is given an integer number between 0 and n (where n is a positive integer) indicating the time period in which it becomes known. In effect, the problem is divided into $n+1$ smaller sub-problems P_0, $P_1, \ldots, P_n$, where n is the number of sub-problems after the initial sub-problem P_0. Each sub-problem P_i contains a subset of requests which become known at time period i. In this study, we found that the number of sub-problems does not impact on the performance of the approaches for solving the dynamic FAP, so the number of sub-problems is fixed at 21 (i.e. n = 20).

Based on the number of the requests known at time period 0 (belonging to the initial sub-problem P_0), 10 different dynamic FAP instances are generated from each static FAP instance. These dynamic FAP instances are named using percentages which indicate the number of requests known at time period 0, namely 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% (note that 100% means all the requests are known at time period 0 and so corresponds to the static FAP instance).

6.3 Overview of the Approaches for the Dynamic FAP

In this section, the solution space and the cost function of the approaches for the dynamic FAP are given. Moreover, this section presents an overview of the structure of these approaches.

6.3.1 Solution Space and Cost function

As the objective of the dynamic FAP is to find a feasible solution with the minimum number of re-assigned requests, the solution space is defined as the set of all possible solutions that satisfies all the constraints (see Section 1.4.1) and the cost function is defined as the number of re-assigned requests. This configuration was previously used in [55] for the dynamic FAP.

6.3.2 Structure of the Approaches for the Dynamic FAP

The dynamic FAP can be divided into three underlying problems, namely the static problem, the online problem and the repair problem. Hence, as in [55], the proposed approaches for the dynamic FAP apply three solution phases, which are the initial solution phase, the online assignment phase and the repair phase. The initial solution phase aims to solve the static problem (the initial sub-problem P_0). Note that if the static problem could not be solved by the initial solution phase, then the repair phase is applied.

After that, the online assignment phase is executed to solve the online problem (the sub-problems $P_1, \ldots, P_{20}$, consecutively). In this phase, no existing assignments are changed. If any request cannot be feasibly assigned, this creates the repair problem, which is solved in the repair phase by attempting to feasibly assign the unassigned requests with the minimum number of re-assigned requests. This phase includes two phases, namely the initial and the advanced repair phases. If the initial repair phase manages to achieve a feasible solution for the current time period, then the next time period is considered. Otherwise, the advanced repair phase is executed, then the approach proceeds to the next time period. The overall structure of the approach for the dynamic FAP is illustrated in Figure 6.2

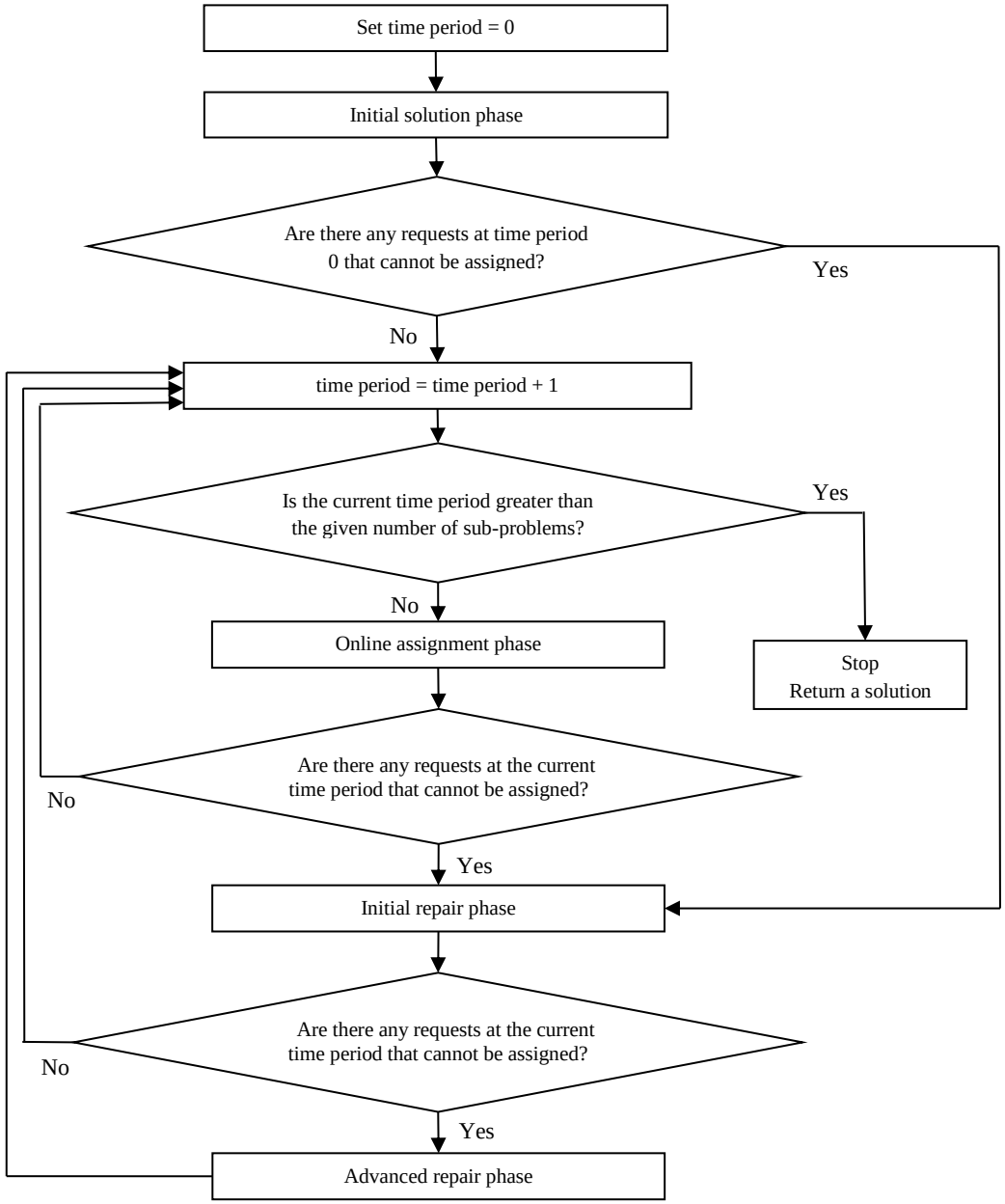

Figure 6.2: Overall structure of the approach for the dynamic FAP.

6.4 Components of the Approaches for the Dynamic FAP

In this section, the descriptions of the three phases of the approaches for the dynamic FAP, namely the initial solution phase, the online assignment phase and the repair phase, are given. Differences between our approaches and existing approaches are also specified in appropriate sections.

6.4.1 The Initial Solution Phase

The aim of this phase is to solve the initial sub-problem P_0. Hence, the initial solution phase which was proposed for TS (see Section 3.4.4) is applied here. If no feasible

solution can be found, then the repair phase (see Section 6.4.3) is applied. In contrast, the initial solution phase in Dupont's approach for the dynamic FAP in [55] aims to only find a feasible solution using the minimum frequency greedy algorithm (see Section 2.7) and if no feasible solution can be found, then the consistent neighbourhood in tabu search (CN-tabu) (see Section 2.7) is applied.

6.4.2 The Online Assignment Phase

The online assignment phase is considered after solving the initial sub-problem P_0. Hence, this phase aims to solve the remaining sub-problems $P_1, \ldots, P_{20}$, consecutively based on their time periods. In this phase, several techniques are applied and compared to solve these sub-problems, whereas Dupont's approach in [55] applied the minimum frequency greedy algorithm (see Section 2.7).

Several decisions need to be made in each time period in order to select requests and frequencies to be feasibly assigned. These decisions are related to the following questions:

- In what order should the new requests be considered?
- For the chosen request, which used feasible frequencies (if available) should be selected?
- For the chosen request, which unused feasible frequencies should be selected, if necessary?

The following stages give answers to the above questions consecutively.

i) Request selection stage

There are two ways to select requests from a sub-problem P_i, where requests can be considered either as individuals or as pairs based on the bidirectional constraints (see Equation 1.1). In the request selection stage, 8 different techniques are discussed as follows:

- *Technique 1:* the request that has the least number of feasible frequencies is selected. In case of a tie, the request that is involved in the highest number of constraints (based on only the currently known data) is selected. In case of a tie, one of them is randomly selected.

- *Technique 2:* the request that has the least number of feasible frequencies is selected. In case of a tie, one of them is randomly selected.
- *Technique 3:* the request that is involved in the highest number of constraints (based on only the currently known data) is selected. In case of a tie, one of them is randomly selected.
- *Technique 4:* the request is randomly selected.

The remaining techniques 5, 6, 7 and 8 are the same as techniques 1, 2, 3 and 4 respectively except that the requests are considered as pairs (instead of individuals) based on the bidirectional constraints (see Equation 1.1).

ii) Used feasible frequency selection stage

This stage is based on one of the following techniques:

- *The Ran technique:* one of the used feasible frequencies is randomly selected.
- *The Min technique:* the lowest value of the set of used feasible frequencies is selected. In case of a tie, one of them is randomly selected.
- *The Most technique:* the most occupied frequency (i.e. the frequency assigned to the most requests) in the set of used feasible frequencies is selected. In case of a tie, one of them is randomly selected.

The *Min* and *Most* techniques aim to maximize the number of frequencies that are not selected from the set of used frequencies. This allows more choices of used frequencies for requests that will appear at later time periods.

iii) Unused feasible frequency selection stage

This stage is based on one of the following techniques:

- *The Feas technique:* the frequency that is feasible for the most requests is selected. In case of a tie, one of them is randomly selected.
- *The Low technique:* the lowest value of the set of unused feasible frequencies is selected. In case of a tie, one of them is randomly selected.
- *The Gap technique:* this is a novel technique that aims to select a frequency from the set of unused feasible frequencies with the largest minimum gap between it and an already used frequency. In case of a tie, one of them is randomly selected.

The largest minimum gap leads to a greater probability that the interference constraints are satisfied. Example 6.1 clarifies the concept of the *Gap* technique.

Example 6.1:

Assume a request needs to be assigned to one of the unused feasible frequencies f_1 and f_2, whereas f_3 and f_4 are used infeasible frequencies. Figure 6.3 shows the gaps between these frequencies, where the red colour indicates a used infeasible frequency and the green colour indicates an unused feasible frequency.

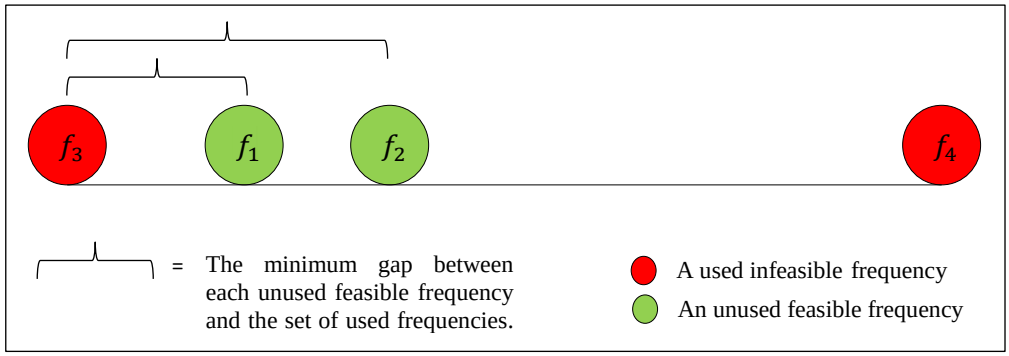

Figure 6.3: An example of the *Gap* technique.

The *Gap* technique starts by finding the minimum gap between each unused feasible frequency and the set of used frequencies as shown in Figure 6.3. After that, the unused feasible frequency that has the largest minimum gap is selected. In Figure 6.3, this corresponds to f_2.

Overall, the *Feas*, *Low* and *Gap* techniques aim to maximize the number of unused frequencies, which allows more choices of unused frequencies for requests that will appear at later time periods.

6.4.3 The Repair Phase

The repair phase is executed only if there is at least one request that could not be feasibly assigned by the online assignment phase as it was previously used in [55]. Hence, the aim of this phase is to attempt to feasibly assign the unassigned requests with the minimum number of re-assigned requests. This phase consists of two stages, namely the initial repair phase and the advanced repair phase.

6.4.3.1 The Initial Repair Phase

The initial repair phase is based on the following simple method: given that a request r_i cannot be feasibly assigned by the online assignment phase, then all the available frequencies for r_i are ordered according to the number of violations that would result when r_i is assigned to each of them. After that, these frequencies are considered in turn starting with the frequency which would result in the minimum number of violations. Assume that the minimum number of violations corresponds to the frequency f_k. Then, the set of requests that clash with r_i with respect to some constraints when f_k is assigned to r_i is produced. After that, each clashed request is attempted to be feasibly re-assigned. In case one of these cannot be feasibly re-assigned, then all re-assigned requests are reversed and the next frequency is considered. Example 6.2 clarifies the concept of the initial repair phase. Hence, this phase is based on greedy assignments; whereas the initial repair phase in Dupont's approach in [55] is based on the branch and bound algorithm (see Section 2.7).

Example 6.2:

Assume r_i could not be feasibly assigned by the online assignment phase, then we attempt to feasibly assign r_i by the initial repair phase. Assume that the available frequencies for r_i are f_j, f_k and f_l. Each frequency is assigned to r_i in turn and the number of violations is given in Table 6.1.

Available frequencies	Number of violations
f_j	6
f_k	4
f_l	10

Table 6.1: Number of violations for each available frequency when it is assigned to r_i.

The first selected frequency is f_k because it results in the minimum number of violations. If this does not result in feasible assignments, then the second selection is f_j, which results in the next smallest number of violations. In case this does not lead to feasible assignments, then the last selection is f_l.

6.4.3.2 The Advanced Repair Phase

This phase is only executed if the initial repair phase results in some unassigned requests. Then, each unassigned request is assigned to the frequency that results in the

minimum number of violations. After that, the number of violations is reduced using the advanced repair phase which is based on either a tabu search repair phase (TSRP), or a hyper heuristic repair phase (HHRP). TSRP is based on the tabu search algorithm described in Chapter 3 and HHRP is based on the hyper heuristic described in Chapter 5, where the objective of these algorithms are modified to find a feasible solution instead of the optimal solution. In contrast, the advanced repair phase in the Dupont's approach in [55] is based on the CN-tabu algorithm (see Section 2.7).

6.5 Experiments and Results

This section compares the performance of various approaches for the dynamic FAP in the online assignment phase and the repair phase. After that, the best approach is compared with Dupont's approach in [55] using the generated datasets in this thesis.

In this study, these approaches were coded using FORTRAN 95 and all experiments were conducted on a 3.0 GHz Intel Core I3-2120 Processor (2nd Generation) with 8GB RAM and a 1TB Hard Drive.

6.5.1 The Online Assignment Phase

The performance of various approaches using different techniques in each selection stage (see Section 6.4.2) is compared using TSRP as the advanced repair phase. In this phase, we have 8 potential techniques in the first stage **(**request selection stage), 3 potential techniques in the second stage (used feasible frequency selection stage) and 3 potential techniques in the third stage (unused feasible frequency selection stage). In total, this gives 72 different various approaches to be compared. Moreover, each approach is tested using 100 dynamic FAP instances. Therefore, considering all the various approaches takes excessive time. Hence, these approaches are compared based on 3 experiments as follows: Experiment 1 fixes the technique of the second and third stage to the *Ran* and *Feas* techniques, respectively, and compares the 8 techniques of the first stage. Experiment 2 tests the techniques of the second stage by fixing the best technique of the first stage and using the *Feas* technique for the third stage, as before. Experiment 3 tests the techniques of the third stage while the best techniques of the first and second stages are fixed.

In order to select the best approach in each experiment, three types of comparisons are used, namely comparisons A, B and C. These comparisons firstly rank the approaches, where the best approach is given the lowest rank. In case of a tie, these are given the average rank.

- Comparison A applies *the instance average rank*, which is calculated for each instance by taking the average rank of all dynamic FAP instances that are generated from that instance.
- Comparison B applies *the dynamic average rank*, which is calculated for each percentage of the number of requests known at time period 0 by taking the average rank of all dynamic FAP instances corresponding to that percentage.
- Comparison C applies *the total rank*, which is calculated by summing the ranks of all the dynamic FAP instances for each approach.

The objective of the three types of comparisons is to find the minimum average rank and the minimum total rank. These comparisons are not independent of one another, but should indicate whether certain techniques work better for different levels of dynamism or for different instances.

Experiment 1: this experiment compares 8 different techniques of the request selection stage, while selecting the *Ran* and *Feas* techniques for used and unused feasible frequency selection stages, respectively. The results of the three types of comparisons are presented in Table 6.2, Table 6.3 and Figure 6.4. Note that a bold number shows the best results among these approaches with different techniques of the requests selection stage.

- *Comparison A*

Instance	Techniques of the request selection stage							
	1	2	3	4	5	6	7	8
CELAR 01	5.45	4.70	7.00	7.70	2.55	**2.10**	3.45	3.05
CELAR 02	4.65	4.45	7.05	7.00	**3.15**	3.25	3.55	3.90
CELAR 03	6.40	3.45	7.05	7.30	3.00	**2.70**	**2.70**	3.40
CELAR 04	2.70	3.00	7.40	7.10	**3.20**	3.90	5.20	11.60
CELAR 11	5.85	5.15	6.40	7.55	2.50	3.35	3.15	**2.05**
GRAPH 01	5.70	6.00	6.30	7.80	2.50	**2.35**	3.05	2.30
GRAPH 02	5.70	5.60	7.00	7.70	**2.15**	2.30	2.60	2.95
GRAPH 08	5.55	5.65	7.10	7.70	**2.00**	3.15	2.55	2.30
GRAPH 09	5.90	6.00	7.10	5.80	2.45	2.85	**2.20**	3.70
GRAPH 14	5.55	5.85	7.00	7.60	2.80	2.50	**1.85**	2.85
Total	53.50	49.80	69.40	73.30	**26.30**	28.50	30.30	38.10

Table 6.2: The instance average rank for each approach based on Experiment 1.

Table 6.2 shows that using techniques 5, 6, 7 and 8 are better than using techniques 1, 2, 3 and 4. This indicates that considering requests as pairs based on bidirectional constraints (see Equation 1.1) led to better results than considering them individually. In this comparison, the approach that applies technique 5 is the best approach.

- *Comparison B*

Dynamic instances	Techniques of the request selection stage							
	1	2	3	4	5	6	7	8
0%	6.05	5.90	7.85	8.76	2.95	2.85	2.83	**2.81**
10%	5.22	5.67	6.88	7.78	**2.00**	2.56	3.33	2.56
20%	5.67	5.11	7.00	7.78	**2.39**	2.44	2.50	3.11
30%	5.44	5.11	6.61	7.89	**2.56**	2.61	3.00	2.78
40%	5.61	5.11	7.00	7.89	2.11	2.89	**2.06**	3.33
50%	5.25	5.15	7.40	7.30	**1.80**	2.80	3.25	3.05
60%	5.80	3.75	7.30	7.60	3.20	2.80	3.10	**2.45**
70%	5.20	5.5	6.70	6.85	3.30	**2.00**	3.25	3.20
80%	5.70	4.95	6.40	6.65	2.60	4.00	**2.45**	3.25
90%	5.50	5.30	6.75	6.00	3.25	3.15	3.60	**2.45**
Total	55.40	51.50	69.80	74.50	**26.20**	28.10	29.40	28.90

Table 6.3: The dynamic average rank for each approach based on Experiment 1.

This comparison confirms that considering the requests as pairs improved the performance of the approach for the dynamic FAP. Therefore, we focus on techniques 5, 6, 7 and 8, where each of them produced the best result for at least one instance, but technique 5 achieves the best performance based on the total dynamic average rank. Additionally, it produced the best result on 4 instances, more than any other techniques. Hence, technique 5 is recommended.

- Comparison C

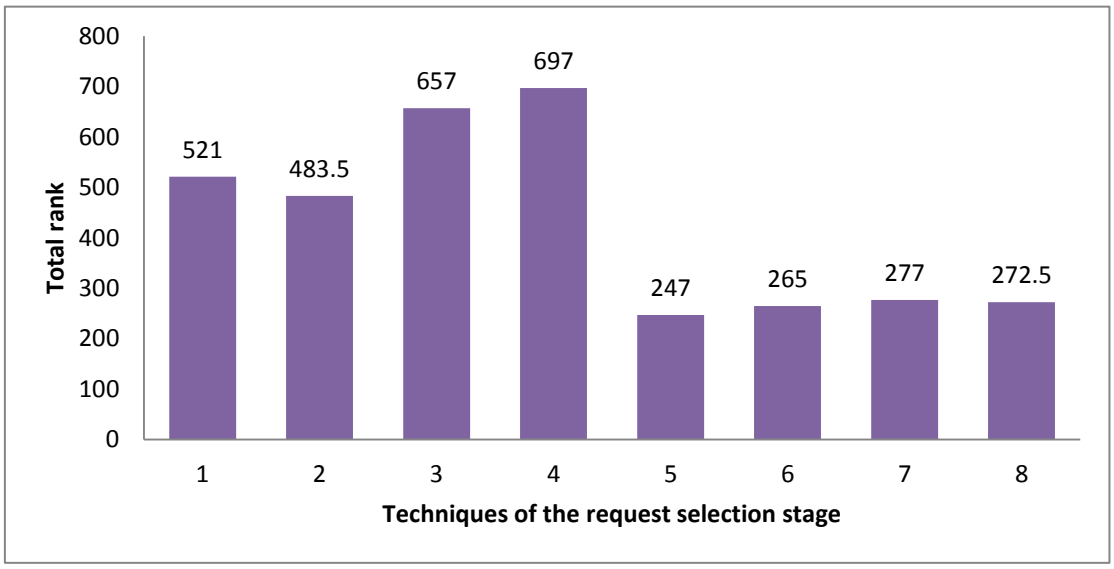

Figure 6.4: The total rank for each approach based on Experiment 1.

Figure 6.4 shows that the minimum total rank is 247, which suggests that the best technique based on comparison C is also technique 5.

Overall, all the three types of comparisons suggest that the best approach is based on using technique 5, which considers requests as pairs based on the bidirectional constraints (see Equation 1.1). In terms of the run time, based on the Wilcoxon signed-rank test, there is no significant difference between them. Hence, in the request selection stage the best technique is technique 5.

Experiment 2: this experiment compares 3 different techniques of the used feasible frequency selection stage, while fixing the remaining techniques by selecting technique 5 for the request selection stage (see Experiment 1) and the *Feas* technique for the unused feasible frequency selection stage. The results of the three comparisons are presented in Table 6.4, Table 6.5 and Figure 6.5. Note that a bold number shows the best results among these approaches with different techniques of the used feasible frequency selection stage.

- *Comparison A*

Instance	Techniques of the used feasible frequency selection stage		
	Ran	*Min*	*Most*
CELAR 01	2.20	2.15	**1.65**
CELAR 02	2.15	**1.80**	2.05
CELAR 03	2.20	1.95	**1.85**
CELAR 04	**1.90**	**1.90**	2.20
CELAR 11	1.95	2.45	**1.60**
GRAPH 01	2.10	**1.70**	2.20
GRAPH 02	**1.70**	2.10	2.20
GRAPH 08	**1.65**	1.95	2.40
GRAPH 09	2.05	**1.75**	2.20
GRAPH 14	2.70	2.30	**1.00**
Total	20.60	20.10	**19.40**

Table 6.4: The instance average rank for each approach based on Experiment 2.

Although there is little difference in the performance between these approaches as shown in Table 6.4, the best total instance average rank is found by the *Most* technique. Hence, comparison A recommends the *Most* technique for the used feasible frequency selection stage.

- *Comparison B*

Dynamic instances	Techniques of the used feasible frequency selection stage		
	Ran	*Min*	*Most*
0%	2.53	2.50	**1.64**
10%	2.33	**1.83**	**1.83**
20%	**1.78**	2.22	2.00
30%	2.28	2.00	**1.72**
40%	1.83	**1.72**	2.44
50%	**1.70**	2.30	2.00
60%	2.10	**1.95**	**1.95**
70%	2.25	1.90	**1.85**
80%	**1.85**	1.95	2.20
90%	2.30	1.95	**1.75**
Total	20.90	20.30	**19.40**

Table 6.5: The dynamic average rank for each approach based on Experiment 2.

Table 6.5 shows that the majority of the minimum dynamic average rank is found by the *Most* technique, which records the best results for 6 out of the 10 instances. In this comparison, the total dynamic average rank also recommends the *Most* technique.

- *Comparison C*

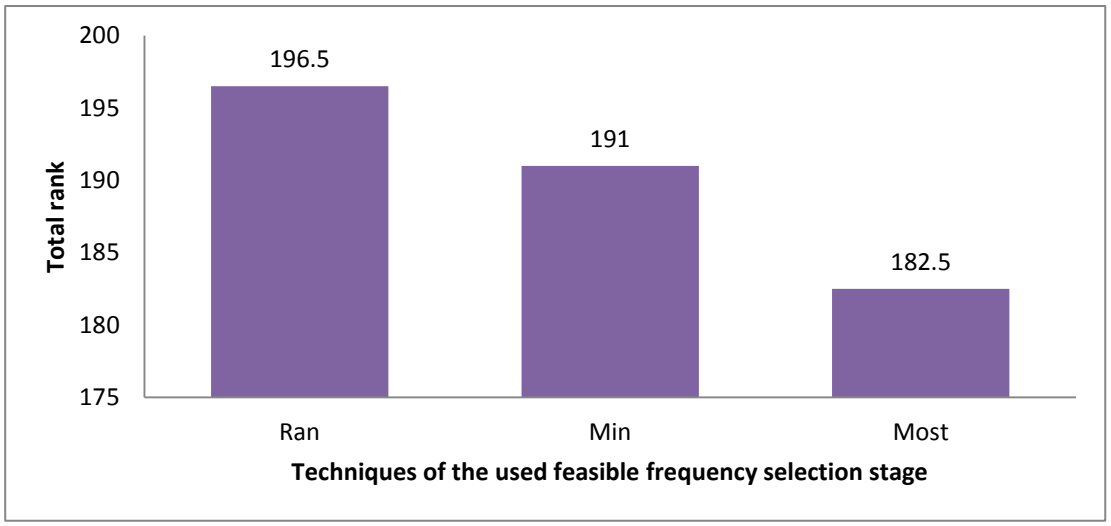

Figure 6.5 The total rank for each approach based on Experiment 2.

Figure 6.5 shows that the minimum total rank is 182.5, which indicates that the best technique based on comparison C is again the *Most* technique.

Overall, although there is little difference in the results achieved by the 3 different techniques in the used feasible frequency selection stage, all the three types of comparisons show that the best approach is based on using the *Most* technique. In terms of the run time, based on the Wilcoxon signed-rank test, there is no significant difference between them.

***Experiment 3**:* this experiment compares 3 different techniques of the unused feasible frequency selection stage, while fixing the remaining techniques by selecting technique 5 for the request selection stage (see Experiment 1) and the *Most* technique for the used feasible frequency selection stage (see Experiment 2). The results of the three comparisons are presented in Table 6.6, Table 6.7 and Figure 6.6. Note that a bold number shows the best results among these approaches with different techniques of the unused feasible frequency selection stage.

- *Comparison A*

Instance	Techniques of the unused feasible frequency selection stage		
	Feas	*Low*	*Gap*
CELAR 01	2.20	**1.65**	2.15
CELAR 02	1.90	2.35	**1.75**
CELAR 03	2.05	**1.95**	2.00
CELAR 04	2.10	**1.80**	2.10
CELAR 11	**1.50**	2.15	2.35
GRAPH 01	1.90	2.40	**1.70**
GRAPH 02	2.35	1.95	**1.70**
GRAPH 08	2.15	**1.60**	2.25
GRAPH 09	2.00	2.35	**1.65**
GRAPH 14	1.60	3.00	**1.40**
Total	19.80	21.20	**19.10**

Table 6.6: The instance average rank for each approach based on Experiment 3.

Table 6.6 shows that the *Gap* techniques give the best results on five instances. Moreover, the total instance average rank shows that the *Gap* technique achieved the best performance. Hence, comparison A recommends the *Gap* technique.

- *Comparison B*

Dynamic instances	Techniques of the unused feasible frequency selection stage		
	Feas	*Low*	*Gap*
0%	2.17	2.62	**1.88**
10%	2.06	2.11	**1.83**
20%	**1.78**	**1.78**	2.44
30%	2.00	2.11	**1.89**
40%	2.38	1.94	**1.67**
50%	**2.00**	**2.00**	**2.00**
60%	**1.75**	2.15	2.10
70%	**1.65**	2.35	2.00
80%	2.30	2.40	**1.30**
90%	**1.85**	2.10	2.05
Total	19.90	21.60	**19.20**

Table 6.7: The dynamic average rank for each approach based on Experiment 3.

Table 6.7 shows that using the *Feas* technique for the unused feasible frequency selection stage achieved 5 minimum dynamic average ranks, the *Low* technique achieved 2 and the *Gap* technique achieved 6 out of 10. Hence, the *Gap* technique is recommended, which agrees with the total dynamic average rank.

- *Comparison C*

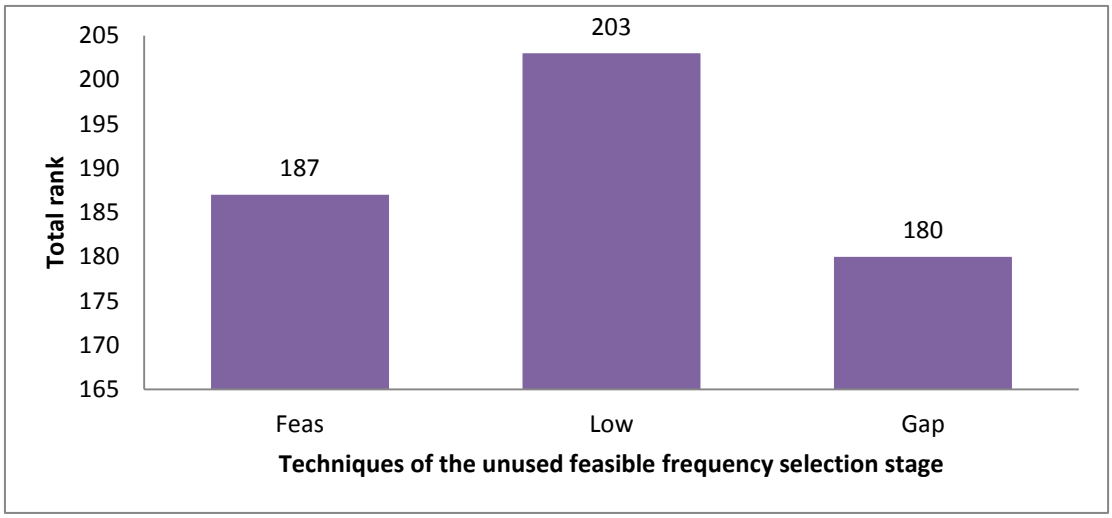

Figure 6.6: The total rank for each approach based on Experiment 3.

Figure 6.6 shows that the minimum number of the total rank was 180, which resulted from using the *Gap* technique. Hence, the best technique based on the comparison C is the *Gap* technique for the unused feasible frequency selection stage.

Overall, the three types of comparisons suggest that the *Gap* technique should be adopted in the unused feasible frequency selection stage. In terms of the run time, based on the Wilcoxon signed-rank test, there is no significant difference between them.

Based on the above three experiments, the best approach for the dynamic FAP is based on technique 5 in the request selection stage, the *Most* technique in the used feasible frequency selection stage and the *Gap* technique in the unused feasible frequency selection stage. This gives a complete description of the online assignment phase.

6.5.2 The Repair Phase

This section presents and compares the performance of various approaches for the dynamic FAP in two stages: the first uses the initial repair phase only and the second includes the advanced repair phase. The results in this section include the number of used frequencies in a feasible solution, the run time and the number of re-assigned requests in the repair phase (denoted by Repair). Note that a dash "-" means that no feasible solution is found.

6.5.2.1 The Initial Repair Phase

The performance of the approach for the dynamic FAP using only the initial repair phase is given in Table 6.8.

Instance	Number of requests known at time period 0									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
CELAR 01	26	28	-	30	-	-	-	26	-	30
Time	8.5 sec	19.5 sec	-	2 min	-	-	-	7 min	-	11 min
Repair	54	854	-	252	-	-	-	250	-	102
CELAR 02	16	18	18	20	16	16	18	18	16	14
Time	0.6 sec	2.9 sec	8.5 sec	13 sec	30 sec	33 sec	1 min	49 sec	54 sec	59 sec
Repair	200	182	160	146	114	114	10	50	48	10
CELAR 03	24	26	20	24	-	26	26	24	28	-
Time	1.7 sec	7.3 sec	16.1 sec	54.1 sec	-	1 min	1.2 min	3.2 min	2.5 min	-
Repair	38	46	334	268	-	228	24	106	4	-
CELAR 04	46	46	46	46	46	46	46	46	46	46
Time	24 sec	24.3 sec	56.3 sec	1.1 min	1.5 min	1.5 min	1.6 min	2.1 min	2.3 min	2.3 min
Repair	18	24	20	18	21	18	24	12	10	16
CELAR 11	-	-	-	-	-	-	-	-	-	-
Time	-	-	-	-	-	-	-	-	-	-
Repair	-	-	-	-	-	-	-	-	-	-
GRAPH 01	-	24	28	-	-	-	22	22	22	20
Time	-	2.8 sec	7.5 sec	-	-	-	47 sec	1.1 min	1 min	2.1 min
Repair	-	18	16	-	-	-	72	52	48	20
GRAPH 02	22	-	26	-	-	28	20	26	18	22
Time	1.7 sec	-	38 sec	-	-	1.5 min	2.8 min	3.1 min	4.5 min	4.8 min
Repair	400	-	26	-	-	6	112	50	86	44
GRAPH 08	-	-	-	-	-	-	-	-	-	-
Time	-	-	-	-	-	-	-	-	-	-
Repair	-	-	-	-	-	-	-	-	-	-
GRAPH 09	-	-	-	-	-	-	-	-	-	-
Time	-	-	-	-	-	-	-	-	-	-
Repair	-	-	-	-	-	-	-	-	-	-
GRAPH 14	16	12	14	16	14	16	20	16	14	14
Time	7.5 sec	42.5 sec	58.5 sec	2.1 min	3.6 min	4.2 min	6.9 min	19 min	13 min	18 min
Repair	916	808	760	634	564	464	152	280	166	76

Table 6.8: Results of the approach for the dynamic FAP using the initial repair phase.

Table 6.8 shows that this approach achieved feasible solutions for all dynamic FAP instances for CELAR 02, CELAR 04 and GRAPH 14. In contrast, this approach could not achieve a feasible solution for all dynamic FAP instances for GRAPH 08 and GRAPH 09. In terms of the number of re-assigned requests (labelled as Repair), this number fluctuates, but a significant number of re-assignments are needed for most

instances. Moreover, Table 6.8 shows that the run time increased with the number of requests known at the time period 0.

6.5.2.2 The Advanced Repair Phase

The performance of the approach for the dynamic FAP using two types of advanced repair phase is compared.

i) The tabu search repair phase (TSRP): the results of the approach for the dynamic FAP using TSRP as the advanced repair phase are presented in Table 6.9.

Instance	Number of requests known at time period 0									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
CELAR 01	26	26	34	26	32	30	30	26	32	26
Time	5.8 sec	9.2 sec	15.5 sec	38.5 sec	48.5 sec	57.5 sec	1 min	2.13 min	2.8 min	3.7 min
Repair	54	172	40	54	16	22	36	250	42	4
CELAR 02	16	16	18	16	18	16	18	16	16	14
Time	0.4 sec	0.5 sec	0.7 sec	1.1 sec	3.3 sec	4.8 sec	7 sec	3.4 sec	12 sec	11 sec
Repair	200	182	160	146	114	114	10	50	48	10
CELAR 03	24	24	22	28	22	26	22	20	24	18
Time	2.1 sec	3.3 sec	5.9 sec	6.2 sec	13.1 sec	5.5 sec	19.1 sec	24.1 sec	28.1 sec	29.1 sec
Repair	38	46	90	268	208	66	60	106	70	50
CELAR 04	46	46	46	46	46	46	46	46	46	46
Time	24 sec	24.3 sec	44 sec	1.1 min	1.1 min	1.3 min	1.5 min	2.1 min	2.3 min	2.3 min
Repair	18	24	8	20	10	18	20	12	10	16
CELAR 11	42	42	40	42	44	32	42	42	42	40
Time	26.3 sec	23.3 sec	2.6 min	1.9 min	3.9 min	1.6 min	3.7 min	4 min	4.7 min	9.3 min
Repair	28	52	418	336	432	24	424	376	376	350
GRAPH 01	20	30	24	26	24	22	22	24	22	22
Time	0.5 sec	0.5 sec	2.1 sec	1.1 sec	11 sec	3.8 sec	11 sec	4.2 sec	15 sec	6.7 sec
Repair	52	6	10	14	10	34	72	52	48	20
GRAPH 02	22	22	28	20	24	24	24	28	22	20
Time	1.2 sec	1.4 sec	3 sec	2.9 sec	5.1 sec	8.1 sec	30.1 sec	30.1 sec	43.1 sec	1.1 min
Repair	400	4	28	22	256	210	46	60	86	42
GRAPH 08	30	36	44	38	36	32	40	36	34	32
Time	10.3 sec	17.3 sec	15.3 sec	14.3 sec	19.3 sec	26.3 sec	59.3 sec	1.1 min	1.5 min	2.1 min
Repair	24	24	44	24	38	78	28	16	34	30
GRAPH 09	40	44	42	38	42	42	44	42	40	30
Time	18.5 sec	20.5 sec	29.5 sec	42.5 sec	1.1 min	1.2 min	1.9 min	2.2 min	3.9 min	4.1 min
Repair	22	24	46	38	46	58	30	10	2	6
GRAPH 14	16	16	16	18	18	14	12	12	14	14
Time	4.5 sec	5.5 sec	7.5 sec	16.5 sec	31.5 sec	1.1 min	2.2 min	3.1 min	7.3 min	15 min
Repair	916	916	760	634	564	6	344	286	166	76

Table 6.9: Results of the approach for the dynamic FAP using TSRP as the advanced repair phase.

Table 6.9 shows that using TSRP as the advanced repair phase resulted in feasible solutions for all instances. The number of re-assigned requests (labelled as Repair) fluctuates and does not have a relationship with the number of requests known at time period 0. Moreover, Table 6.9 shows that the run time increased with the number of requests known at the time period 0. The effect of the number of requests known at time period 0 on the run time is shows in Figure 6.7.

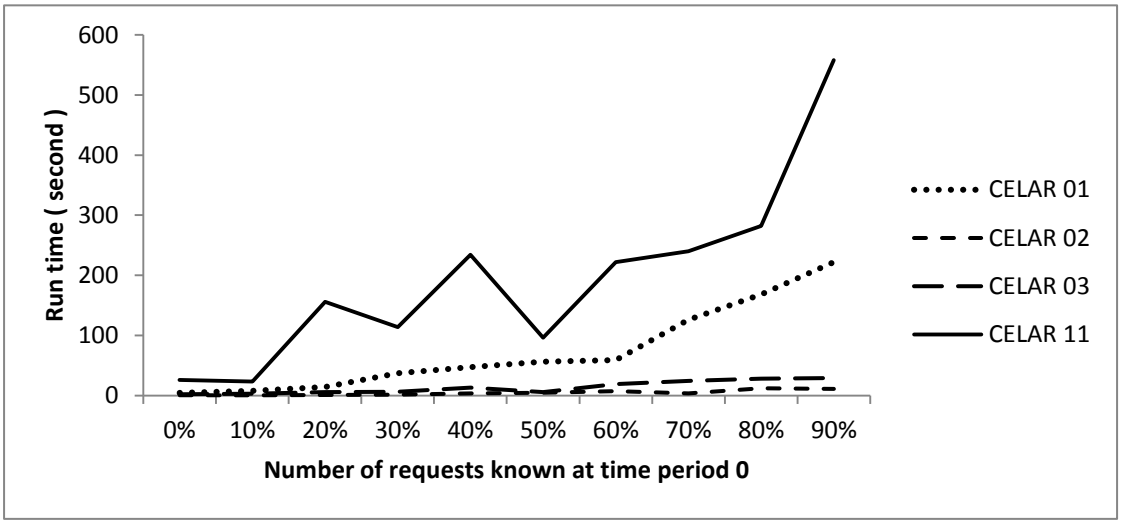

Figure 6.7: The run time for all dynamic FAP instances of the selected instances.

Figure 6.7 shows that the run time of each instance increased with the number of requests known at time period 0. Moreover, it indicates that the approach is fastest when no requests are known at time period 0. In contrast, the approach is slowest when 90% of the requests are known at time period 0.

ii) The hyper heuristic repair phase (HHRP): the results of the approach for the dynamic FAP using HHRP as the advanced repair phase are given in Table 6.10.

Instance	Number of request known at time period 0									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
CELAR 01	26	28	38	30	30	28	26	26	26	30
Time	8.5 sec	19.5 sec	9.8 min	2 min	6.2 min	6 min	6.1 min	7 min	10 min	11 min
Repair	54	854	592	252	322	28	94	250	174	102
CELAR 02	16	18	18	20	16	16	18	18	18	14
Time	0.6 sec	2.9 sec	8.5 sec	13 sec	30 sec	33 sec	1 min	49 sec	54 sec	59 sec
Repair	200	182	160	146	24	114	10	50	48	10
CELAR 03	24	26	20	24	22	26	26	24	28	20
Time	1.7 sec	7.3 sec	16.1 sec	54.1 sec	30 sec	1 min	1.2 min	3.2 min	2.5 min	3.4 min
Repair	38	46	334	268	114	228	24	106	4	50
CELAR 04	46	46	46	46	46	46	46	46	46	46
Time	24 sec	24.3 sec	56.3 sec	1.1 min	1.5 min	1.5 min	1.6 min	2.1 min	2.3 min	2.3 min
Repair	18	24	20	18	21	18	24	12	10	16
CELAR 11	44	44	46	-	-	46	-	-	46	-
Time	28 min	22 min	20 min	-	-	29 min	-	-	21 min	-
Repair	656	656	652	-	-	650	-	-	646	-
GRAPH 01	26	24	28	26	26	28	22	22	22	20
Time	0.6 sec	2.8 sec	7.5 sec	11 sec	40 sec	37 sec	47 sec	1.1 min	1 min	2.1 min
Repair	6	18	16	0	2	6	72	52	48	20
GRAPH 02	22	22	26	22	20	28	20	26	18	22
Time	1.7sec	6.8 sec	38 sec	25 sec	48 sec	1.5 min	2.8 min	3.1 min	4.5 min	4.8 min
Repair	400	12	26	24	58	6	112	50	86	44
GRAPH 08	34	34	38	38	30	34	32	32	32	28
Time	1.1 min	26.3 sec	51.3 sec	1.2 min	2.1 min	2.2 min	4.4 min	3.1 min	3.8 min	6.1 min
Repair	258	216	172	118	208	262	84	206	78	292
GRAPH 09	40	38	38	38	38	38	40	38	38	24
Time	20.5 sec	1.1 min	1.4 min	2.2 min	2.9 min	5.7 min	5.9 min	11 min	10 min	17 min
Repair	180	100	244	102	74	200	322	358	48	18
GRAPH 14	16	12	14	16	14	16	20	16	14	14
Time	7.5 sec	42.5 sec	58.5 sec	2.1 min	3.6 min	4.2 min	6.9 min	19 min	13 min	18 min
Repair	916	808	760	634	564	464	152	280	166	76

Table 6.10: Results of the approach for the dynamic FAP using the HHRP as the advanced repair phase.

Table 6.10 shows that using HHRP as the advanced repair phase resulted in feasible solutions for almost all the instances. The number of re-assigned requests (labelled as Repair) fluctuates with no clear pattern. Moreover, the run time increased with the number of requests known at time period 0.

iii) Results comparison of using TSRP and HHRP as the advanced repair phase: the results of the approach using TSRP (see Table 6.9) achieved feasible solutions for all the instances, whereas the approach using HHRP (see Table 6.10) did not achieve feasible solutions for all the instances. The number of dynamic FAP instances for each CELAR or GRAPH instance for which TSRP or HHRP re-assigned fewer requests is given in Table 6.11.

Instance	TSRP	HHRP	Total
CELAR 01	8	0	8
CELAR 02	0	1	1
CELAR 03	2	3	5
CELAR 04	3	1	4
CELAR 11	10	0	10
GRAPH 01	2	4	6
GRAPH 02	4	4	8
GRAPH 08	10	0	10
GRAPH 09	10	0	10
GRAPH 14	1	3	4
Total	50	16	-

Table 6.11: The number of dynamic FAP instances for each CELAR or GRAPH instance for which TSRP or HHRP re-assigned fewer requests.

Note that the sum of each row in Table 6.11 may not be equal to 10 because the same numbers of re-assigned requests were achieved by both approaches using TSRP or HHRP, or the initial repair phase was able to find feasible solutions without requiring the use of the advanced repair phase. Furthermore, Table 6.11 shows that TSRP re-assigned fewer requests in 50 instances, whereas HHRP re-assigned fewer requests in 16 instances. This suggests that TSRP is the best type of the advanced repair phase in terms of achieving feasible solutions with the minimum number of re-assigned requests.

The total run time of all dynamic FAP instances for each CELAR or GRAPH instance of two approaches using TSRP or HHRP as the advanced repair phase are shown in Figure 6.8.

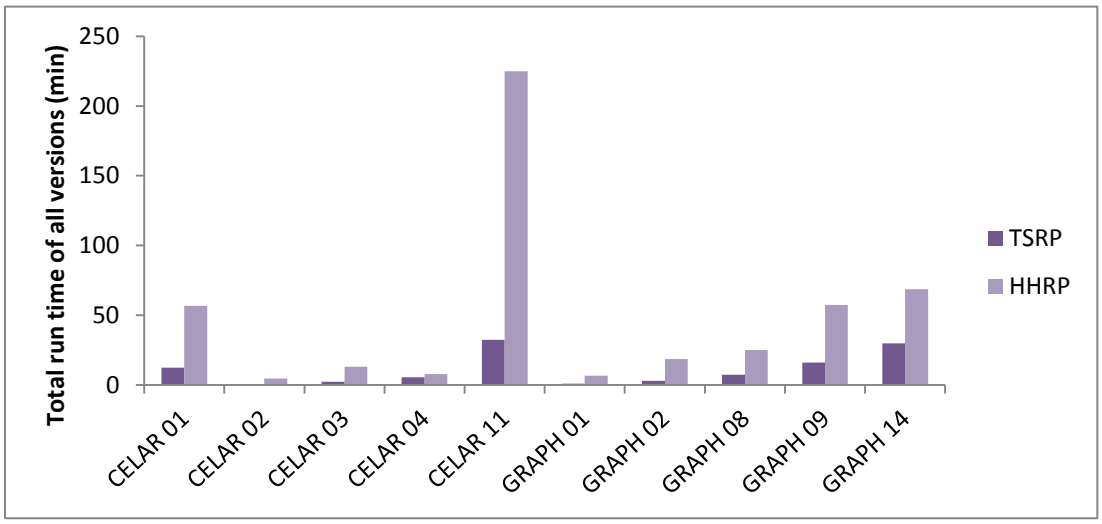

Figure 6.8 The total run time of all dynamic FAP instances for each CELAR or GRAPH instance using two different types of the advanced repair phase.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the performance using TSRP as the advance repair phase gives significantly better results than using HHRP. The total run time for each approach using TSRP or HHRP for all instances is shown in Figure 6.9.

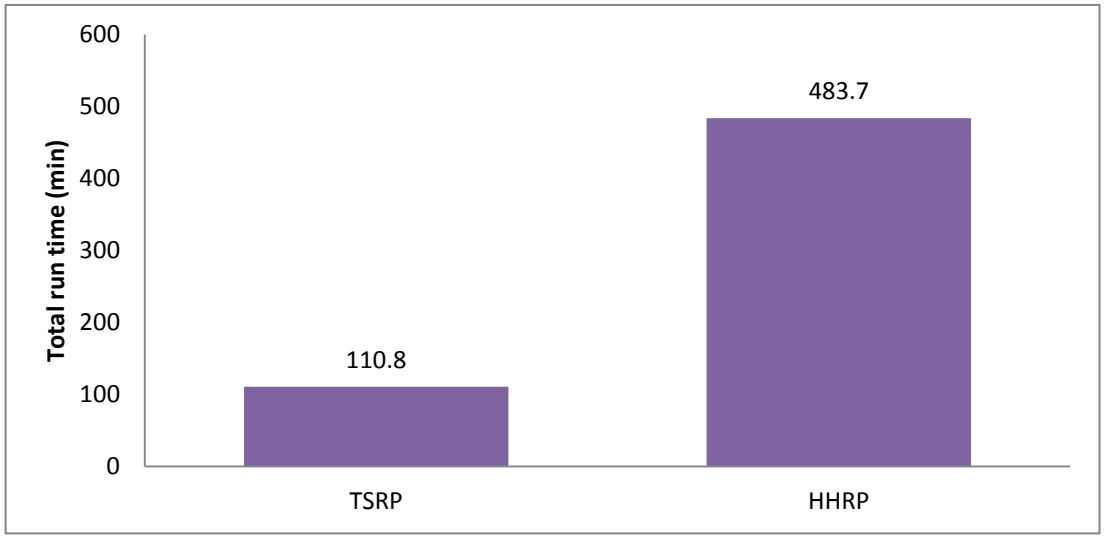

Figure 6.9: The total run time of all dynamic FAP instances using two different types of the advanced repair phase.

To sum up, using TSRP as the advanced repair phase for the dynamic FAP achieved the best results in terms of the objective of the dynamic FAP, i.e. achieving feasible solutions with the minimum number of re-assigned requests. Moreover, using TSRP resulted in the best run time, indicating that this is more appropriate than HHRP.

6.5.3 Results Comparison with Other Approaches

The performance of the Dupont's approach in [55] is compared with our approach for the dynamic FAP. Private correspondence with the authors in [55] revealed that they had not kept copies of either their dynamic datasets or their software, so we re-implemented Dupont's approach in [55] to solve the dynamic FAP using our generated dynamic FAP datasets in this thesis and the results are shown in Table 6.12.

Instance	Number of requests known at time period 0									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
CELAR 01	36	30	42	34	30	32	30	34	30	34
Time	7.2 min	8.9 min	9.2 min	11 min	11.5 min	21 min	24 min	26 min	30 min	31 min
Repair	230	912	620	340	322	320	240	430	234	420
CELAR 02	18	18	18	22	22	18	18	18	18	16
Time	2 sec	4.8 sec	6.2 sec	10 sec	18 sec	38 sec	55 sec	57 sec	1.3 min	1.8 min
Repair	200	190	190	160	70	30	90	40	80	30
CELAR 03	32	30	34	24	30	32	36	30	30	30
Time	20 sec	45 sec	50 sec	54 sec	52 sec	1.8 min	2.1 min	4.1 min	4.7 min	5.2 min
Repair	70	200	430	330	204	220	110	98	60	40
CELAR 04	46	46	46	46	46	46	46	46	46	46
Time	1.7 min	2.3 min	4.1 min	6.1 min	6.2 min	8.1 min	10.2 min	11.9 min	20.1 min	30 min
Repair	42	80	32	22	52	24	92	88	22	72
CELAR 11	42	44	44	42	44	44	44	42	40	40
Time	8.1 min	8 min	9.5 min	9.7 min	10 min	11 min	13 min	14.7 min	17 min	18 min
Repair	356	556	312	400	334	366	342	322	398	338
GRAPH 01	40	40	42	32	36	40	42	44	38	38
Time	38 sec	54 sec	2.3 min	3.1 min	3.4 min	3.7 min	4.7 min	5.5 min	7.1 min	7.2 min
Repair	110	42	146	204	172	190	176	144	142	120
GRAPH 02	36	40	40	40	40	40	42	38	40	42
Time	1.7 min	2.8 min	2.9 min	3 min	3.1 min	4.2 min	4.8 min	5.2 min	5.3 min	6.8 min
Repair	212	76	252	302	294	306	258	256	256	242
GRAPH 08	48	46	48	44	44	44	48	48	42	44
Time	1.6 min	1.5 min	2 min	2.1 min	2.8 min	3.5 min	3.4 min	4.5 min	4.8 min	5.1 min
Repair	252	194	264	306	276	300	286	288	318	302
GRAPH 09	46	46	42	46	46	44	44	46	44	42
Time	4 min	4.1 min	5.3 min	6.2 min	6.7 min	8.5 min	8.8 min	10.2 min	12 min	17 min
Repair	308	178	300	298	280	278	274	290	308	302
GRAPH 14	26	26	36	34	34	32	34	34	34	28
Time	9.1 min	10 min	11 min	12 min	13.4 min	14 min	15.5 min	19.2 min	24.1 min	26 min
Repair	916	916	840	744	564	912	340	230	850	420

Table 6.12: Results of Dupont's approach for the dynamic FAP.

Table 6.12 shows that Dupont's approach achieved feasible solutions for all instances. By looking at the number of re-assigned requests (labelled as Repair), it can be seen that this number fluctuates and does not have a relationship with the number of requests known at time period 0.

By comparing the performance of Dupont's approach (see Table 6.12) with our approach (see Table 6.9), it is found that Dupont's approach gave a higher number of re-assigned requests compared with our approach. Figure 6.10 shows the average number of re-assigned requests for each instance in our approach and Dupont's approach.

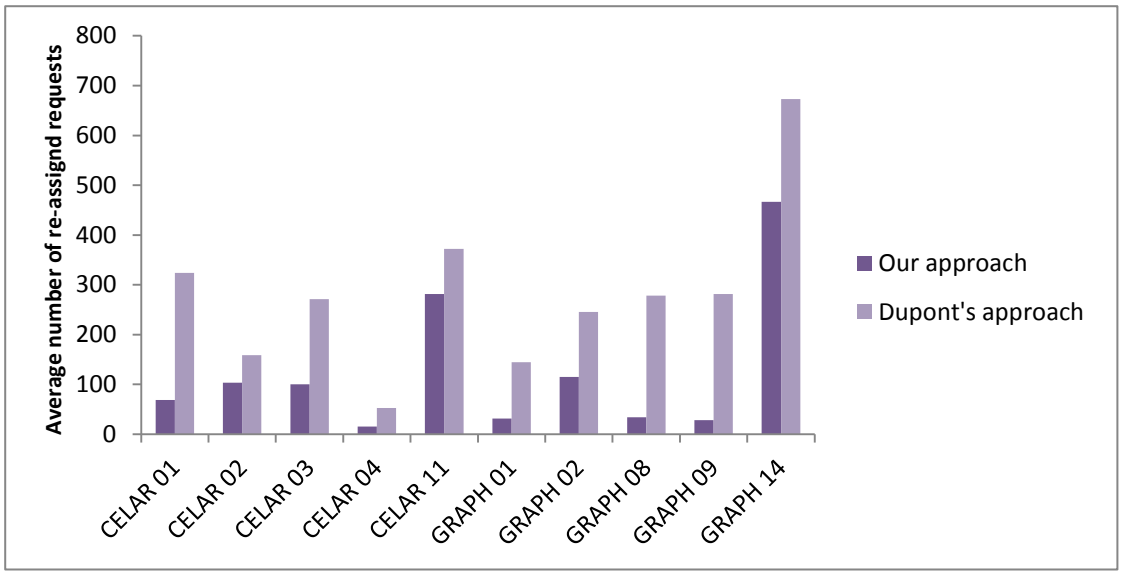

Figure 6.10: Average number of re-assigned requests of our approach and Dupont's approach.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the total number of re-assigned requests of our approach is significantly better than Dupont's approach. Hence, our approach achieved better results in terms of achieving feasible solutions with the minimum number of re-assigned requests. The average run time of our approach and Dupont's approach is shown in Figure 6.11

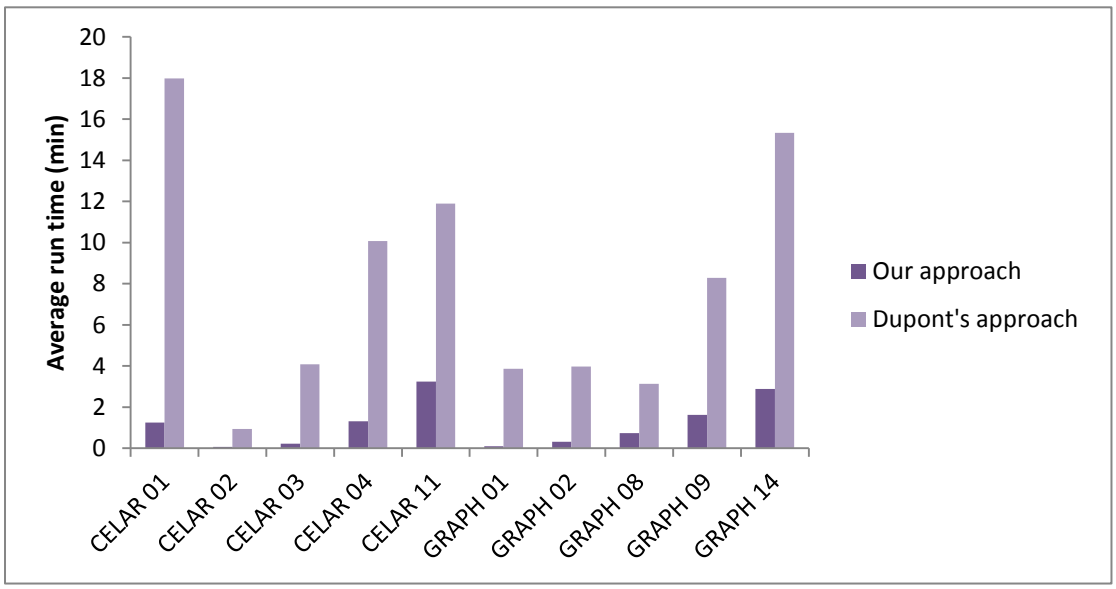

Figure 6.11: The average run time of our approach and Dupont's approach.

It is found by the Wilcoxon signed-rank test at the 0.05 significance level that the average run time of our approach is significantly better than Dupont's approach.

Overall, based on the comparisons above, it is found that our approach is competitive compared with Dupont's approach.

6.6 An Approach for the Static FAP

In this study, a novel approach called the dynamic tabu search (DTS) is proposed to solve the MO-FAP, which is a variant of the static FAP. This approach is inherited from the approach for the dynamic FAP (see Section 6.3). It models the static FAP as a dynamic FAP through dividing this problem into smaller sub-problems, which are then solved in turn in a dynamic process. As TS is the best heuristic algorithm considered in this study (see Chapter 3), it is used to construct DTS. This approach aims to find a feasible solution for each sub-problem with the minimum number of used frequencies. Note that there are no restrictions on the number of re-assigned requests because the static FAP does not concern the number of re-assignments (while the dynamic FAP does). Moreover, we aim to investigate whether applying DTS to solve the static FAP has better performance than solving the entire problem as a whole.

In order to implement this approach, the static FAP needs to be broken down into smaller sub-problems, which can be done as described in Section 6.2. Breaking the static FAP into smaller sub-problems can be viewed as modelling the static FAP into a dynamic FAP. An example of how a static FAP instance is modelled as a dynamic FAP instance is illustrated in Figure 6.12, where each node indicates a request, each edge represents a bidirectional or an interference constraint and each colour reflects a time period in which a request becomes known for the first time.

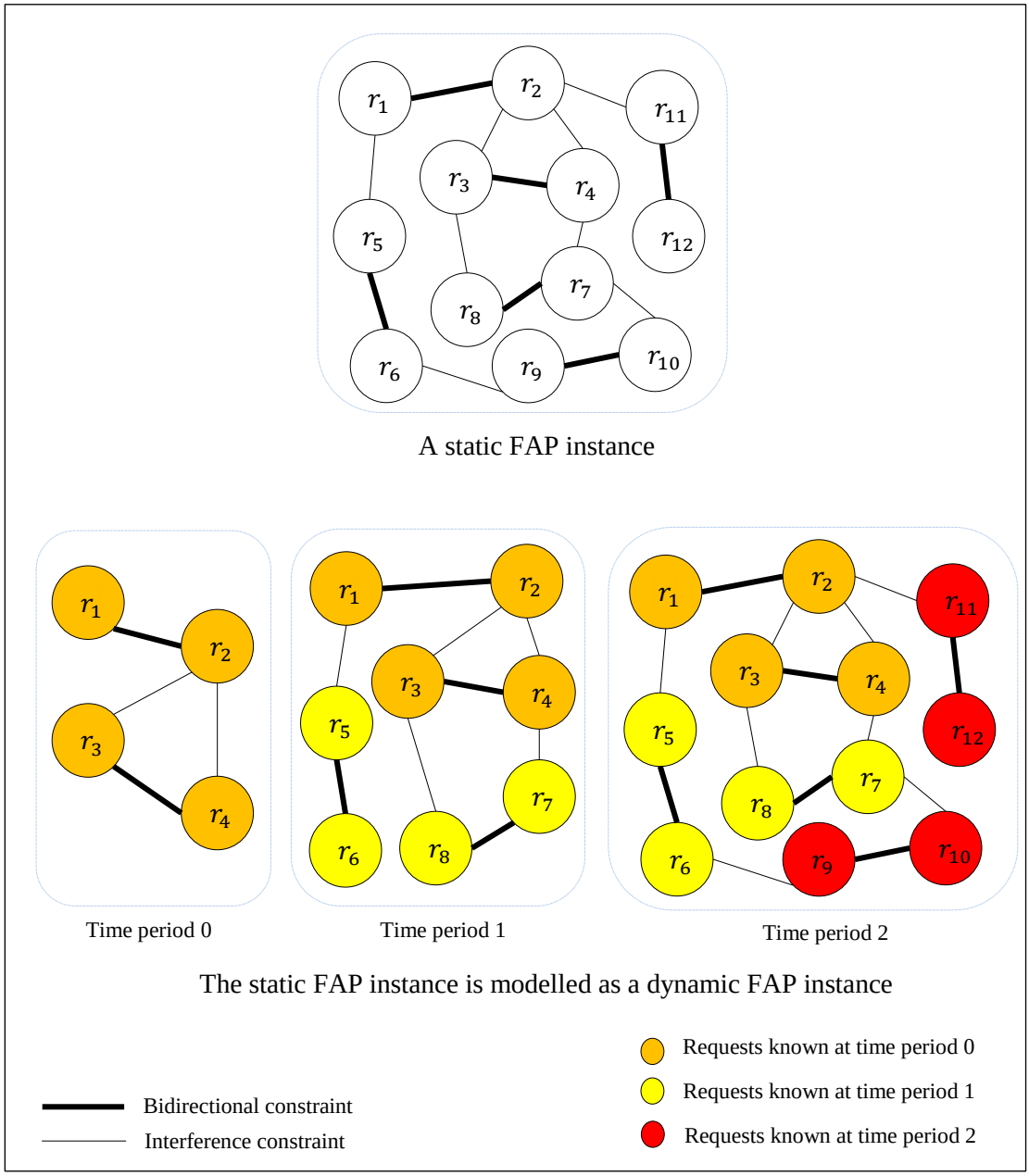

Figure 6.12: An example of modelling a static FAP instance as a dynamic FAP instance over 3 time periods.

6.6.1 Experiments and Results of the DTS Approach

The results of the DTS approach for the static FAP are presented and compared in the following three subsections. The first subsection gives the results of the DTS approach. The second subsection compares the performance of DTS with our TS algorithm. The third subsection compares the performance of DTS with other algorithms in the literature.

6.6.1.1 Results Comparison of the DTS Approach

The results of the DTS approach for the MO-FAP, where each instance is modelled as a dynamic FAP in 10 different versions (based on the number of requests known at time period 0), are given in Table 6.13. Note that a bold number means that the optimal solution was achieved by DTS.

Instance	Number of requests known at time period 0										Optimal Solution
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	
CELAR 01	**16** 2.9 min	**16** 3 min	**16** 4.3 min	**16** 4.5 min	**16** 4.8 min	**16** 4.2 min	18 5.2 min	**16** 5.1 min	**16** 6 min	**16** 6.7 min	16
CELAR 02	**14** 27 sec	**14** 46 sec	**14** 63 sec	**14** 1.1 min	**14** 1.5 min	**14** 1.2 min	**14** 2 min	**14** 1.4 min	**14** 2.3 min	**14** 2.1 min	14
CELAR 03	**14** 51 sec	**14** 1.7 min	**14** 1.6 min	**14** 3.9 min	16 4.3 min	**14** 3.9 min	**14** 4.6 min	**14** 5.3 min	**14** 5.8 min	**14** 5.9 min	14
CELAR 04	**46** 42 sec	**46** 45 sec	**46** 51 sec	**46** 56 sec	**46** 58 sec	**46** 58 sec	**46** 1 min	**46** 1.1 min	**46** 1.3 min	**46** 1.8 min	46
CELAR 11	28 5.2 min	36 7 min	32 6.9 min	30 11 min	28 13 min	32 17 min	32 21 min	36 26 min	32 29 min	32 31 min	22
GRAPH 01	**18** 52 sec	**18** 48 sec	**18** 1.8 min	**18** 4.7 min	**18** 5.1 min	**18** 4.8 min	20 6.9 min	20 6.4 min	20 7.1 min	24 7.4 min	18
GRAPH 02	**14** 41 sec	**14** 1.1 min	**14** 1.9 min	**14** 1.3 min	**14** 2.2 min	**14** 3.6 min	**14** 7.8 min	**14** 6.5 min	**14** 11 min	**14** 16 min	14
GRAPH 08	**18** 1.9 min	**18** 2.2 min	**18** 2.1 min	**18** 3.6 min	**18** 3.8 min	**18** 4.6 min	**18** 4.4 min	**18** 4.8 min	20 4.5 min	**18** 5.5 min	18
GRAPH 09	**18** 3 min	**18** 11 min	**18** 15 min	**18** 17 min	**18** 28 min	**18** 33 min	**18** 41 min	**18** 1.1 hrs	**18** 1.4 hrs	**18** 1.9 hrs	18
GRAPH 14	**8** 20 sec	**8** 2.1 min	**8** 2.1 min	**8** 4.2 min	**8** 6.3 min	**8** 13 min	**8** 19 min	**8** 39 min	**8** 1.1 hrs	**8** 1.8 hrs	8

Table 6.13: Results of the DTS approach for the MO-FAP.

Table 6.13 shows that DTS achieved feasible solutions for all instances. Moreover, this approach achieved the optimal solutions for all instances except CELAR 11. Note that the run time gradually increases with the number of requests known at time period 0. To clarify that, the run times of all versions of CELAR 01, GRAPH 01 and GRAPH 02 are shown in Figure 6.13. The selected instances represent different numbers of requests and constraints.

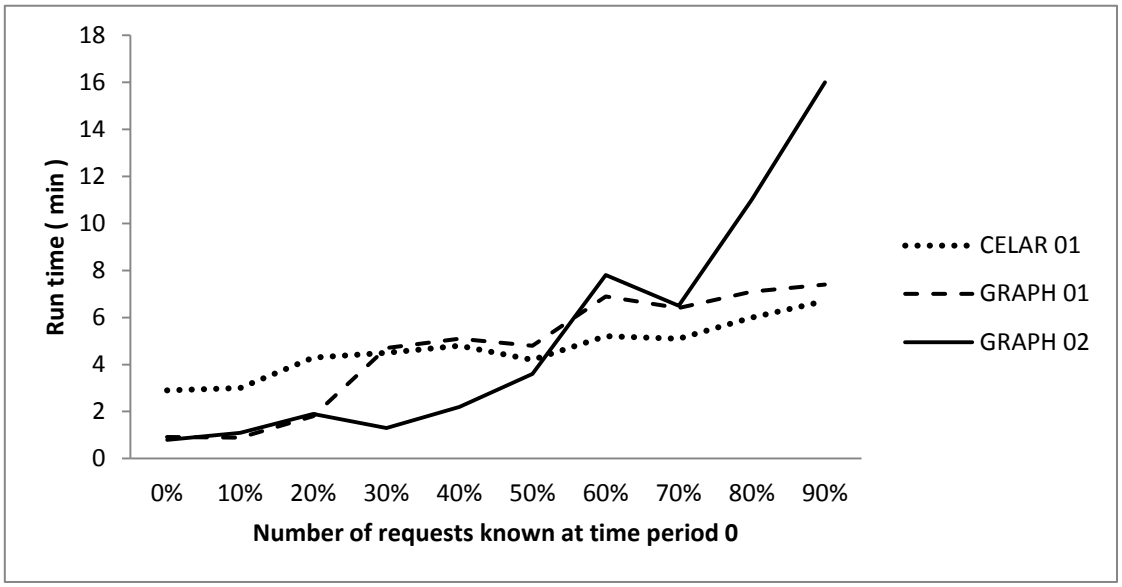

Figure 6.13: The run time of all versions of the selected instances.

Figure 6.13 shows that the run time increased with the number of requests known at time period 0. In terms of both the quality of the solution and the run time, 0% is the best number of requests known at time period 0. As the decision of what percentage of requests are known at time period 0 is not fixed but can be chosen as part of the approach, we choose to apply DTS with no requests being known at time period 0.

6.6.1.2 Results Comparison with the Tabu Search Algorithm

The performance of TS (see Table 3.7) and DTS (see Table 6.13) are shown in Figure 6.14.

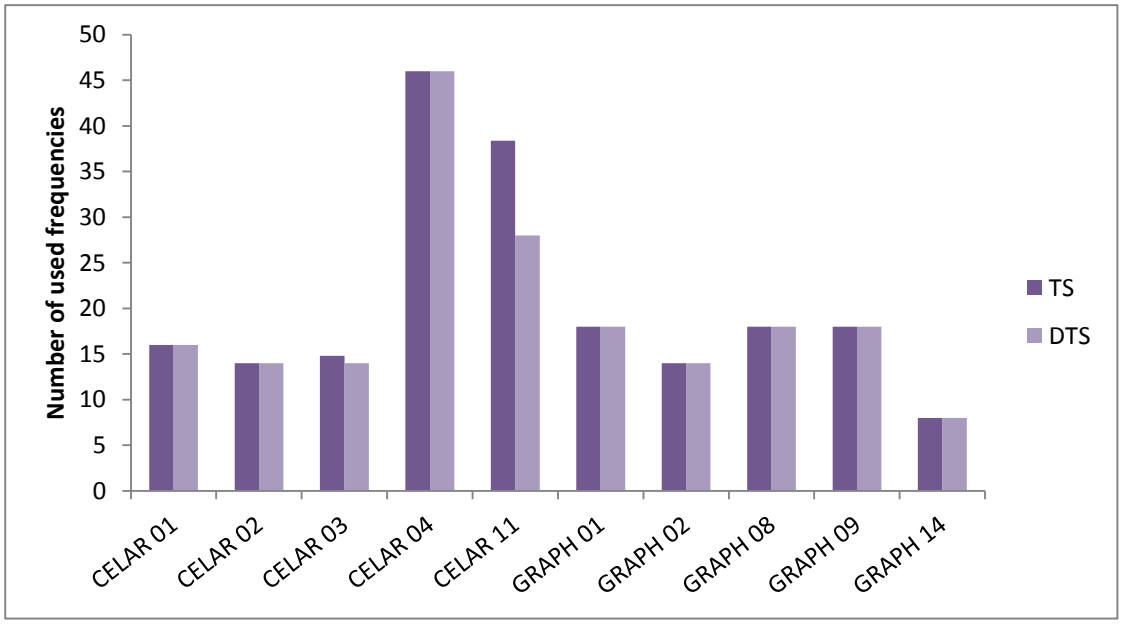

Figure 6.14: The solutions quality of TS and DTS.

Figure 6.14 shows that DTS improved the results of CELAR 03 and CELAR 11, whereas both approaches achieved the optimal solutions in the rest of the instances. Furthermore, the run time of TS and DTS is shown in Figure 6.15.

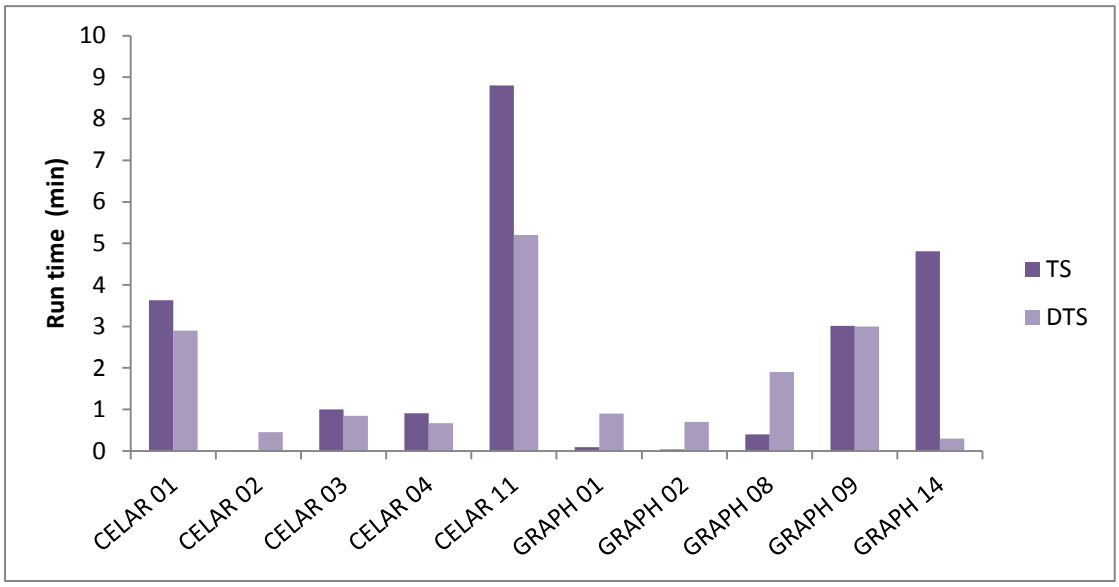

Figure 6.15: The run time of TS and DTS.

Figure 6.15 shows that DTS achieved better run times than TS on CELAR 01, CELAR 03, CELAR 04, CELAR 11 and GRAPH 14. In contrast, TS achieved better run time on the other instances. Furthermore, DTS achieved better total run time (of all instances) than TS as shown in Figure 6.16.

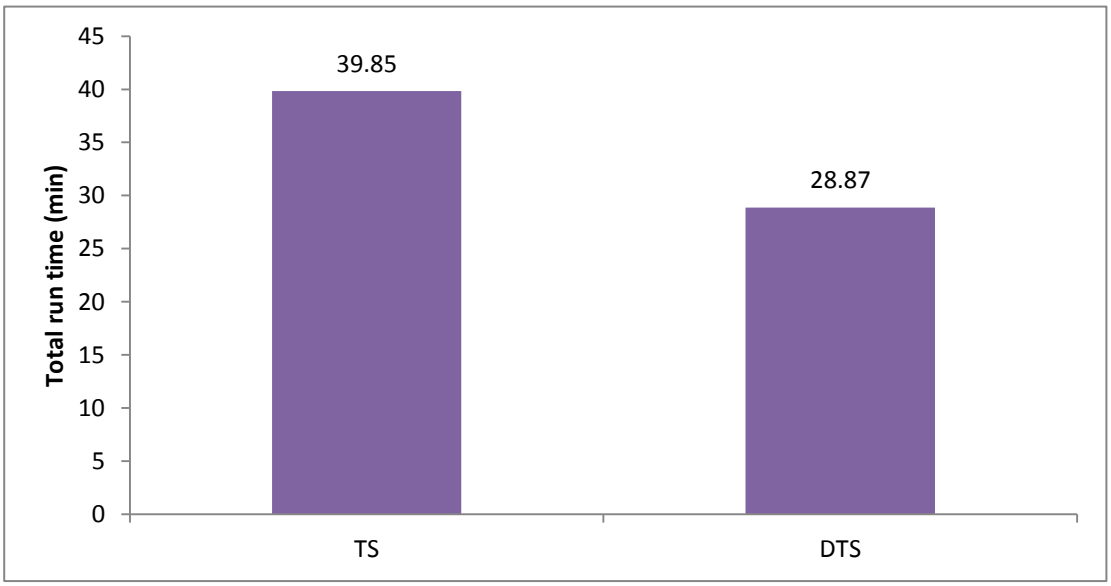

Figure 6.16: The total run time of TS and DTS

Overall, this study suggests that solving the static FAP through modelling it as a dynamic FAP gives competitive results compared with the approach that solves the static FAP as a whole. Hence, this shows the ability of this approach to achieve better performance, which gives a good indication that this should be studied extensively as future work.

6.6.1.3 Results Comparison with Other Algorithms

This section compares the performance of DTS with the algorithms considered in this thesis and other algorithms in the literature. Table 6.14 shows the best found results of these algorithms. Note that a bold number means that the optimal solution was achieved and a dash "-" means that the result is not available.

Instance	GENET [16]	Genetic algorithm [94]	Potential reduction [151]	A nonlinear approach [150]	Evolutionary search [34]	Simulating annealing [145]	Variable depth search [145]	TS [15]	TS [145]	Our TS algorithm	Our ACO algorithm	Our HH algorithm	DTS	Optimal solution
CELAR 01	**16**	20	**16**	**16**	-	**16**	**16**	18	**16**	**16**	18	**16**	**16**	16
CELAR 02	**14**	**14**	**14**	-	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	14
CELAR 03	**14**	16	16	16	**14**	**14**	**14**	**14**	**14**	**14**	16	16	**14**	14
CELAR 04	**46**	**46**	**46**	-	-	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**	46
CELAR 11	24	32	-	-	-	24	24	24	**22**	38	-	36	28	22
GRAPH 01	**18**	20	**18**	**18**	**18**	-	-	**18**	**18**	**18**	20	**18**	**18**	18
GRAPH 02	**14**	16	**14**	**14**	**14**	-	-	16	**14**	**14**	16	**14**	**14**	14
GRAPH 08	22	-	**18**	**18**	-	-	-	24	20	**18**	24	20	**18**	18
GRAPH 09	22	28	**18**	**18**	-	-	-	22	22	**18**	-	20	**18**	18
GRAPH 14	-	14	10	10	-	-	-	12	10	**8**	10	10	**8**	8

Table 6.14: Results of DTS and the algorithms considered in this thesis, and other algorithms in the literature.

Table 6.14 shows that DTS achieved the optimal solution for all the instances except for CELAR 11. Additionally, DTS achieved a better result for CELAR 11 compared with the result which was found by our TS algorithm. Moreover, DTS and our TS algorithm are the only approaches that achieved the optimal solution for GRAPH 14. In contrast, the optimal solution for CELAR 11 was found in [145] using TS. Overall, DTS showed competitive performance compared with the algorithms considered in this thesis and other algorithms in the literature.

6.7 Conclusions

This chapter discussed and compared various approaches for the dynamic FAP, where the best heuristic algorithms considered in this thesis were used to construct these approaches. Several techniques are applied to improve the performance of these approaches. One of these, called the *Gap* technique, is novel. This technique aims to identify a good frequency to be assigned to a given request. For the purpose of this study, new dynamic datasets were generated from the static benchmark datasets (CELAR and GRAPH). Moreover, the new dynamic FAP datasets have been made available for other researchers, which can be found on the dynamic FAP website[1].

These approaches solve the dynamic FAP through solving three underlying problems of the dynamic FAP, which are the static problem, the online problem and the repair problem. Hence, these approaches consist of three solution phases: the initial solution phase (where the requests known at time period 0 are feasibly assigned, if possible), the online assignment phase (where the requests which dynamically arrive are feasibly assigned, if possible) and the repair phase (where the unassigned requests from the previous phases are feasibly assigned, if possible). The repair phase includes two stages: the initial repair phase and the advanced repair phase. Furthermore, the two best heuristic algorithms in this study were implemented and compared as the advanced repair phase, which are TSRP and HHRP. It was found that the best type of advanced repair phase is TSRP. Moreover, the best approach for the dynamic FAP showed competitive performance compared with other approaches in the literature.

Later in this chapter, a novel approach (called DTS) was proposed to solve the MO-FAP, which is a variant of the static FAP, by modelling it as a dynamic FAP through dividing the static FAP into smaller sub-problems. Then, these sub-problems are consecutively solved using TS (which is the best heuristic algorithm in this study) as the advanced repair phase. The proposed approach showed the ability to improve the performance of TS which solves the static FAP as a whole and showed competitive performance compared with other algorithms in the literature.

Finally, the research questions given in the beginning of this chapter can be answered as follows:

[1] https://dynamicfap.wordpress.com/

- *Can TS and HH for the static FAP be successful on the dynamic FAP?*
  TS and HH were shown to successfully tackle the dynamic FAP, where TS achieved the best performance as advanced repair phase (see Section 6.5.2.2).

- *Can the proposed approach that models the static FAP as a dynamic FAP be an effective method for the static FAP?*
  The proposed approach, namely DTS, achieved competitive performance compared with the algorithms considered in this thesis and other algorithms in the literature which solved the static FAP as a whole (see Table 6.14). This suggests that solving the static FAP through modelling it as a dynamic FAP can improve the solutions compared with the solutions which have been found by solving the static problem as a whole. Moreover, this suggests that this approach can be extensively studied as future work.

Chapter 7

Conclusions and Future Work

7.1 Introduction

This thesis considered the frequency assignment problem (FAP), which is related to wireless communication networks. This problem has many applications such as mobile phones, TV broadcasting and Wi-Fi. The aim of the FAP is to assign frequencies to wireless communication connections (also known as requests) while satisfying a set of constraints, which are usually related to prevention of a loss of signal quality. In this thesis, two variants of the FAP were considered, namely the static and the dynamic FAPs.

This thesis consists of two parts. In the first part, three heuristic algorithms, namely tabu search (TS), ant colony optimization (ACO) and hyper heuristic (HH), were designed and developed for the static FAP to identify an appropriate solution method for such problem. In the second part, various approaches for the dynamic FAP were designed using the best performing heuristic algorithms considered in the first part of this thesis. We investigated whether heuristic algorithms which work well on the static FAP also prove efficient on the dynamic FAP. Finally, this thesis proposed a novel

approach to solve the static FAP by modelling it as a dynamic FAP through dividing this problem into smaller sub-problems, which are then solved consecutively based on their time periods.

This chapter is organized as follows: Section 7.2 summarizes the study of the three heuristic algorithms for the static FAP as discussed in Chapters 3, 4 and 5. Section 7.3 summarizes the study of the various approaches for the dynamic FAP and the proposed approach for the static FAP which models it as a dynamic FAP as presented in Chapter 6. Finally, this chapter is closed with some suggestions of future work in Section 7.4.

7.2 Heuristic Algorithms for the Static FAP

This thesis investigated three heuristic algorithms (TS, ACO and HH), which were mainly designed to solve the minimum order FAP (MO-FAP), which is a variant of the static FAP. Each of these represents a different characteristic of heuristic algorithms. TS and ACO represent two different classes of meta-heuristics, where TS represents a class of a local search-based algorithm and ACO represents a construction-based algorithm. HH represents a different characteristic of heuristic algorithms which work at a higher level. HH is based on the idea that each heuristic has strengths and weaknesses, and therefore combining several heuristics may lead to an improved algorithm capable of solving problems with a wide range of characteristics. The selected heuristic algorithms have been successfully implemented on many difficult combinatorial problems in the literature.

Tabu Search Algorithm

In this thesis, several techniques were used to improve the performance of this algorithm and to make it different from existing TS algorithms for the static FAP. One of the novel techniques was applying a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model. These lower bounds were used to ensure that we never waste time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bound of each domain as there is no feasible solution in this search area. Another novel technique was hybridising TS with multiple neighbourhood structures, one of which was used as a diversification technique. In contrast, existing TS algo-

rithms for the static FAP in the literature implemented only a single neighbourhood structure.

Moreover, two different configurations of the TS algorithm were discussed and compared. One configuration relaxes interference constraints, while the other configuration relaxes bidirectional and interference constraints (see Equation 1.1 and 1.2, respectively). In both configurations, the cost function is defined as the number of violations. Relaxing some constraints creates the following sub-problem: minimizing the number of violations with a fixed number of used frequencies.

In this study, the TS algorithm consists of three phases, namely the initial solution phase, the creating violations phase and the improvement phase. This algorithm starts with the initial solution phase to generate an initial solution using a greedy heuristic. If the initial solution is feasible but not optimal, then the creating violations phase is used to produce an infeasible solution that uses fewer frequencies. After that, the improvement phase is used to reduce the number of violations to zero (by solving the sub-problem). If a solution with zero violations, i.e. a feasible solution, is found using this phase, then the number of used frequencies is reduced in the creating violations phase and the sub-problem is reconsidered. The process is repeated until a feasible solution can no longer be found. In case the initial solution is infeasible, the creating violations phase can be omitted and the search moves immediately to the improvement phase.

Based on the experimental results using the CELAR and the GRAPH datasets, it was found that the best approach of TS was based on the first configuration, which relaxes interference constraints. Moreover, this algorithm outperformed other algorithms in the literature. Furthermore, it is noted that applying the same TS algorithm, which has been mainly designed to solve the MO-FAP, on other variants of the static FAP without significant changes was not successful. This finding agrees with what has been found in the literature. It is likely that more significant changes were required for this algorithm to work well on these problems.

Ant Colony Optimization Algorithm

In this thesis, some techniques were used to improve the performance of the ACO algorithm and make it different from other ACO algorithms for the static FAP in the

literature. One of these was applying the concept of a well-known graph colouring algorithm, namely recursive largest first, to improve the process of selecting frequencies and requests to be assigned. Moreover, another technique was applied to improve the trail updates by increasing the level of trail between the unassigned requests and all available frequencies for them to be more likely assigned in the next generation. Furthermore, some of the key factors in producing a high quality ACO implementation are examined such as different definitions of trail ($T_A\ RF$ and $T_A\ RR$) and visibility (based on the number of feasible frequencies and based on the degree), and the optimization of numerous parameters.

In this study, ACO consists of a given number of generations, each of which contains a given number of ants. Each ant starts constructing a solution by selecting a frequency to be assigned to all feasible requests. The process is repeated until no frequencies can be selected. After all ants in the current generation construct their solutions, if no feasible solution can be found, then a local search is used to attempt to achieve a feasible solution. Then, the trail is evaporated and updated. After that, the next generation creates solutions by the same process.

Based on the experimental results using the CELAR and the GRAPH datasets, it was found that the best trail of ACO was based on $T_A\ RF$ and the visibility definition was based on the number of feasible frequencies. ACO did not prove to be as efficient as TS, although this algorithm performed equally well compared with existing ACO algorithms in the literature.

Hyper Heuristic Algorithm

In this thesis, some techniques are applied in HH to make our algorithm more efficient and different from existing HH algorithms for the static FAP. One of the novel techniques was applying a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist, based on the underlying graph colouring model. These lower bounds are used to ensure that we never waste time trying to find a feasible solution with a set of frequencies that do not satisfy the lower bounds, since there is no feasible solution in this search area. Another technique was applying simple and advanced LLHs associated with an independent tabu list for each LLH.

In this study, the HH algorithm consists of three phases, namely the initial solution phase, the creating violations phase and the low level heuristic (LLHs) phase. The first two phases were inherited from our TS algorithm to allow fair comparison. The process of the LLHs phase can be divided into two stages: the LLHs selection mechanism and the move acceptance criteria. One of the LLHs is selected each iteration based on the selection mechanism to find a new solution. Two types the selection mechanism are compared for HH, which are random and probabilistic selection. After that, this solution is accepted or rejected based on the move acceptance criteria, which accepts worse solutions for a limited number of times to diversify the search.

To the best of my knowledge, there are no published papers using HH to solve the static FAP using the datasets considered in this thesis (CELAR and GRAPH). Hence, this is the first attempt to solve such datasets using HH. The experimental results showed that random selection performed better than probabilistic selection. Overall, the performance of HH was superior to ACO and competitive with TS, although generally not of the same standard. However, HH was of sufficient quality to be applied to the dynamic FAP.

To sum up, the best performing heuristic algorithm in this study was TS, with HH also being competitive, whereas ACO achieved poor performance. This suggests that local search-based algorithms are more suitable for solving the static FAP than population-based algorithms and HH algorithms.

7.3 Approaches for Dynamic and Static FAPs

In the dynamic FAP, new requests become known over a period of time and frequencies need to be assigned to those requests effectively and promptly. This problem has received little attention so far in the literature compared with the static FAP. In this thesis, various approaches are designed to solve the dynamic FAP. In order to assess these approaches, new dynamic FAP datasets were generated from the static FAP datasets and have been made available to other researchers on the dynamic FAP website[1]. The objective of the dynamic FAP is to find a feasible solution with the minimum number of re-assigned requests. Changing frequencies which have been as-

[1] https://dynamicfap.wordpress.com/

signed previously is technically allowed. However, in practice this can be time consuming and can take up human resources.

The best heuristic algorithms for the static FAP considered in this thesis were used to construct approaches for the dynamic FAP. This allowed us to investigate whether heuristic algorithms which work well on the static FAP also prove efficient on the dynamic FAP. Additionally, several techniques are applied to improve the performance of these approaches. One of these, called the *Gap* technique, is novel. This technique aims to identify a good frequency to be assigned to a given request. As the dynamic FAP consists of three underlying problems, which are the static problem, the online problem and the repair problem, these approaches consist of three different solution phases. These are the initial solution phase (which aims to feasibly assign requests known at time period 0), the online assignment phase (which aims to feasibly assign requests that arrive dynamically) and the repair phase (which aims to feasibly assign unassigned requests from the previous phase by re-assigning other requests).

The repair phase includes two stages, which are the initial repair phase and the advanced repair phase. Two types of the advanced repair phase were implemented and compared, namely the tabu search repair phase (TSRP) and the hyper heuristic repair phase (HHRP), as these were the best heuristic algorithms for the static FAP in this study. It was found that the best performing approach was based on TSRP as the advanced repair phase. Overall, the performance of the best approach for the dynamic FAP in this study was competitive compared with existing approaches in the literature.

Furthermore, this thesis proposed a novel approach, which is called the dynamic tabu search (DTS) approach, to solve the MO-FAP, which is a variant of the static FAP. This approach modelled the MO-FAP as a dynamic FAP through dividing it into smaller sub-problems, which are then solved consecutively based on their time periods. DTS is inherited from the approach for the dynamic FAP in this study using TS as an advanced repair phase. Several techniques were used to improve the performance of this approach. One of these was applying a lower bound on the number of frequencies that are required from each domain for a feasible solution to exist. Moreover, the *Gap* technique was applied, which aims to identify a good frequency to be assigned to a given request. The proposed approach showed the ability to improve the

results which were found by the heuristic algorithms in this thesis that solve the MO-FAP as a whole. Moreover, it showed competitive performance compared with other algorithms in the literature. This suggests that this could be extensively studied as future work.

7.4 Future Work

The study presented in this thesis leads to many possible directions for future work, either of applying new or existing techniques to the heuristic algorithms considered in this thesis for solving the static and dynamic FAPs. One of the ideas which may improve the performance of TS is applying more advanced neighbourhood structures such as swapping requests based on forming chains similar to Kempe chains in the graph colouring problem (GCP). A Kempe chain can be defined as a set of vertices in two colours that form a chain of clashing vertices with respect to some constraints and if they are all swapped between the two colours then feasibility is maintained. Example 7.1 clarifies the concept of Kempe chains in a GCP.

Example 7.1:

Assume a GCP instance consists of 8 vertices, 2 colours, and 6 edges between these vertices as shown in Figure 7.1.

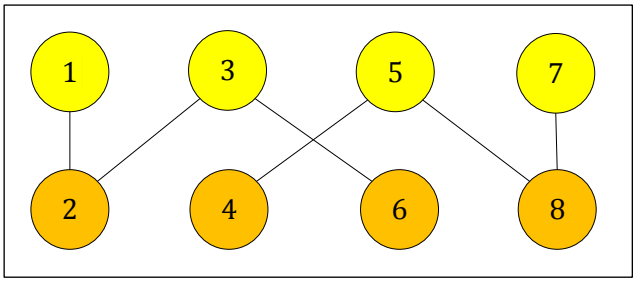

Figure 7.1: A GCP instance in Example 7.1.

Figure 7.1 shows that there are two chains. The first chain includes the vertices 1, 2, 3, and 6, and the second chain includes 4, 5, 8 and 7. The colours of the vertices within each chain can be swapped without any constraint being affected and feasibility is maintained. For example, in the first chain, it is possible to swap the colour of 1 and 3 with the colour of 2 and 6.

When Kempe chains are applied in the static FAP, we have to take into account different types of constraints (see Section 1.4.1), which make feasibility more complicat-

ed to achieve compared with the GCP. In other words, swapping a chain based on one type of constraint may break another type of constraint. Example 7.2 clarifies the difficulties of applying Kempe chains in the static FAP by considering only bidirectional and interference constraints.

Example 7.2:

Assume a static FAP instance consists of 9 requests, 3 frequencies f_1, f_2, and f_3, where $f_1 \leq f_2 \leq f_3$, and a set of interference constraints. Figure 7.2 shows this instance, where each node represents a request, each colour represents a frequency and each edge represents an interference constraint.

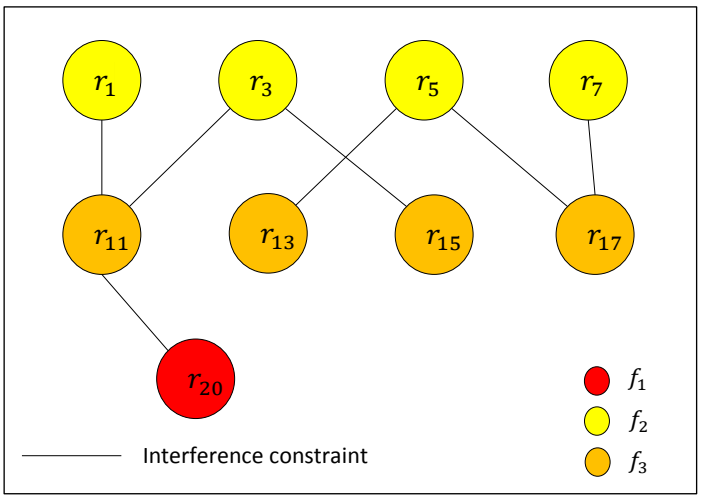

Figure 7.2: The static FAP instance considered in Example 7.2.

Figure 7.2 shows that there is a chain including the requests r_1, r_{11}, r_3 and r_{15}. Note that swapping the frequencies of r_{11} and r_3 may break the interference constraint between r_{11} and r_{20} because $f_2 \leq f_3$. Therefore, we need to take into account the interference constraint which links a request outside the chain to a request in the chain. Furthermore, as each request is linked to its partner by a bidirectional constraint (see Equation 1.1), swapping this chain may also break the bidirectional constraints.

Furthermore, as this thesis introduced interesting techniques to solve the static and dynamic FAPs, it would be worthwhile applying these to solve other problems. Moreover, we could extend our study by including other heuristic algorithms for the static and dynamic FAPs.

Bibliography

1. Aardal, K.I., Hipolito, A., Van Hoesel, C.P.M., Jansen, B., Roos, C. and Terlaky, T., 1996. A branch-and-cut algorithm for the frequency assignment problem. *METEOR, Maastricht research school of Economics of Technology and Organizations*.
2. Aardal, K., Hurkens, C., Lenstra, J.K. and Tiourine, S., 2002. Algorithms for radio link frequency assignment: The CALMA project. *Operations Research, 50*(6), pp.968-980.
3. Aardal, K.I., Van Hoesel, S.P., Koster, A.M., Mannino, C. and Sassano, A., 2003. Models and solution techniques for frequency assignment problems. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, *1*(4), pp.261-317.
4. Aladağ, Ç.H. and Hocaoğlu, G., 2007. A tabu search algorithm to solve a course timetabling problem. *Hacettepe Journal of Mathematics and Statistics*, *36*(1).
5. Al-Sultan, K.S. and Al-Fawzan, M.A., 1999. A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research*, *86*, pp.91-103.
6. Altiparmak, F. and Karaoglan, I., 2007. A genetic ant colony optimization approach for concave cost transportation problems. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pp. 1685-1692. IEEE.
7. Archetti, C., Speranza, M.G. and Hertz, A., 2006. A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science*, *40*(1), pp.64-73.
8. Ayob, M. and Kendall, G., 2003. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies, InTech*, 3, pp. 132-141.
9. Basu, S., 2012. Tabu search implementation on traveling salesman problem and its variations: *a literature survey, American Journal of Operations Research, 2,* pp. 163-173.
10. Bessière, C. and Régin, J.C., 2001. Refining the Basic Constraint Propagation Algorithm. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI*, 1, pp. 309-315.
11. Blum, C., 2005. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, *2*(4), pp.353-373.
12. Blum, C. and Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, *35*(3), pp.268-308.
13. Bonabeau, E., Dorigo, M. and Theraulaz, G., 1999. Swarm intelligence: from natural to artificial systems, 1, *Oxford University press.*

14. Bouhafs, L. and Koukam, A., 2006. A combination of simulated annealing and ant colony system for the capacitated location-routing problem. In *Knowledge-Based Intelligent Information and Engineering Systems*, pp. 409-416. Springer Berlin Heidelberg.
15. Bouju, A., Boyce, J.F., Dimitropoulos, C.H.D., Vom Scheidt, G. and Taylor, J.G., 1995. Tabu search for the radio links frequency assignment problem. *Applied Decision Technologies (ADT'95), London*.
16. Bouju, A., Boyce, J.F., Dimitropoulos, C.H.D., Vom Scheidt, G., Taylor, J.G., Likas, A., Papageorgiou, G. and Stafylopatis, A., 1995, June. Intelligent search for the radio link frequency assignment problem. In *Proceedings of the International Conference on Digital Signal Processing, Cyprus*.
17. Bron, C. and Kerbosch, J., 1973. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM, 16*(9), pp.575-577.
18. Bullnheimer, B., Hartl, R.F. and Strauss, C., 1997. A new rank based version of the Ant System. *A computational study, Central European Journal for Operational Research and Economic, 7*(1), pp.25-38.
19. Bullnheimer, B., Hartl, R.F. and Strauss, C., 1999. An improved ant system algorithm for thevehicle Routing Problem. *Annals of Operations Research, 89*, pp.319-328.
20. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Qu, R., 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society, 64*(12), pp.1695-1724.
21. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S., 2003. Hyper-heuristics: An emerging direction in modern search technology. *International Series in Operations Research and Management Science*, pp.457-474.
22. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J.R., 2010. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pp. 449-468. Springer US.
23. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. and Qu, R., 2009. A survey of hyper-heuristics. *Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham*.
24. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J.R., 2010. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pp. 449-468. Springer US.
25. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R., 2007. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research, 176*(1), pp.177-192.

26. Camazine, S., 2003. *Self-organization in biological systems*. Princeton University Press.
27. Chakhlevitch, K. and Cowling, P., 2008. Hyperheuristics: recent developments. In *Adaptive and Multilevel Metaheuristics*, pp. 3-29. Springer Berlin Heidelberg.
28. Charon, I. and Hudry, O., 1993. The noising method: a new method for combinatorial optimization. *Operations Research Letters*, *14*(3), pp.133-137.
29. Chaves-González, J.M., Vega-Rodríguez, M.A., Gómez-Pulido, J.A. and Sánchez-Pérez, J.M., 2011. Optimizing a realistic large-scale frequency assignment problem using a new parallel evolutionary approach. *Engineering Optimization*, *43*(8), pp.813-842.
30. Chiarandini, M. and Stützle, T., 2007. Stochastic local search algorithms for graph set T-colouring and frequency assignment. *Constraints*, *12*(3), pp.371-403.
31. Coffman, Jr, E.G., Garey, M.R. and Johnson, D.S., 1983. Dynamic bin packing. *SIAM Journal on Computing*, *12*(2), pp.227-258.
32. García, O.C., Triguero, F.H. and Stützle, T., 2002. A review on the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware & Soft Computing*, *9*(3), pp.141-175.
33. Costa, D. and Hertz, A., 1997. Ants can colour graphs. *Journal of the Operational Research Society*, *48*(3), pp.295-305.
34. Crisan, C. and Mühlenbein, H., 1997. The frequency assignment problem: A look at the performance of evolutionary search. In *Artificial Evolution*, pp. 263-273. Springer Berlin Heidelberg.
35. de Werra, D. and Gay, Y., 1994. Chromatic scheduling and frequency assignment. *Discrete Applied Mathematics*, *49*(1), pp.165-174.
36. De Werra, D. and Hertz, A., 1989. Tabu search techniques. *Operations-Research-Spektrum*, *11*(3), pp.131-141.
37. Denzinger, J., Fuchs, M. and Fuchs, M., 1996. *High performance ATP systems by combining several AI methods*. Technische Universität Kaiserslautern, Fachbereich Informatik.
38. Di Caro, G. and Dorigo, M., 1998. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, pp.317-365.
39. Dias, T.M., Ferber, D.F., De Souza, C.C. and Moura, A.V., 2003. Constructing nurse schedules at large hospitals. *International Transactions in Operational Research*, *10*(3), pp.245-265.
40. Dorigo, M., 1992. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*.

41. Dorigo, M. and Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on, 1*(1), pp.53-66.
42. Dorigo, M. and Socha, K., 2006. An introduction to ant colony optimization. *Handbook of approximation algorithms and metaheuristics*, pp.1-26.
43. Dorigo, M. and Stützle, T., 2003. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of metaheuristics*, pp.250-285. Springer US.
44. Dorigo, M., Di Caro, G. and Gambardella, L.M., 1999. Ant algorithms for discrete optimization. *Artificial life*, *5*(2), pp.137-172.
45. Dorigo, M., Maniezzo, V. and Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26*(1), pp.29-41.
46. Dorigo, M., Colorni, A. and Maniezzo, V., 1991. *Ant system: An autocatalytic optimizing process*. Tech. Report 91-016, Politecnico di Milano, Italy.
47. Dorne, R. and Hao, J.K., 1996. Constraint handling in evolutionary search: A case study of the frequency assignment. In *Parallel Problem Solving from Nature—PPSN IV*, pp. 801-810. Springer Berlin Heidelberg.
48. Dowsland, K.A. and Thompson, J.M., 2005. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, pp.426-438.
49. Dowsland, K.A. and Thompson, J.M., 2008. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, *156*(3), pp.313-324.
50. Dror, M. and Trudeau, P., 1989. Savings by split delivery routing. *Transportation Science*, pp.141-145.
51. Dueck, G., 1993. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics, 104*(1), pp.86-92.
52. Dueck, G. and Scheuer, T., 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, *90*(1), pp.161-175.
53. Dumitrescu, I. and Stützle, T., 2003. Combinations of local search and exact algorithms. In *Applications of Evolutionary Computing*, pp. 211-223. Springer Berlin Heidelberg.
54. Dupont, A. and Vasquez, M., 2005. Solving the dynamic frequency assignment problem. In *MIC2005: The Sixth Metaheuristics International Conference, Vienna, Austria*.

55. Dupont, A., Linhares, A.C., Artigues, C., Feillet, D., Michelon, P. and Vasquez, M., 2009. The dynamic frequency assignment problem. *European Journal of Operational Research*, *195*(1), pp.75-88.
56. Einsenblatter, A., Grotschel, M. and Koster, A.M.C.A., 2000. *Frequency assignment and ramifications of coloring*. Technical Report 00-47, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, December 2000.
57. Elkhyari, A., Guéret, C. and Jussien, N., 2004. Constraint programming for dynamic scheduling problems. In *Proceedings of International Scheduling Symposium, Hiroshi Kise, editor*, (04), pp.84-89.
58. Eshghi, K. and Salari, E., 2008. An ACO algorithm for the graph coloring problem. *International Journal of Contemporary Mathematical Sciences*, *3*(6), pp.293-304.
59. Feo, T.A. and Resende, M.G., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), pp.109-133.
60. Fisher, H. and Thompson, G.L., 1963. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*, *3*(2), pp.225-251.
61. Fleurent, C. and Ferland, J.A., 1996. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, *63*(3), pp.437-461.
62. Fogel, D.B., 1995. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence (Piscataway, NJ: IEEE).
63. Galinier, P., Gendreau, M., Soriano, P. and Bisaillon, S., 2005. Solving the frequency assignment problem with polarization by local search and tabu. *4OR*, *3*(1), pp.59-78.
64. Gambardella, L.M. and Dorigo, M., 1997. HAS-SOP: Hybrid ant system for the sequential ordering problem. *Technical Report IDSIA*, pp. 11-97, Lugano, Switzerland.
65. Gambardella, L.M. and Dorigo, M., 1996. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In *International Conference on Evolutionary Computation*, pp. 622-627.
66. Garey, M.R. and Johnson, D.S., 1979. A Guide to the Theory of NP-Completeness. *WH Freemann, New York*.
67. Garrido, P. and Riff, M.C., 2010. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, *16*(6), pp.795-834.
68. Gendreau, M. and Potvin, J.Y., 2010. *Handbook of metaheuristics*, 2. Springer New York.
69. Gendreau, M., Guertin, F., Potvin, J.Y. and Taillard, E., 1999. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, *33*(4), pp.381-390.
70. Glover, F., 1977. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, *8*(1), pp.156-166.

71. Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), pp.533-549.
72. Glover, F., 1989. Tabu search-part I. *ORSA Journal on Computing*, *1*(3), pp.190-206.
73. Glover, F., 1990. Tabu search—part II. *ORSA Journal on Computing*, *2*(1), pp.4-32.
74. Glover, F., 1990. Tabu search: A tutorial. *Interfaces*, *20*(4), pp.74-94.
75. Glover, F. and Laguna, M., 1997. Tabu Search Applications. In *Tabu Search* (pp. 267-303). Springer US.
76. Goldberg, D.E., 1989. *Genetic algorithms in search optimization and machine learning* (Vol. 412). Reading Menlo Park: Addison-wesley.
77. Gonzalez, T.F. ed., 2007. *Handbook of approximation algorithms and metaheuristics*. CRC Press.
78. Gözüpek, D., Genç, G. and Ersoy, C., 2009, September. Channel assignment problem in cellular networks: A reactive tabu search approach. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on* (pp. 298-303). IEEE.
79. Grove, E.F., 1995, Online Bin Packing with Lookahead. In Preceding of the 6th Annual *ACM-SIAM Symposium on Discrete Algorithms*, pp. 430-436. Society for Industrial and Applied Mathematics.
80. Guntsch, M. and Middendorf, M., 2002. Applying population based ACO to dynamic optimization problems. In *Ant Algorithms*, pp. 111-122. Springer Berlin Heidelberg.
81. Gutjahr, W.J., 2003. A generalized convergence result for the graph-based ant system metaheuristic. *Probability in the Engineering and Informational Sciences*, *17*(04), pp.545-569.
82. Gutjahr, W.J., 2004. S-ACO: An ant-based approach to combinatorial optimization under uncertainty. In *Ant Colony Optimization and Swarm Intelligence*, pp. 238-249. Springer Berlin Heidelberg.
83. Hale, W.K., 1980. Frequency Assignment: Theory and applications. *Proceedings of the IEEE*, *68*(12), pp.1497-1514.
84. Halldórsson, M.M. and Szegedy, M., 1992, September. Lower bounds for on-line graph coloring. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 211-216. Society for Industrial and Applied Mathematics.
85. Hao, J.K. and Perrier, L., 1996. Tabu search for the frequency assignment problem in cellular radio networks. Technical Report LG12P, EMA-EERIE, Parc Scientifique Georges Besse, Names France.
86. Hao, J.K., Dorne, R. and Galinier, P., 1998. Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, *4*(1), pp.47-62.
87. Hertz, A. and de Werra, D., 1987. Using tabu search techniques for graph coloring. *Computing*, *39*(4), pp.345-351.

88. Hertz, A. and de Werra, D., 1990. The tabu search metaheuristic: how we used it. *Annals of Mathematics and Artificial Intelligence*, *1*(1), pp.111-121.
89. Hertz, A. and Zufferey, N., 2006. A new ant algorithm for graph coloring. In *Workshop on Nature Inspired Cooperative Strategies for Optimization NICSO*, pp. 51-60.
90. Holder, A., 2006. Mathematical Programming Glossary. INFORMS Computing Society, http://*glossary.computing.society. informs.org/, 2006-2007, originally authored by Harvey Greenberg, 1999-2006*.
91. Holland, J.H., 1975. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.
92. Hopfield, J.J., 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), pp. 2554-2558.
93. Iredi, S., Merkle, D. and Middendorf, M., 2001, March. Bi-criterion optimization with multi colony ant algorithms. In *Evolutionary Multi-Criterion Optimization*, pp. 359-372. Springer Berlin Heidelberg.
94. Kapsalis, A., Chardaire, P., Rayward-Smith, V.J. and Smith, G.D., 1995. The radio link frequency assignment problem: A case study using genetic algorithms. In *Evolutionary Computing*, pp. 117-131. Springer Berlin Heidelberg.
95. Kendall, G. and Hussin, N.M., 2005. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary Scheduling: Theory and Applications*, pp. 309-328. Springer US.
96. Kendall, G. and Mohamad, M., 2004. Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, 2, pp. 769-773. IEEE.
97. Kendall, G. and Mohamad, M., 2004. Channel assignment optimisation using a hyper-heuristic. In *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, 2, pp. 791-796. IEEE.
98. Kirkpatrick, S. Gelatt, C. D. and Vecchi, M. P., 1983. Optimization by simulated annealing. *Science*, 4220, pp. 671–680.
99. Knuth, D.E., 1976. Big omicron and big omega and big theta. *ACM Sigact News*, *8*(2), pp.18-24.
100. Koster, A.M.C.A., 1999. *Frequency assignment: Models and algorithms*. Arie Koster.
101. Koster, A.M., van Hoesel, S.P. and Kolen, A.W., 1999, June. Optimal solutions for frequency assignment problems via tree decomposition. In *Graph-theoretic concepts in computer science*, pp. 338-350. Springer Berlin Heidelberg.

102. Kouvelis, P. and Yu, G., Robust discrete optimization and its applications. 1997. *Kluwer Academic Publishers, Boston.*
103. Kunz, D., 1991. Channel assignment for cellular radio using neural networks. *Vehicular Technology, IEEE Transactions on*, *40*(1), pp.188-193.
104. Leighton, F.T., 1979. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, *84*(6), pp.489-506.
105. Likas, A. and Stafylopatis, A., 1996. Group updates and multiscaling: An efficient neural network approach to combinatorial optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *26*(2), pp.222-232.
106. Linhares, A.C., Michelon, P. and Feillet, D., 2010. An exact site availability approach to modeling the D-FAP. *Electronic Notes in Discrete Mathematics*, *36*, pp.1-8.
107. Luna, F., Blum, C., Alba, E. and Nebro, A.J., 2007. ACO vs EAs for solving a real-world frequency assignment problem in GSM networks. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 94-101.
108. Maniezzo, V., 1999. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, *11*(4), pp.358-369.
109. Maniezzo, V. and Carbonaro, A., 2000. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, *16*(8), pp.927-935.
110. Maximiano, M.D.S., Vega-Rodriguez, M.A., Gomez-Pulido, J.A. and Sánchez-Pérez, J.M., 2009, December. Multiobjective frequency assignment problem using the MO-VNS and MO-SVNS algorithms. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pp. 221-226. IEEE.
111. Metzger, B.H., 1970. Spectrum Management Technique presented at 38th National ORSA meeting. *Detroit, MI (Fall 1970).*
112. Mladenović, N. and Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research*, *24*(11), pp.1097-1100.
113. Mohr, R. and Henderson, T.C., 1986. Arc and path consistency revisited. *Artificial Intelligence*, *28*(2), pp.225-233.
114. Montemanni, R. and Smith, D.H., 2010. Heuristic manipulation, tabu search and frequency assignment. *Computers & Operations Research*, *37*(3), pp.543-551.
115. Montemanni, R., Smith, D.H. and Allen, S.M., 2002. An ANTS algorithm for the minimum-span frequency-assignment problem with multiple interference. *Vehicular Technology, IEEE Transactions on*, *51*(5), pp.949-953.
116. Nowicki, E. and Smutnicki, C., 1996. A fast taboo search algorithm for the job shop problem. *Management Science*, *42*(6), pp.797-813.

117. Nowicki, E. and Smutnicki, C., 2005. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, *8*(2), pp.145-159.
118. OFCOM, 2013. Annual licence fees for 900 MHz and 1800 MHz spectrum consultation, United Kingdom Frequency Allocation Table, Issue No. 17
119. Osman, I.H., 2003. Focused issue on applied meta-heuristics. *Computers & Industrial Engineering*, *44*(2), pp.205-207.
120. Osman, I.H. and Kelly, J.P. eds., 2012. *Meta-heuristics: Theory and Applications*. Springer Science & Business Media.
121. Ouelhadj, D. and Petrovic, S., 2009. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, *12*(4), pp.417-431.
122. Özcan, E., Bilgin, B. and Korkmaz, E.E., 2008. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, *12*(1), pp.3-23.
123. Park, T. and Lee, C.Y., 1996. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research Society of Japan*, *39*(2), pp.258-265.
124. Parsapoor, M. and Bilstrup, U., 2013. Ant colony optimization for channel assignment problem in a clustered mobile ad hoc network. In *Advances in Swarm Intelligence*, pp. 314-322. Springer Berlin Heidelberg.
125. Pirim, H., Eksioglu, B. and Bayraktar, E., 2008. *Tabu Search: a comparative study* (pp. 1-27). INTECH Open Access Publisher.
126. Plumettaz, M., Schindl, D. and Zufferey, N., 2010. Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, *61*(5), pp.819-826.
127. Pour, H.D. and Nosraty, M., 2006. Solving the facility and layout and location problem by ant-colony optimization-meta heuristic. *International Journal of Production Research*, *44*(23), pp.5187-5196.
128. Psaraftis, H.N., 1988. Vehicle routing: Methods and studies. *Dynamic Vehicle Routing Problems. North Holland, Amsterdam, The Netherlands*, pp.223-248.
129. Rattadilok, P., Gaw, A. and Kwan, R.S., 2004. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Practice and Theory of Automated Timetabling V*, pp. 51-67. Springer Berlin Heidelberg.
130. Ribeiro, C.C. and Hansen, P. eds., 2012. *Essays and surveys in metaheuristics*, 15. Springer Science & Business Media.
131. Roberts, F.S., 1991. T-colorings of graphs: recent results and open problems. *Discrete mathematics*, *93*(2-3), pp.229-245.
132. Ross, P., 2005. Hyper-heuristics. In *Search methodologies*, pp. 529-556. Springer US.

133. Rubinstein, R.Y., 2001. Combinatorial optimization via the simulated cross-entropy method. *Encyclopedia of Operations Research and Management Science. Boston, Kluwer Academic Publishers*.
134. Schulz, M. and Eisenblätter, A., 2003. Solving Frequency Assignment Problems with Constraint Programming. *Technische Universität Berlin, Institut für Mathematik*.
135. Segura, C., Miranda, G. and León, C., 2011. Parallel hyperheuristics for the frequency assignment problem. *Memetic Computing*, *3*(1), pp.33-49.
136. Skiena, S.S., 1998. *The algorithm design manual: Text*, 1. Springer Science & Business Media.
137. Smith, D.H., Allen, S.M., Hurley, S. and Watkins, W.J., 1998, October. Frequency assignment: Methods and algorithms. In *Proceedings of the NATO RTA SET/ISET symposium on frequency assignment, sharing and conservation in systems (aerospace), Aalborg, Denmark*, pp. K1-K18.
138. Smith, K. and Palaniswami, M., 1997. Static and dynamic channel assignment using neural networks. *Selected Areas in Communications, IEEE Journal on*, *15*(2), pp.238-249.
139. Socha, K., 2004. ACO for continuous and mixed-variable optimization. In *Ant Colony Optimization and Swarm Intelligence*, pp. 25-36. Springer Berlin Heidelberg.
140. Socha, K. and Dorigo, M., 2008. Ant colony optimization for continuous domains. *European Journal of Operational Research*, *185*(3), pp.1155-1173.
141. Sörensen, K. and Glover, F.W., 2013. Metaheuristics. In *Encyclopedia of Operations Research and Management Science*, pp. 960-970. Springer US.
142. Stützle, T. and Dorigo, M., 1999. ACO algorithms for the traveling salesman problem. *Evolutionary Algorithms in Engineering and Computer Science*, pp.163-183.
143. Stützle, T. and Hoos, H.H., 2000. MAX–MIN ant system. *Future Generation Computer Systems*, *16*(8), pp.889-914.
144. Talbi, E.G., 2009. *Metaheuristics: from design to implementation*, 74. John Wiley & Sons.
145. Tiourine, S.R., Hurkens, C.A.J. and Lenstra, J.K., 2000. Local search algorithms for the radio link frequency assignment problem. *Telecommunication Systems*, *13*(2-4), pp.293-314.
146. Van Benthem, H.P., 1995. GRAPH-Generating Radio link frequency Assignment Problems Heuristically. *Master thesis, Delft University of Technology*.
147. Vasquez, M., Dupont, A. and Habet, D., 2005. Consistent neighbourhood in a tabu search. In *Metaheuristics: Progress as Real Problem Solvers*, pp. 369-388. Springer US.

148. Vishwanathan, S., 1992. Randomized online graph coloring. *Journal of Algorithms*, *13*(4), pp.657-669.
149. Voß, S., Martello, S., Osman, I.H. and Roucairol, C. eds., 2012. *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Springer Science & Business Media.
150. Warners, J.P., 1998. A nonlinear approach to a class of combinatorial optimization problems. *Statistica neerlandica*, *52*(2), pp.162-184.
151. Warners, J.P., Terlaky, T., Roos, C. and Jansen, B., 1997. A potential reduction approach to the frequency assignment problem. *Discrete Applied Mathematics*, *78*(1), pp.251-282.
152. Watkins, W.J., Hurley, S. and Smith, D.H., 1998. Evaluation of models for area coverage. *Report to UK Radiocommunications Agency, Department of Computer Science, Cardiff University*.
153. White, G.M. and Xie, B.S., 2000. Examination timetables and tabu search with longer-term memory. In *Practice and Theory of Automated Timetabling III*, pp.85-103. Springer Berlin Heidelberg.
154. Wu Q., Zhang J.and Xu X., 1999. Variability of ant colony algorithm. Computer Research and Development, 36(10).
155. Yu-Bin, Z., Yu-Cai, Z. and Hui, X., 2009, September. A tabu search algorithm for frequency assignment problem in wireless communication networks. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*, pp.1-4. IEEE.