UNITED KINGDOM · CHINA · MALAYSIA

University of Nottingham

Doctoral Thesis

Machine Learning for Improving Heuristic Optimisation

Author:
Shahriar Asta

Supervisor:
Dr. Ender Özcan

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Automated Scheduling, Optimisation and Planning (ASAP) Research Group
School of Computer Science

September 2015

Abstract

Heuristics, metaheuristics and hyper-heuristics are search methodologies which have been preferred by many researchers and practitioners for solving computationally hard combinatorial optimisation problems, whenever the exact methods fail to produce high quality solutions in a reasonable amount of time. In this thesis, we introduce an advanced machine learning technique, namely, tensor analysis, into the field of heuristic optimisation. We show how the relevant data should be collected in tensorial form, analyzed and used during the search process. Four case studies are presented to illustrate the capability of single and multi-episode tensor analysis processing data with high and low abstraction levels for improving heuristic optimisation. A single episode tensor analysis using data at a high abstraction level is employed to improve an iterated multi-stage hyper-heuristic for cross-domain heuristic search. The empirical results across six different problem domains from a hyper-heuristic benchmark show that significant overall performance improvement is possible. A similar approach embedding a multi-episode tensor analysis is applied to the nurse rostering problem and evaluated on a benchmark of a diverse collection of instances, obtained from different hospitals across the world. The empirical results indicate the success of the tensor-based hyper-heuristic, improving upon the best-known solutions for four particular instances. Genetic algorithm is a nature inspired metaheuristic which uses a population of multiple interacting solutions during the search. Mutation is the key variation operator in a genetic algorithm and adjusts the diversity in a population throughout the evolutionary process. Often, a fixed mutation probability is used to perturb the value at each *locus*, representing a unique component of a given solution. A single episode tensor analysis using data with a low abstraction level is applied to an online bin packing problem, generating locus dependent mutation probabilities. The tensor approach improves the performance of a standard genetic algorithm on almost all instances, significantly. A multi-episode tensor analysis using data with a low abstraction level is embedded into multi-agent cooperative search approach. The empirical results once again show the success of the proposed approach on a benchmark of flow shop problem instances as compared to the approach which does not make use of tensor analysis. The tensor analysis can handle the data with different levels of abstraction leading to a learning approach which can be used within different types of heuristic optimisation methods based on different underlying design philosophies, indeed improving their overall performance.

Acknowledgements

I am deeply grateful to Dr. Ender Özcan, my supervisor who generously rendered help and encouragement during the process of this thesis. Whatever I have accomplished in pursuing this undertaking is due to his guidance and thoughtful advice. To him my thanks.

I would like to thank the examiners, Professor Robert John and Professor Shengxiang Yang, for their valuable comments.

My greatest debt is, as always, to my parents and my lovely wife who persistently rendered me the much needed support in my academic endeavours.

Contents

List of Figures

List of Tables

Acronyms

AdapHH Adaptive Hyper-Heuristic

AL Apprenticeship Learning

ALS Alternating Least Square

CHAMP Creating Heuristics viA Many Parameters

CHeSC Cross-domain Heuristic Search Challenge

CP Decomposition Canonical Polyadic Decomposition

HyFlex Hyper-heuristic Flexible framework

IE Improving or Equal

LS Local Search heuristic

MACS Multi-Agent Cooperative Search

MU Mutation heuristic

NA Naïve Acceptance

RR Ruin-Recreate heuristic

SRNA Simple Random heuristic selection with Naïve Acceptance strategy

SRIE Simple Random heuristic selection with Improving or Equal acceptance strategy

TB-MACS Tensor-Based Multi-Agent Cooperative Search

TeBHH Tensor-Based Hyper-Heuristic

TeBHA-HH Tensor-Based Hybrid Acceptance Hyper-Heuristic

XO Crossover heuristic

Chapter 1

Introduction

Heuristics have been used as rule-of-thumb approaches to solve computationally difficult optimisation problems, since exact methods often fail to produce solutions with a *reasonable* quality in a em reasonable amount of time. Throughout the time, numerous heuristics have been designed and successfully applied to particular problems. It has been observed that different heuristics have different performances on different problem domains or even on different instances from the same problem domain.

Metaheuristics are search methodologies that provide guidelines for heuristic optimisation [8]. Once implemented and tailored for a specific problem domain or even a specific class of instances in a domain, metaheuristics similar to heuristics often cannot be reused and applied to another domain or even a different class of instances in some cases. In other words, they often lack generality. Another issue is that they come with a set of parameters which often influences their performance, hence they require tuning which is a time consuming and costly process. Generality, re-usability, simplicity and solution quality are among the few peculiarities required in a powerful high level search method. In order to achieve a higher level of generality, automated intelligent search methodologies, namely *hyper-heuristics* have emerged [9–17]. High level hyper-heuristics operate on the space of low level heuristics which operate on the solutions. This indirect operation is usually achieved by devising a method to control/manage/mix or even generate low level heuristics through a *domain barrier* (Figure 1.1). No problem domain specific information flow is allowed from the domain level to the hyper-heuristic level during the whole search process. Hyper-heuristics can either select/manage a set of fixed heuristics or generate new heuristics to solve a given problem. The former is referred to as *selection hyper-heuristics* and the latter as *generation hyper-heuristics*. Also, depending on the way they handle the feedback from the search process, they can be grouped into *learning* and *no learning* methods [13]. A selection hyper-heuristic has two main components:

heuristic selection method and *move acceptance method*. The heuristic selection method decides which low level heuristic to apply to the solution in hand at each search step, creating a new solution. Then the move acceptance method is used to decide whether the old solution should be replaced by the new one or not.

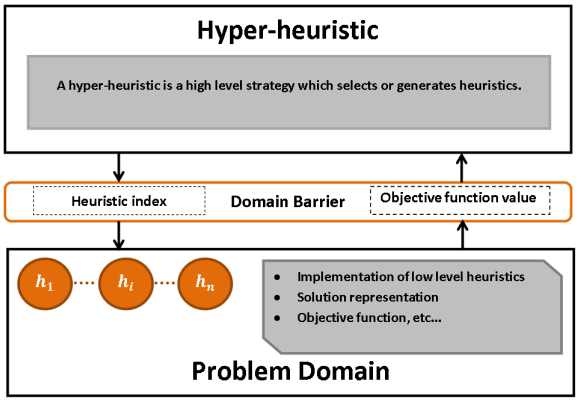


FIGURE 1.1: Domain barrier in hyper-heuristics [1].

Ideally, hyper-heuristics are designed to be general in the area of application, simple in design and off-the-shelf when it comes to re-usability. Furthermore, to increase their generality, they incorporate automated features which enables them to adapt to the difficulties of uncharted territory. This being the ideal case, there is still a huge chasm between where hyper-heuristics stand today and the ideal point. This is not to say that hyper-heuristics have achieved little. These algorithms have met some of the expectations with respect to generality. However, much needs to be done and a long way should be covered in order to bring hyper-heuristics to an ideal point. Improvement in generality and re-usability without compromising the quality of solutions achieved is imperative if one's goal is achieving the ideal hyper-heuristic.

One way to deal with the problem of generality in (hyper-/meta) heuristics is to consider using machine learning techniques. Machine learning is the science of learning useful rules and recognising hidden patterns from example data [18, 19]. These examples are either presented as data or are acquired from direct interaction with the environment, leading to offline and online variants of machine learning algorithms respectively. Offline machine learning techniques use training algorithms to mine the data for hidden patterns. This is while online algorithms build up a generalized model gradually as they interact with the environment they are inhabiting. Furthermore, supervised and unsupervised training approaches are possible. In supervised machine learning, example data is accompanied by the desired output for each decision point. The desired output is usually provided by a human expert which is why the approach is called supervised learning. In case such supervision is missing, the task of learning is un-supervised. The patterns/rules discovered by machine learning algorithms are powerful in describing the

data. They can be used as state-action mappers by choosing the right action in various unseen decision points. They can also be used as classifiers to group new arriving data to their respective classes. The predictive power of machine learning techniques has enabled researchers to solve a variety of challenging problems such as face recognition [20], natural language processing [21] and bioinformatics [22] among others.

Search algorithms can greatly benefit from machine learning approaches and their generalisation power. Patterns extracted by machine learning algorithms can be used to further refine the operations of the search algorithm and improve it's performance. Since machine learning approaches build generalizable models of the data provided to them, search algorithms can use this general model and apply it to a wider range of problems and their instances. This way, one could expect higher levels of generality from the search algorithms. Different ways exist to combine search methods and machine learning approaches. In cases where data can be gathered from the performance of the search algorithm, this data can be presented in supervised or un-supervised form to the machine learning algorithm. The learning procedure can then extract hidden patterns and useful rules and pass it over to the search algorithm which in turn uses this information to refine its operations. One such cycle is referred to as a *learning episode*. Repeating this cycle leads to a *multiple episode* learning system.

Combining search algorithms and machine learning can also be approached from a memetic computing point of view. Memetic computing is an umbrella term for algorithms with various components where there is interaction between the components [23, 24]. The notion of memes as a unit of cultural transmission is interpreted as search strategy in Memetic Algorithms [25]. A learning (hyper-/meta) heuristic can also be perceived as a memetic computing algorithm. Considering various components of a selection hyper-heuristic, they constantly interact with one another while the learning mechanism uses these interactions to continuously generate new guidelines using which the algorithm improves its performance. Memetic computing is a general term and is not exclusive to population-based approaches where a pool of candidate solutions is used during the search. In fact, efficient single point memetic computing algorithms which use one candidate solution only have been introduced [24, 26, 27].

Data abstraction level can have a determining impact on the performance of a machine learning algorithm both in terms of accuracy and expressiveness. Data is regarded as highly abstract whenever it has a simple structure and/or the amount of detail it contains is small. Conversely, the presence of complexities, concrete structures and a high amount of details in data decreases it's abstraction level. Such data is considered to have low level of abstraction [28]. Depending on the level of abstraction in data, the performance of a machine learning algorithm may vary. Highly abstract data are

often easy to model, though, the model is not necessarily very accurate. The accuracy in prediction depends on what features the data contains and whether these features have good discriminative properties. The output of a given machine learning method on abstract data is also expected to be fairly understandable. Data with low abstraction level is hard to model, however, more complexity in such data leads to more general models with high predictive power given a proper choice of the learning technique and efficient training procedure. The output of the learning algorithm is also expected to be complex and thus hard to interpret. This is not to say that abstract data are useless or that one should only consider non-abstract data for learning purposes. On the contrary, abstract data can be very useful whence the data features are carefully designed and the data is properly collected.

Different search algorithms utilize different design paradigms. Some algorithms (such as hyper-heuristics) are high level methods and use abstract information to perform the search. Some hyper-heuristic frameworks use a conceptual domain barrier. This domain barrier restricts the flow of information from the search space to the high level strategy. Often, this information only contains the objective function value along with the indices and types of low level heuristics available for a given problem domain. Thus, the information using which the hyper-heuristics performs the search are minimal. Consequently, the trace the hyper-heuristic leaves behind also contains minimal information and can be regarded as highly abstract data. There are also hyper-heuristic algorithms which do not use the domain barrier concept [29]. These hyper-heuristics re-formulate the representation of the candidate solution in form of heuristics. Thus, low level heuristics in these cases have more information on the solution space compared to the hyper-heuristics which use the domain barrier. However, they still don't deal with the solution space directly. Naturally, the trace left behind by these hyper-heuristics is less abstract. Finally, there are domain specific metaheuristics which search directly in the solution space. The data provided by these algorithms is rich as it contains the solutions themselves and is not abstract. Abstraction levels in data produced by various algorithms using different design perspectives is illustrated in Figure 1.2(a).

Thus, while embedding machine learning algorithms in search methodologies, one needs to consider certain criteria. The machine learning should be able to extract useful patterns from various domains with very different natures. The models generated by the learning technique should have a reasonably high level of generality and reduce the frequency of training for new and unseen problem instances/domains. It also needs to improve upon the performance of the heuristic it is attached to. Finally, the machine learning approach should be able to deal with various levels of data abstraction if it would be considered as a fitting learning technique for a wide range of search and optimisation algorithms.

1.1 Research Motivation and Contributions

Using machine learning techniques to improve the performance of search algorithms is not a new strategy. Several studies have been conducted on the role of machine learning in improving the performance of search algorithms [30–35]. However, they suffer from few drawbacks and usually fail to satisfy crucial criteria. Some of these algorithms lack sufficient generality, high performance, agility and in some cases originality and novelty. Working on domain independent data and operating on data with different levels of abstraction are also important issues which are usually ignored.

In this study, we introduce an advanced machine learning technique, namely, tensor analysis, into the field of heuristic optimisation. This is the first time that tensor analysis is used in heuristic optimisation. Tensor analysis approaches use data in form of multi-dimensional arrays and collecting data in this form can sometimes be difficult. The data discussed here is collected from the search history produced by running a given (hyper-/meta) heuristic for a short amount of time. In this study, we show how this trace data is collected in a tensorial form. We also provide guidelines as to how to analyze this data and interpret the results. Moreover, we show how to use the results and embed the tensor analysis approach in various heuristic optimisation algorithms. To show the efficiency of the proposed approach, different case studies with different optimisation methods has been considered in our experiments. Both single and multiple learning episodes have been considered with data of different levels of abstraction to show that the proposed approach is capable of producing good quality results in different conditions. Figure 1.2(b) shows the relation between different approaches proposed in this study and the data abstraction level. The circles beside each method is the order in which they will be described later.

A single episode tensor analysis using data with a high abstraction level is utilized to improve a multi-stage hyper-heuristic for cross-domain heuristic search. The empirical results using the Hyper-heuristic Flexible (HyFlex) framework [36] on six different problem domains show that significant performance improvement is possible [37]. The problem domains in HyFlex include Bin Packing (BP), Satisfiability (Max-SAT), Personnel Scheduling (PS), Flow-shop Scheduling (FS), Vehicle Routing Problem (VRP) and the Travelling Salesman Problem (TSP). Our results suggest that the proposed approach is powerful and capable of producing high quality solutions. Interestingly, it was observed that, tensor analysis, when interpreted correctly, transforms the underlying hyper-heuristic into an iterated local search algorithm [38–40] where the candidate solution is periodically subjected to intensification followed by diversification. This can also be seen as a memetic computing approach in which a good analysis on the correlation

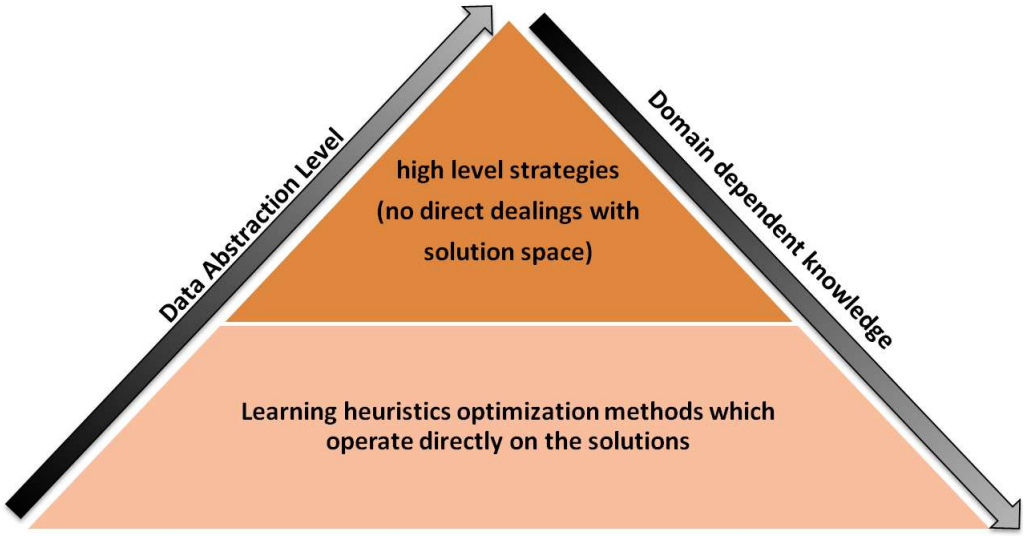
between low level heuristics has been made and a good balance between intensifying and diversifying operations is achieved.

Encouraged by the previous results, in order to investigate whether the proposed approach is capable of extracting useful pattern continuously, we shifted our focus towards the multi-episode performance of the proposed approach. A similar approach embedding a multi-episode tensor analysis is applied to the nurse rostering problem. This approach is evaluated on a well-known nurse rostering benchmark consisting of a diverse collection of instances obtained from different hospitals across the world. The empirical results indicate the success of the tensor-based hyper-heuristic, improving upon the best-known solutions for four particular instances.

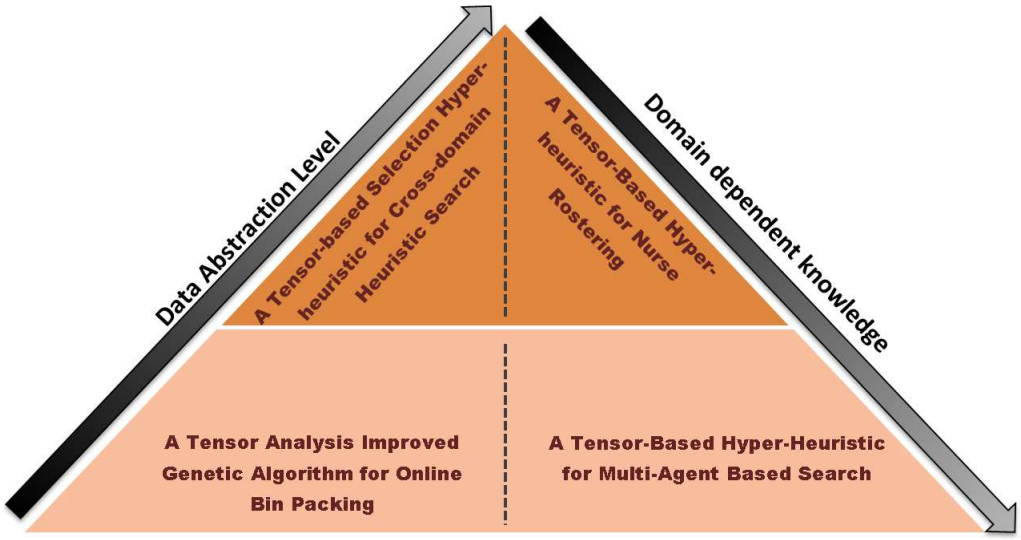
Moving to lower levels of data abstraction, the tensor analysis approach is embedded in a genetic algorithm framework. Genetic algorithm is a well-known population-based metaheuristic, inspired from biological evolution, which uses multiple candidate solutions during the search process. Mutation in a genetic algorithm is the key variation operator adjusting the diversity in a population throughout the evolutionary process. Often, a fixed mutation probability is used to perturb the value of a gene (locus), representing a component of a solution. The genetic algorithm framework considered here is a hyper-heuristic algorithm which dismisses the usage of the domain barrier concept. Instead, the framework re-formulates the representation of the candidate solutions in form of ranking heuristics. Therefore, the data provided to the tensor analysis approach has a lower abstraction level compared to the data acquired from the HyFlex framework. However, the level of abstraction is higher than metaheuristics as the genetic algorithm still doesn't search in the solution space directly. A multiple episode tensor analysis using data with a low abstraction level is applied to an online bin packing problem, generating locus dependent mutation probabilities. The empirical results show that the tensor approach improves the performance of a standard Genetic Algorithm on almost all instances, significantly [41].

Finally, a multi-episode tensor analysis using data with a low abstraction level is embedded into multi-agent cooperative search approach. Unlike former optimisation frameworks considered earlier, the multi-agent approach searches directly in the solution space providing data with the lowest level of abstraction possible. The empirical results once again shows the success of the proposed approach on a benchmark of flow shop problem instances.

As a conclusion, the tensor analysis approach is powerful in that it generates high quality solutions. It also is capable of operating on a wide range of problems, exhibiting cross-domain performance and requires short training time. Moreover, it can handle data from various ends of the abstraction spectrum leading to an approach that can be embedded



(a)



(b)

FIGURE 1.2: (a) algorithms with different underlying design philosophies produce data with different levels of abstraction (b) the machine learning methodology proposed in this study has been integrated to various heuristic optimisation methods and therefore on data with different abstraction levels.

within different types of heuristic optimisation methods with different underlying design philosophies and indeed improve their performance (Figure 1.2(b)).

1.2 Structure of Thesis

The thesis is structured as follows:

- **Chapter 1** introduces the thesis topic and relevant concepts.

- **Chapter 2** provides the background and literature survey. A detailed discussion on metaheuristics and hyper-heuristics is provided in this section. Also, since the methodology used here heavily relies on machine learning, basic data science concepts and details of advanced methods used in this work are covered in this section.

- **Chapter 3** introduces a tensor-based hyper-heuristic for cross-domain heuristic search. An extensive set of experiments has been conducted across thirty problem instances from six different domains of a benchmark. A report of the results from those experiments along with analytic discussions regarding the proposed approach is given in this section.

- **Chapter 4** investigates the usefulness of multiple learning episodes at each run of a heuristic optimisation method, unlike the previously proposed approach which uses a single learning episode. Can the proposed approach extract new patterns in subsequent episodes? An extensive discussion on the experimental results, described in this section, reflects on these questions and shows that the proposed approach is indeed powerful and capable of pattern detection as the search state changes.

- **Chapter 5** Unlike the studies in Chapters 3 and 4 where the focus was on selection hyper-heuristics, in this chapter, generation hyper-heuristics and the role of tensor analysis in improving the performance of such hyper-heuristics are considered. The tensor learning is used to detect patterns indicating mutation probabilities for each locus of a chromosome in the genetic algorithm. The experimental result of the proposed approach shows that, similar to selection hyper-heuristics, the tensor learning approach can improve the performance of the the generation hyper-heuristic significantly.

- **Chapter 6** focuses on the idea of using tensor analysis with a massive extension. The framework discussed in previous chapters is extended to a distributed agent-based learning system. Agents which use different metaheuristics during the search construct tensorial data and the proposed method is used to extract patterns from the experience of various agents, each using different search policies.

- **Chapter 7** presents the conclusions of the research outcome and points out some future research directions.

1.3 Academic Publications Produced

The following academic articles, conference papers and extended abstracts have been produced as a result of this research.

- Shahriar Asta, Ender Özcan, Tim Curtois, A Tensor-Based Hyper-heuristic for Nurse Rostering, Submitted. [Journal]
- Shahriar Asta, Ender Özcan, Simon Martin, Edmund K. Burke, A Multi-agent System Embedding Online Tensor Learning for Flowshop Scheduling, Submitted. [Journal]
- S. Asta and E. Özcan, A Tensor Analysis Improved Genetic Algorithm for Online Bin Packing, Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '15), Madrid, Spain, 2015. [Conference [41]]
- S. Asta and E. Özcan, A Tensor-based Selection Hyper-heuristic for Cross-domain Heuristic Search, Information Sciences, vol. 299, pp. 412–432, 2015. [Journal [37]]
- S. Asta and E. Özcan, An Apprenticeship Learning Hyper-Heuristic for Vehicle Routing in HyFlex, IEEE Symposium Series on Evolving and Autonomous Learning Systems (IEEE SSCI - EALS 2014), Orlando, Florida, USA, pp. 65–72, 2014. [Conference [42]]
- S. Asta, E. Özcan, A Tensor-based Approach to Nurse Rostering, Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, E. Ozcan, E. K. Burke, B. McCollum (Eds.), ISBN: 978-0-9929984-0-0, pp. 442–445, 2014. [Conference [43]]
- Shahriar Asta, Ender Özcan, Andrew J. Parkes, Batched Mode Hyper-heuristics, the 7th Learning and Intelligent OptimizatioN Conference (LION13), Catania, Italy, LNCS 7997, pp. 404–409, 2013. [Conference [44]]
- Shahriar Asta, Ender Özcan, Andrew J. Parkes, A. Şima Etaner-Uyar, Generalizing Hyper-heuristics via Apprenticeship Learning.EvoCOP 2013: 169–178, 2013. [Conference [45]]

Chapter 2

Background

This chapter provides an overview of the basic concepts in heuristic optimisation. Moreover, an in-depth description of the methodologies used in this study, together with related scientific literature review is given.

2.1 Metaheuristics

Pearl [46] defined heuristic as an intelligent search strategy for computer problem solving. In the field of optimisation problems, a heuristic can be considered as an educated guess or a 'rule of thumb' search method for finding a reasonable solution within a reasonable time. In many application areas, exact methods might fail to provide a solution to a given computationally hard problem. Although the heuristic algorithms are designed to speed up the process of discovering a high quality solution, there is no guarantee for achieving optimality. Heuristics are often problem-dependent methods that work well for an instance of a problem and may or may not be used to solve another instance of another problem or even the same problem.

The term metaheuristic was first used by Glover [47] to describe Tabu Search and has recently been defined in [8] as "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms.". Metaheuristics can be broadly classified into population-based metaheuristics, also called multi-point metaheuristics, and single-solution metaheuristics, also called single-point metaheuristics. The population-based metaheuristics, such as, Evolutionary Algorithms [48], consist of a collection of individual solutions which are maintained in a population while the single-solution metaheuristics, such as, Tabu Search [47], differ from population-based in that they improve and maintain a single solution. The

capabilities of *exploration* (diversification), being able to jump to the other regions of the search space and *exploitation* (intensification), being able to perform local search within a limited region using accumulated experience, and maintaining the balance in between them are crucial for a metaheuristic, influencing its performance. Different metaheuristics have different ways of maintaining that balance.

There is a slight confusion about metaheuristics being problem domain independent or specific. The confusion arises from the fact that the "implementation" of a relevant heuristic optimisation algorithm is also often referred to as using the same name for the metaheuristic "framework". The framework is applicable across different domains, however the implementation of a metaheuristic is domain specific. For example, the implementation of tabu search for bin packing problem can not be used for solving the traveling salesman problem. Metaheuristics (i.e., their implementations) have to be tailored for a specific problem domain and often, they are successful in obtaining high quality solutions for that domain. However, metaheuristics being a subset of heuristics come with no guarantee for the optimality of the obtained solutions. Moreover, they cannot be used for solving an instance from another problem domain. The maintenance of metaheuristics could be costly requiring expert intervention. Even a slight modification in the description of the problem could require maintenance. Almost all metaheuristics have parameters and their performance could be sensitive to the setting of those parameters. There are automated parameter tuning methods, such as F-race [49], REVAC [50] and ParamILS [51] to overcome this issue. The parameter tuning process increases the overall computation time of an approach while searching for a high quality solution to a given problem instance. However, there could be a trade-off and a higher quality solutions could be obtained for a given problem in the expense of spending more time on tuning. A selected set of well known metaheuristics, including Iterated Local Search and some Evolutionary Algorithms are briefly described in the following sections.

2.1.1 Iterated Local Search

Iterated Local Search (ILS) ([38–40]) can be considered as a modification to local search (hill climbing) methods. Local search methods are a relatively simple class of metaheuristics, based on the concept of locality between candidate solutions for a given problem. A typical local search algorithm moves through the search space from one solution to another within a neighbourhood, where a neighbour is defined as any state that can be reached from the current solution through some modification. In the case that a local search method will only move from one solution to another when that move results in some improvement with respect to a particular objective function, it is referred to as hill climbing. In cases where no improving neighbours are available, the local search/hill

climbing method can get stuck in local optima. ILS is designed to remedy this shortcoming [38]. The ILS method (Algorithm 1) operates iteratively by applying local search (lines 2 and 5) to a given solution followed by perturbing the solution (line 4). After each cycle of local search and perturbation, the new solution and the old solution are checked against an acceptance criteria which ultimately decides whether to accept the new solution or to continue from the old solution (line 6). Thus, ILS performs a search in the space of local optima and in doing so, it maintains a balance between intensification and diversification during the search.

Algorithm 1: Iterated Local Search (adopted from [38])

```
1 s_0 ← generate initial solution;
2 s* ← localSearch(s_0);
3 while not termination do
4 |   s' ← perturbation(s*, history);
5 |   s'* ← localSearch(s');
6 |   s* ← acceptanceCriterion(s*, s'*, history);
7 end
```

Ever since its introduction, ILS has been applied on a wide range of problems such as Capacitated Vehicle Routing Problem [52], Scheduling Problem [53], Aircraft Landing Problem [54], Quadratic Assignment Problem [55] and Team Orienteering Problem [56]. Moreover, numerous variants of the ILS approach including hybridisation with other methods has emerged during the time. In [57], a variant of the ILS approach is proposed, in which instead of a single perturbation and a single local search heuristic, several operators of each type are available. At each perturbation stage, the proposed variant selects a perturbation operator at random and applies it. This is while, in the local search stage, all available local search operators are applied in sequence so that one local search heuristic operates on the solution created by applying the local search heuristic before it. This framework was applied in a cross-domain fashion on a range of problem domains (Bin Packing, Permutation Flowshop and Personnel Scheduling) and produced high quality results. In a later work [58], this ILS variant was modified to cater for more adaptiveness and the integration of the extreme value-based adaptive operator selection strategy which uses credit assignment [59]. The new variant was tested on a range of well-known Capacitated Vehicle Routing Problem instances and produced good results. A multi-start ILS combined with Mixed Integer Linear Programming was proposed in [60] to solve a real-world periodic vehicle routing problem with time window. In [61] the ILS algorithm is hybridized with the Variable Neighbourhood Descent as the search engine.

A detailed account of the ILS approach, more variants and application areas can be

found in [38]. However, what is interesting (a point also confirmed in [38]) is that, metaheuristics which incorporate local search heuristics at some point in the algorithm can be considered as iterated local search. In other words, these metaheuristics, iteratively, generate some solution which is passed to a local search heuristic to exploit the solution in its neighbouring area. Whether this is performed in periodic intervals or in an irregular manner does not change the fact that what such metaheuristic is essentially doing is a form of iterated local search.

2.1.2 Evolutionary Algorithm

Evolutionary algorithms are a class of search techniques inspired from the natural process of evolution. Genetic Algorithms [62] (GAs) form a subclass of evolutionary algorithms, which, as illustrated in Algorithm 2, iteratively modifies a population of solutions from one generation (i.e., iteration) to the next via the use of mutation and crossover operators, perturbing a solution or recombining multiple solutions, respectively. Better candidate solutions have a higher chance to undergo crossover. At each generation, the old solutions get replaced by new solutions based on their fitness values, indicating the quality of solutions with respect to an evaluation (objective) function.

Algorithm 2: Genetic Algorithm

```
1  t ← 0;
2  P(t) ← initialize population;
3  evaluate population P(t);
4  while not termination do
5  |   P_P(t) ← P(t).selectParents();
6  |   P_C(t) ← reproduction(P_P(t));
7  |   mutate(P_C(t));
8  |   evaluate(P_C(t));
9  |   P(t + 1) ← buildNextGenerationFrom(P_C(t), P(t));
10 |   t ← t + 1;
11 end
```

Memetic Algorithms (MA) were introduced by Moscato [25] as a set of evolutionary algorithms that make heavy use of hill climbing. The term *meme* refers to a piece of know-how or an instruction unit which is transmissible and replicable. A simple MA introduces a local search phase into a Genetic Algorithm after crossover and mutation have been performed during the evolutionary process (Algorithm 3). Since their emergence, MAs and subsequent variants of MAs have been applied to a wide variety of problems including: educational timetabling [63–67], multi-objective optimisation problems [68, 69], permutation flow shop scheduling [70], protein folding [71], quadratic assignment problem [72] drug therapies [73], and the travelling salesman problem [74, 75]. In

addition to the application of MAs to solve practical optimisation problems, a number of studies have sought to understand the concepts underpinning MAs and the behaviour of memes [76]. Indeed the performance of an MA has been observed to be strongly linked to the choice of local search mechanism used [77]. For the interested reader, a comprehensive survey of Memetic Algorithms is offered by Neri et al. [23].

In addition to the traditional MAs which combine evolution and local search, a separate branch of *adaptive* MAs which use multiple memes emerged [78]. A subset of *adaptive* MAs evolve memes as part of the genotype for each individual in a population. Such algorithms are usually referred to as *self-adaptive* or Multi-meme Memetic Algorithms (MMAs) [79] and transfer memes from one generation to the next using inheritance mechanisms. Generally an MMA will contain individuals made up of both genetic and memetic material, with memes co-evolved during evolution. Özcan et al. [80] implemented two *adaptive* MAs as hyper-heuristics for cross-domain heuristic search.

Algorithm 3: Memetic Algorithm

```
1  t ← 0;
2  P(t) ← initialize population;
3  evaluate population P(t);
4  while not termination do
5  |   P_P(t) ← P(t).selectParents();
6  |   P_C(t) ← reproduction(P_P(t));
7  |   mutate(P_C(t));
8  |   localSearch(P_C(t));
9  |   evaluate(P_C(t));
10 |   P(t+1) ← buildNextGenerationFrom(P_C(t), P(t));
11 |   t ← t+1;
12 end
```

MAs are special cases of a broader concept in computational intelligence, namely, Memetic Computing (MC) [81, 82]. Currently, memetic computation serves as an umbrella term which includes any methodology which consists of several interacting components. Furthermore, the notion of memetic computing is not exclusive to population-based methods [24, 26, 27]. Single point search methods which operate on one candidate solution only are also covered by the concept of memetic computation. Subsequently, in a learning single point heuristic which learns patterns/rules from the search environment to improve it's performance, the extracted patterns are memes which are passed to other components of the heuristic to make decision making more efficient.

2.2 Hyper-heuristics

Hyper-heuristics have emerged as effective and efficient methodologies for solving hard computational problems [10]. They perform search over the space formed by a set of low level heuristics, rather than solutions directly [15]. Burke et al. [13] defined a hyper-heuristic as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". Hyper-heuristics are not allowed to access problem domain specific information. It is assumed that there is a conceptual *barrier* between the hyper-heuristic level and problem domain where the low level heuristics, solution representation, etc. reside. This specific feature gives hyper-heuristics an advantage of being more general than the existing search methods, since the same hyper-heuristic methodology can be reused for solving problem instances even from different domains. A hyper-heuristic either selects from a set of available low level heuristics or generates new heuristics from components of existing low level heuristics to solve a problem, leading to a distinction between *selection* and *generation* hyper-heuristic, respectively [13]. Also, depending on the availability of feedback from the search process, hyper-heuristics can be categorized as *learning* and *no-learning*. Learning hyper-heuristics can be further categorized into online and offline methodologies depending on the nature of the feedback. Online hyper-heuristics learn *while* solving a problem whereas offline hyper-heuristics process collected data gathered from training instances prior to solving the problem. Many researchers and practitioners have been progressively involved in hyper-heuristic studies for solving difficult real world combinatorial optimisation problems ranging from channel assignment to production scheduling (Table 2.1). More on hyper-heuristics can be found in [11, 15, 83].

Table 2.1: Some selected problem domains in which hyper-heuristics were used as solution methodologies.

Problem Domain [Reference]	Problem Domain [Reference]
Channel assignment [84]	Job shop scheduling [9]
Component placement sequencing [85]	Sales summit scheduling [1]
Examination timetabling [86]	Space allocation [87]
Nurse rostering [88]	University course timetabling [88]
Orc quest, logistics domain [89]	Vehicle routing problems [90]
Packing [91]	Production scheduling [92]

2.2.1 Selection Hyper-heuristics

A selection hyper-heuristic has two main components: *heuristic selection* and *move acceptance* methods. While the task of the heuristic selection is to choose a low level heuristic at each decision point, the move acceptance method accepts or rejects the

resultant solution produced after the application of the chosen heuristic to the solution in hand. This decision requires measurement of the quality of a given solution using an *objective* (evaluation, fitness, cost, or penalty) function. Over the years, many heuristic selection and move acceptance methods have been proposed. The following sections provide a review of these techniques, though, a comprehensive survey on hyper-heuristics including their components can be found in [11, 15].

2.2.1.1 Heuristic Selection and Move Acceptance Methodologies

In this section, we describe some of the basic and well known heuristic selection approaches. [93, 94] are the earliest studies testing simple heuristic selection methods as a selection hyper-heuristic component. One of the most basic and preliminary approaches to select low level heuristics is the Simple Random (SR) approach requiring no learning at all. In SR, heuristics are chosen and applied (once) at random. Alternatively, when the randomly selected heuristic is applied repeatedly until the point in which no improvement is achieved, the heuristic selection mechanism is Random Gradient. Also, when all low level heuristics are applied and the one producing the best result is chosen at each iteration, the selection mechanism is said to be greedy. The heuristic selection mechanisms discussed so far do not employ learning. There are also many selection mechanisms which incorporate learning mechanisms. Choice Function (CF) [93, 95, 96] is one of the learning heuristic selection mechanisms which has been shown to perform well. This method is a score-based approach in which heuristics are adaptively ranked based on a composite score. The composite score itself is based on few criteria such as: the individual performance profile of the heuristic, the performance profile of the heuristic combined with other heuristics and the time elapsed since the last call to the heuristic. The first two components of the scoring system emphasise on the recent performance while the last component has a diversifying effect on the search.

In [89], a Reinforcement Learning (RL) approach has been introduced for heuristic selection. Weights are assigned to heuristics and the selection process takes weight values into consideration to favour some heuristics to others. In [88], the RL approach is hybridized with Tabu Search (TS) where the tabu list of heuristics which are temporarily excluded from the search process is kept and used. In [97] a selection hyper-heuristic based on Particle Swarm Optimisation was proposed to solve the resource provisioning-based scheduling in grid environment. [98] proposed a selection hyper-heuristics based on the Ant Colony Optimisation algorithm to schedule intercell transfers in cellular manufacturing systems. There are numerous other heuristic selection mechanisms. Interested reader can refer to [11] for further detail.

As for the move acceptance component, currently, there are two types of move acceptance methods: *deterministic* and *non-deterministic* [11]. The deterministic move acceptance methods make the same decision (as for acception/rejection of the solution provided by a heuristic) irrespective of the decision point. In contrast to deterministic move acceptance, non-deterministic acceptance strategies incorporate some level of randomness resulting in different decisions for the same decision point. The non-deterministic move acceptance methods almost always are parametric, utilising parameters such as time or current iteration.

Initial studies on selection hyper-heuristics focused on some simple move acceptance methods, such as Improvement Only (IO) [93], Improvement and Equal (IE) [99] and Naive Acceptance (NA) [57, 93]. The IO acceptance criteria only accepts solutions which offer an improvement in the current objective value. The IE method accepts all solutions which result in objective value improvement. It also accepts solutions which do not change the current objective value. Both IO and IE strategies are deterministic strategies. The NA strategy is a non-deterministic approach which accepts all improving and equal solutions by default and worsening solutions with a fixed probability of α. Although there are more elaborate move acceptance methods, for example, Monte-Carlo based move acceptance strategy [85], Simulated Annealing [100], Late Acceptance [101], there is strong empirical evidence that combining simple components under a selection hyper-heuristic framework with the right low level heuristics could still yield an improved performance. [102] shows that the performance of a selection hyper-heuristic could vary if the set of low level heuristics change, as expected. [103] describes the runner up approach at a high school timetabling competition, which uses SR as heuristic selection and a move acceptance method with 3 different threshold value settings. Moreover, there are experimental and theoretical studies showing that mixing move acceptance can yield improved performance [104–106]. More sophisticated acceptance algorithms can be found in the scientific literature [11].

2.2.2 Generation Hyper-heuristics

Genetic Programming (GP) is one of the most commonly used techniques to generate hyper-heuristics [107–110]. Some problem domains for which GP is used to construct either a heuristic or component of a solution method include job shop scheduling [111], 0/1 knapsack [112], uncapacitated examination timetabling [113], satisfiability problem [114] and strip packing [115]. Drake et al. [116] used grammatical evolution to generate components of a variable neighbourhood search approach for solving vehicle routing

problem. A recent work in [117] provided an evolutionary approach based on gene expression programming to generate selection hyper-heuristic components for cross domain search.

Another emerging approach in generation hyper-heuristics is the use of data mining techniques to automatically generate new effective heuristics. In [45], apprenticeship learning (AL) technique was used to generalize hyper-heuristics in the Bin-Packing domain (see Chapter 5 for details). The AL method has a wide range of applications in control and robotics and is heavily based on Inverse Reinforcement Learning (IRL)[118]. Inspired by the IRL approach, the main idea in [45] is to create an algorithm which learns what course of action to take, by simply *watching* a couple of other algorithms which perform well in various problem domains. That is, the algorithm learns the behaviour of an *expert* algorithm by constructing a dataset via recording the actions of experts at each state of the search process. The classifier produced from this dataset is used to predict the best action at a given search state while dealing with an unseen problem instance. The AL-based approach in [45] was trained on small problem instances and was capable of generalising the extracted knowledge to larger problem instances. The major drawback of the approach proposed in [45] was that the definition of the search state was problem dependent.

In a later study [42], the work in [45] was extended by providing a general, problem domain independent state definition. The authors also investigated whether an AL hyper-heuristic is able to perform similar (or better) than the expert algorithm on a given problem domain. Furthermore, various hyper-heuristic components from which expert knowledge can be extracted were analysed. The modified framework was applied to various instances of the vehicle routing problem and it was observed that the generated AL hyper-heuristic is able to outperform the expert on the majority of instances exhibiting an impressive performance.

In [35] Learning Classifier Systems (LCS) are used to generate hyper-heuristics which solve Constraint Satisfaction Problem (CSP). LCS are adaptive rule-based systems which automatically generate a set of rules from the data [119]. Therefore, the LCS is trained in an offline manner on a set of training instances and necessary rules are generated. This rule set is then used as a selection hyper-heuristic to decide which heuristic to choose at each step of the search. The result of applying the proposed framework on various CSP test instances showed that the proposed approach provides reasonably good solutions which are competent with some of the well-known algorithms in the field.

Also, in [120], an evolutionary process is used to evolve back propagation neural networks which in turn predicts which heuristics to use at each search step. The generated neural network is hence a selection hyper-heuristic and the evolutionary process creating this

neural network is considered to be a generation hyper-heuristic. The proposed framework is designed for CSP problem instances. For each new problem instance a new neural network has to be trained (offline). The work in [35] and [120] was later modified to generate selection hyper-heuristics using logistic regression [34].

2.2.3 Hyper-heuristics Flexible Framework (HyFlex)

Hyper-heuristics Flexible Framework (HyFlex) [36] is an interface to support rapid development and comparison of various (hyper-/meta) heuristics across various problem domains. The HyFlex platform promotes the reusability of hyper-heuristic components. In HyFlex, hyper-heuristics are separated from the problem domain via a domain barrier [1] to promote domain-independent automated search algorithms. Hence, problem domain independent information, such as the number of heuristics and objective value of a solution, is allowed to pass through the domain barrier to the hyper-heuristic level (Figure 2.1). On the other hand, pieces of problem dependent information, such as, problem representation and objective function are kept hidden from the higher level search algorithm. Restricting the type of information available to the hyper-heuristic to a domain independent nature is considered to be necessary to increase the level of generality of a hyper-heuristic over multiple problem domains. This way the same approach can be applied to a problem from another domain without requiring any domain expert knowledge or intervention.

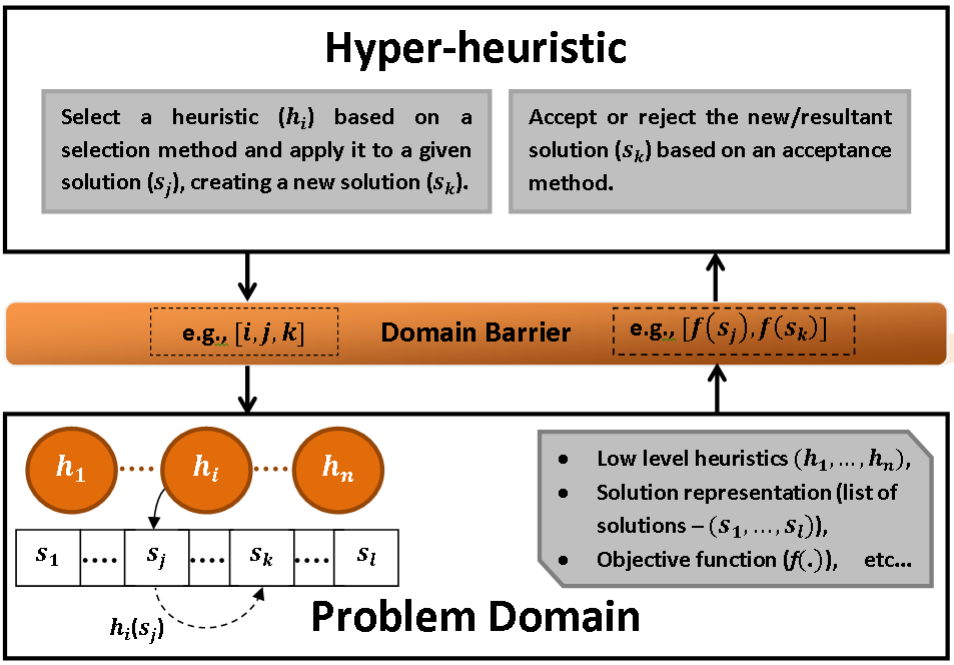


FIGURE 2.1: A selection hyper-heuristic framework [1].

HyFlex v1.0 is implemented in Java respecting the interface definition and was the platform of choice at a recent competition referred to as the Cross-domain Heuristic Search Challenge (CHeSC 2011) [1]. The CHeSC 2011 competition aimed at determining the state-of-the-art selection hyper-heuristic judged by the median performance of the competing algorithms across thirty problem instances, five for each problem domain. Formula 1 scoring system is used to assess the performance of hyper-heuristics over problem domains. In formula 1 scoring system, for each instance, the top eight competing algorithms receive scores of $10, 8, 6, 5, 4, 3, 2$ or 1 depending on their rank on a specific instance. Remaining algorithms receive a score of 0. These scores are then accumulated to produce the overall score of each algorithm on all problem instances. The competing algorithms are then ranked according to their overall score. The number of competitors during the final round of the competition was 20. Moreover, a wide range of problem domains is covered in CHeSC 2011. Consequently, the results achieved in the competition along with the HyFlex v1.0 platform and the competing hyper-heuristics currently serve as a benchmark to compare the performance of novel selection hyper-heuristics.

The CHeSC 2011 problem domains include Boolean Satisfiability (SAT), One Dimensional Bin Packing (BP), Permutation Flow Shop (FS), Personnel Scheduling (PS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). Each domain provides a set of low level heuristics which are classified as mutation (MU), local search (LS)(also referred to as hill climbing), Ruin and Re-create (RR) and crossover (XO) heuristics (operators). Each low level heuristic, depending on it's nature (i.e. whether it is a mutational or a local search operator) comes with an adjustable parameter. For instance, in mutational operators, the *mutation density* determines the extent of changes that the selected mutation operator yields on a solution. A high mutation density indicates wider range of new values that the solution can take, relevant to its current value. Lower values suggest a more conservative approach where changes are less influential. As for the *depth* of local search operators, this value relates to the number of steps completed by the local search heuristic. Higher values indicate that local search approach searches more neighbourhoods for improvement. Table 2.2 summarizes the low level heuristics for each domain of CHeSC 2011 and groups them according to their type (e.g. MU, RR, XO and LS).

The description of each competing hyper-heuristic can be reached from the CHeSC 2011 website. The ranking of twenty CHeSC 2011 participants is provided in Table 2.3. The top three selection hyper-heuristics that generalize well across the CHeSC 2011 problem domains are multi-stage approaches of AdapHH [121], VNS-TW [122] and ML [123].

[1] http://www.asap.cs.nott.ac.uk/external/chesc2011/

TABLE 2.2: The number of different types of low level heuristics {mutation (MU), ruin and re-create heuristics (RR), crossover (XO) and local search (LS)} used in each CHeSC 2011 problem domain.

Domain	MU	RR	XO	LS	Total
SAT	6	1	2	2	11
BP	3	2	2	1	8
PS	1	3	5	3	12
PFS	5	2	4	4	15
TSP	5	1	3	4	13
VRP	3	2	3	2	10

TABLE 2.3: Rank of each hyper-heuristic (denoted as HH) competed in CHeSC 2011 with respect to their Formula 1 scores.

Rank	HH		Score	Rank	HH		Score
1	AdapHH	[121]	181	11	ACO-HH	[124]	39
2	VNS-TW	[122]	134	12	GenHive	[125]	36.5
3	ML	[123]	131.5	13	DynILS	[126]	27
4	PHUNTER	[127]	93.25	14	SA-ILS	[N/A]	24.25
5	EPH	[128]	89.75	15	XCJ	[N/A]	22.5
6	HAHA	[129]	75.75	16	AVEG-Nep	[130]	21
7	NAHH	[131]	75	17	GISS	[132]	16.75
8	ISEA	[133]	71	18	SelfSearch	[134]	7
9	KSATS-HH	[135]	66.5	19	MCHH-S	[136]	4.75
10	HAEA	[137]	53.5	20	Ant-Q	[138]	0

The winning algorithm, AdapHH is a multi-phase learning hyper-heuristic [121]. AdapHH adaptively determines the subset of low-level heuristics to apply at each phase. The duration with which each heuristic is applied is also dynamically determined during the search. The algorithm accepts only improving solutions in the absence of which the algorithm refuses to accept worsening solutions until no improvements are observed within an adaptively adjusted number of iterations. The parameters of each low-level heuristic are dynamically modified via a reinforcement learning method. This is while low-level heuristics are selected based on a quality index measure. This measure uses few weighted performance metrics to compute the quality index for each heuristic. Among these metrics are the number of new best solutions explored, the total improvement and worsening during the run and also the current phase and finally the remaining execution time. A heuristic with a quality index less than the average of the quality indexes of all the heuristics is excluded from the selection process in the corresponding phase. Using a tabu style memory, the number of phases in which the heuristic is consecutively excluded is maintained. Whenever this number exceeds a threshold the heuristic gets excluded

permanently. AdapHH also employs a relay hybridisation with which effective pairs of heuristics which are applied consecutively are identified.

The hyper-heuristic algorithm which ranked the second in the competition, VNS-TW [122], is a two phase algorithm. The first phase consists of applying mutational or ruin and recreate type of low-level heuristics to a population of initial solutions. Subsequently, all the local search heuristics are applied until no more improvements are observed. In the next phase, the algorithm shifts from a population-based method into a single solution approach in which the best solution achieved in the first phase is used. Iteratively, A circular priority queue of mutational heuristics is formed based on the severity of changes that they imply. Subsequent to the application of heuristics a local search is applied. This approach was ranked first in the Personnel Scheduling problem domain and second in the overall.

The third ranking hyper-heuristic which is proposed in [123], namely ML, relies explicitly on intensification and diversification components during the search process. A solution is generated initially which goes through a diversification stage by the application of a mutational or ruin-recreate low-level heuristic. The solution achieved at this stage is then subjected to a local search heuristic for further improvement. The local search heuristic is applied until no further improvements can be achieved. The acceptance mechanism, accepts improving solutions as well as the cases where the solution has not improved over the last 120 iterations.

The PHunter hyper-heuristic [127] was the algorithm which ranked fourth in the CHeSC 2011 competitions. PHunter imitates the actions of a traditional pearl hunter. In the PHunter algorithm the low level heuristics are grouped as either local search (intensifier heuristics) or non-local search. The former imitates the diving behaviour of a pearl hunter while the latter is equivalent to the resurfacing behaviour of the pearl hunter following a change in the search area for another dive. Furthermore, dives (local search) are grouped into shallow and deep dives. A shallow dive is a local search with low depth of search parameter value while a deep dive assigns higher values to the parameter depth of search. Also, a tabu list is employed to enlist undesired intensification and diversification moves. The PHunter algorithm determines portfolios for categorized low level heuristics by training and using a C4.5 decision tree [139]. It then uses these portfolios in various conditions where each condition is estimated through an adaptation stage at the beginning of the search. The search procedure is based on repeated intensification and diversification and can generally be described as a type of Iterated Local Search (ILS).

The remaining hyper-heuristics in Table 2.3 employ a variety of interesting concepts. The hyper-heuristic proposed in [128] which is based on Evolutionary Programming

with Co-evolution, simultaneously evolves population of both solutions and heuristics. The HAHA algorithm [129] is an interesting hyper-heuristic which repeatedly switches between single-point and population-based search strategies. The algorithm is an online learning hyper-heuristic with embedded adaptation mechanism to guide the selection of low level heuristics. In contrast to other CHeSC 2011 contestants, the hyper-heuristic proposed in [131] uses a non-adaptive method by discarding the dominated heuristics in tuning procedure. This pre-processing step results in a fixed (non-adaptive) algorithm schemata which is used to solve all the instances. The ISEA algorithm [133] uses Evolutionary Algorithm to guide an Iterated Local Search through the search landscape. The Hybrid Adaptive Evolutionary Algorithm (HAEA) [137] takes a probabilistic approach to adapt the likelihood of selecting and applying low level heuristics as the search makes progress. The algorithm proposed in [135] combines Simulated Annealing, Tabu Search and Reinforcement Learning to construct a hyper-heuristic. Heuristics are chosen from a tabu list while simulated annealing is used as the acceptance methodology. Reinforcement learning has been used in the sense that heuristics with good performance are rewarded during the search. In [124] Ant Colony Optimisation has been used as the higher level strategy to select low level heuristics. The hyper-heuristics in [125] uses a five phase approach based on the Genetic Hive algorithm. The core of the algorithm repeatedly performs intensification, stagnation and diversification steps throughout the search. The hyper-heuristic in [126] proposes the use of Dynamic Iterated Local Search to adaptively adjust the intensity of the mutation operator within a hyper-heuristic which is based on ILS. The study in [130] proposes a hyper-heuristic (AVEG-Nep) which is based on reinforcement learning. In heuristics research community, it is usually said that a (hyper-/meta) heuristic uses reinforcement learning whenever a rewarding scheme is used to reward well performing components of the algorithm. However, in machine learning and robotics community where this algorithm was invented and later modified, only having a rewarding scheme does not suffice to consider the algorithm as a member of reinforcement learning group of algorithms. The standard definition and formulation of this popular and powerful approach is given in [140]. The AVEG-Nep hyper-heuristic stays loyal to this formulation and can be considered as a real reinforcement learning-based hyper-heuristic. The algorithm proposed in [132] uses a generic iterated simulated annealing method as the higher level strategy to select low level heuristics and decide whether or not to accept the outcome. In the hyper-heuristic proposed in [134] self search is used to decide on the selection of the next heuristic during the search. This decision is based on a variety measures: variation in quality of solutions, application time, frequency of applying each heuristic and the number of times a low level heuristic fails to yield a difference in the quality of the current solution. The algorithm proposed in [136] assumes that heuristic selection follows a Markov chain which models the probability of moving from one low level heuristic to another. This model is then used as a core

decision making mechanism for heuristic selection throughout the search. Finally, the hyper-heuristic proposed in [138] is an agent-based system which combines Ant Colony Optimisation algorithm with a reinforcement scheme. The Ant-Q algorithm is a constructive hyper-heuristic which aims at building a good sequence of low level heuristics to be applied on the problem instance.

Following the competition, the hyper-heuristics community started to recognize the CHeSC 2011 competition results as a benchmark for evaluating the performance and generality level of a selection hyper-heuristic. There is a growing number of studies evaluating the performances of new selection hyper-heuristics on the CHeSC 2011 benchmark. [141] tested a variant of choice function hyper-heuristic over the CHeSC 2011 benchmark. An adaptive neighbourhood iterated local search is proposed and applied on HyFlex problem domains [142, 143]. In [144], an iterated local search method is tested on HyFlex problem domains. [145] evaluated variants of *late acceptance*-based selection hyper-heuristics on the CHeSC 2011 benchmark and points out the best configuration for the late acceptance strategy which accepts the current solution if its quality is better than the quality of a solution obtained from a certain number of iterations ago.

HyFlex was later extended in [44] where it was modified to cater for new problem domains, more instances and accommodate batch processing of problem instances by lifting the per-instance time constraint. Instead of considering a time cap for each instance and each run, an overall time limit was introduced leaving hyper-heuristics free to decide on how much of the overall time they would spend on a given instance. The new version (HyFlex v1.1 as opposed to HyFlex v1.0 which was used in CHeSC 2011) was the framework of choice for the Second Cross-Domain Heuristic Challenge (CHeSC 2014). The public problem instances considered in this new challenge were chosen from: Boolean Satisfiability (SAT), One Dimensional Bin Packing (BP), Permutation Flow Shop (FS) and Personnel Scheduling (PS) problems. The hidden instances used in the competition were chosen from: Personnel Scheduling, Multi-dimensional Knapsack and Vehicle Routing problems. The competition consisted of two different tracks: single and multi-threaded.

2.3 Machine Learning

Machine learning is the science of learning useful rules and recognising hidden patterns from example data [18, 19]. These examples are either presented as data or are acquired from direct interaction with the environment, leading to offline and online variants of machine learning algorithms respectively. Offline machine learning techniques use training algorithms to mine the data for hidden patterns. This is while online algorithms

build up a generalized model gradually as they interact with the environment they are inhabiting. Furthermore, supervised, semi-supervised and un-supervised training approaches are possible. In supervised machine learning, example data is accompanied by the desired output for each decision point while in unsupervised learning desired output is not provided. Semi-supervised techniques have access to the desired output during the training only.

Machine learning techniques can be considered as efficient theory-based heuristics, however, they use some sort of probabilistic, statistic or derivative information/inference to produce predictive models [18, 19]. Perhaps the most evident difference between machine learning techniques and other heuristics (such as metaheuristics and evolutionary algorithms) is that rather than trying to optimize a given objective function (which is the case in most heuristics) they instead build a model of the objective function using statistical, probabilistic or gradient methods. (Semi-) Supervised approaches tend to do this using example data provided to them in training while unsupervised methods usually draw inference from the data without considering a desired output. Moreover, (hyper-/meta)heuristics can also be considered and used for machine learning, however, this is out of scope of this study.

Traditionally, a supervision dataset contains records of data where each record is a vector consisted of feature values which describe the attributes of a decision point. The desired output (also called label, decision or action) given for each record is usually provided by human experts (hence the term supervised). Data labels describe a class label, an action to be taken or a desired value for the given record (decision point). A dataset in supervised learning can thus be formulated as in the following equation.

$$\mathcal{D} = \{(\phi_e^i, a_i)\} \quad , \quad i = 1 \cdots N \tag{2.1}$$

where $\mathcal{D}$ denotes the dataset which is a set of N feature records/vectors (ϕ_e^i) and their respective action labels (a_i). A feature vector, is a vector of length r where r is the number of features in a given dataset. Unlike supervised learning, in un-supervised learning, feature vectors are not labelled. Thus, a dataset prepared for an un-supervised learning method can be expressed with the following equation.

$$\mathcal{D} = \{(\phi_e^i)\} \quad , \quad i = 1 \cdots N \tag{2.2}$$

Features and labels can take their values from real or integer numbers or even have nominal values. Given a dataset, depending on the existence of record labels, supervised or un-supervised training algorithms can be chosen and applied on the data. However,

prior to training, data is usually split into training and test datasets. The former is used to build a predictive model which is then applied on the latter to evaluate the accuracy of the built model. The training is where the learning occurs and the output of the training algorithm is a model which describes the data in a generalized manner. The resulting model can have various applications depending on the data and objective of the training algorithms. When the goal is classification, the model is used to assign each unseen data record to the right class label. In case the objective is sequential decision making, given that a training algorithm which properly fits this purpose is chosen (i.e. Q-Learning), the model can be used as a state-action mapper which decides what to do next given an unseen state. In summary, the predictive power of machine learning techniques has enabled researchers to solve a variety of challenging problems such as face recognition [20], natural language processing [21] and bioinformatics [22] among others. In what follows, a description of machine learning methods used in this study is provided.

2.3.1 k-means clustering

The k-means algorithm is an un-supervised machine learning method which partitions the data into k mutually exclusive clusters where k is given a priori. It partitions the example data such that records confined in each cluster are as close as possible to one another (minimum within cluster distance) while being as far from records in other clusters as possible (maximum between clusters distance). Several metrics can be used to measure the distance. Some popular distance metrics are Euclidean distance and cosine similarity [18]. Each cluster is identified by a central point referred to as cluster centroid. Centroids are simply the mean of the feature values of the records in the cluster. Hence, centroids have the same dimension as each record of the example data (i.e. both have a length of r).

Having a dataset $\mathcal{D}$ (as in Equation 2.2), the k-means algorithm partitions the data into k clusters using the following equation to minimize the within cluster distance for all the points j in each cluster.

$$\underset{\mathbf{C}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{i \in c_j} \left(||\phi^i - \phi^{c_j}|| \right)^2 \tag{2.3}$$

where $\mathbf{C} = \{c_1, c_2, \cdots c_k\}$ is the set of all cluster centroids, the jth centroid is denoted by c_j and ϕ^{c_j} is its feature vector. The algorithm which actually achieves this is based on Expectation-Maximisation for which a pseudo code is given in Algorithm 4.

Algorithm 4: The k-means clustering algorithm.

1 Initialize cluster centroids to random points $\phi^{c_1}, \phi^{c_2}, \cdots, \phi^{c_k} \in \mathbb{R}^r$;
2 **while** *not converged* **do**
3 **for** *(each record* $i = 1 \cdots N$ *in the dataset)* **do**
4 assign record i to cluster with centroid c_j where $c_j \leftarrow \text{argmin}_{c_j} ||\phi^i - \phi^{c_j}||^2$;
5 **end**
6 **for** *(each centroid* $j = 1 \cdots k$*)* **do**
7 $\phi^{c_j} = \frac{\sum_{i=1}^{N} 1\{i \in c_j\}\phi^i}{\sum_{i=1}^{N} 1\{i \in c_j\}}$;
8 **end**
9 **end**

In Algorithm 4, first cluster centroids are randomly initialized (line 1). Then each feature vector ϕ^i is assigned to the closest cluster represented by the centroid point c_j (line 3 to 5). The distance metric here is the Euclidean distance (i.e. $||\phi^i - \phi^{c_j}||$). In the subsequent step, the coordinates of the centroids are updated. The notation $1\{i \in c_j\}$ is a check which determines whether the record i belongs to cluster c_j. Thus, the numerator of the fraction in line 7, is the sum of feature values of the records belonging to cluster c_j and the denominator is the count of these records. The resulting value is hence the centroid feature values/coordinates.

2.3.2 Learning from demonstration

Learning from Demonstration (LfD) is the task of learning a policy from demonstration data provided by an expert. This technique is also referred to as Apprenticeship Learning (AL), imitation learning, learning by watching or programming by demonstration [118]. Throughout this study, we will refer to this technique as apprenticeship learning. In the recent years, the focus of the study on AL has been mainly in the field of robotics and control where a robot learns a task from the demonstrations provided by a human expert. The ever increasing presence of the robots in environments where the end-users are not experts in robotics, control or programming makes useful utilisation of robotic facilities a hard task which requires constant robotic-expert intervention. AL techniques are introduced to simplify the interaction of non-robotics-experts with robots. Employing AL techniques, the end-user will not have to be a robotic-expert in order to construct an engineered and efficient control algorithm for the robot. AL is a technique in which an agent learns a task, policy, model of a system or a set of rules from demonstrations provided by a human demonstrator which is referred to as the *expert*. The term expert refers to the end-user of the robot/agent with sufficient knowledge in the field in which the robot/agent is supposed to operate and it does not refer to an expert in robotics,

algorithm design or programming. It is assumed that the actions of the expert reflect the optimum policy, an assumption which may not always be true.

The demonstration dataset is precisely as in Equation 2.1 where the features describe a state for each time unit and the label is the action chosen by the expert at that time unit. The dataset transfers expert's knowledge to the agent. What distinguishes apprenticeship learning from other supervised learning methods though is that in apprenticeship learning an explicit reward function or criteria is not used. That is, the robot/agent learns to imitate a complicated task and the data presented to the task is highly abstract (like welding a car's roof in case of industrial robots) [118]. Once the dataset is provided various machine learning techniques such as Inverse Reinforcement Learning[118], Gaussian Mixture Models [146], Confidence-Based Learning [147] and Dogged Learning [148] can be used to act as a state-action mapper for the robot/agent.

There is no reason why the same cannot be done in heuristic optimisation. There are some algorithms which serve as state-of-the-art algorithm and have high performances in a number of problem domains. These algorithms can be viewed as experts in the context of apprenticeship learning. Furthermore, more than one expert can be employed where each expert has high performance in at least one problem domain different than other experts. Data could be collected from the trace of these experts on the problem instances they are good at. It is expected that, similar to expert data in robotics, the collected data is highly abstract. A generalized model can be built using this data, resulting in a learned machine which theoretically could perform at least as good as the experts on all the problem domains. In this study, the feasibility of this theory has been put to test. In the coming sections a report of these trials is given. Also, the challenges encountered during experiments are presented.

2.3.3 Tensor Analysis

Many problems produce data of a nature which is best described and represented in high dimensional data structures. However, for the sake of simplicity, the dimensionality of data is often deliberately reduced. For instance, in face recognition, the set of images of various individuals constitutes a three dimensional data structure where the first two dimensions are the x and y coordinates of each pixel in each image and the third dimension is the length of the dataset. This is while, in classical face recognition (like the eigenface method), each face image is reduced to a vector by concatenating the pixel rows of the image. Consequently, the dataset, which is 3D (or maybe higher) in nature, is reduced to a 2D dataset for further processing. However, [149] shows that by sticking to the original dimensionality of the data and representing the dataset as a 3^{rd}-order

tensor, better results are achieved in terms of the recognition rate. This is due to the fact that collapsing the naturally 3D data into a matrix results in loss of information regarding the dependencies, correlation and useful redundancies in the feature space [150]. On the other hand, representing the data in its natural high dimensional form helps with preserving the latent structures in the data. Employing tensor analysis tools helps in capturing those latent relationships ([151], [152]). A good example of this is given in [153] where the goal is to classify specific human actions (such as waving hands, running and jugging) in a set of videos. Each instance in the dataset is a video containing one of the desired actions performed by a human actor. Each video in itself is three dimensional. That is, video frames are stacked one after another to form a three dimensional data structure (or a $3^{rd}-$order tensor). Video tensors of a specific action are then concatenated to one another resulting in a three dimensional corpora representing the given action (each video is a three dimensional array, hence, concatenating them will also result in another three dimensional array). Having a tensor for each action available in the dataset, [153] used tensor analysis to classify human actions in unseen videos and detect body part motions most associated to each action automatically. Similar applications have been registered in various research areas such as hand written digit recognition [154], image compression [155], object recognition [156], gait recognition [157], Electroencephalogram (EEG) classification [158], Anomaly detection in streaming data [159], dimensionality reduction [150], tag recommendation systems [160] and Link Prediction on web data [161].

Tensor decomposition (a.k.a tensor factorisation) is used to detect latent relationships between various modes of data. The reason why factorisation works in this context and is able to extract latent relationships between various modes of data lies in the elegant mechanism with which it deals with high dimensional datasets. Tensor factorisation merges information from various dimensions of data by generalising matrix Singular Value Decomposition (SVD) to higher dimensions and applying it to all modes of data [171]. Several factorisation methods exists in the literature. Higher Order SVD (HOSVD) [162], Tucker decomposition [163], Parallel Factor (a.k.a PARAFAC or CANDECOMP or CP) [164] and Non-negative Tensor Factorisation (NTF) [165] are among numerous factorisation methods proposed by researchers. Tucker decomposition has applications in data compression [166], dimensionality reduction [149] and noise removal [167] among others. Also CP decomposition has been used for noise removal and data compression [168]. In addition, it has a wide range of applications in many scientific areas such as Data Mining [169] and telecommunications [170]. There are few other factorisation methods which mainly originate from CP and/or Tucker methods. Each of these methods (such as INDSCAL, PARAFAC2 and PARATUCK2) are widely known in specific fields such as chemometrics or statistics. For more details on these methods

the reader is referred to [171]. Also, the studies in [172] and [173] provide very detailed comparison between factorisation methods.

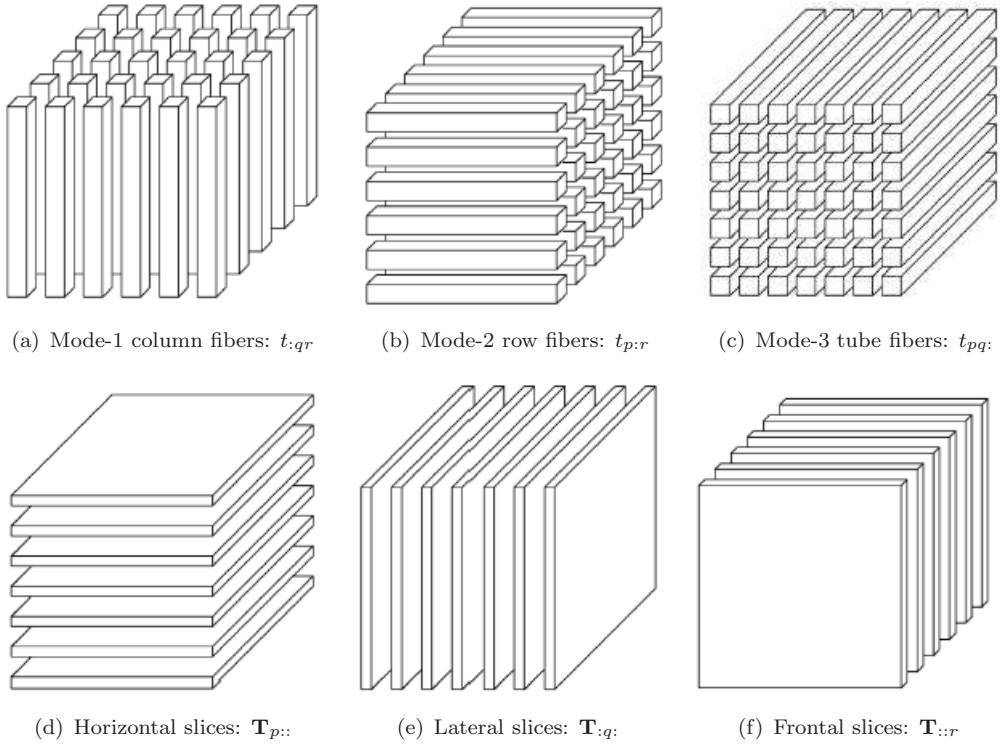
2.3.3.1 Notation and Preliminaries

Tensors are multidimensional arrays and the *order* of a tensor indicates its dimensionality. Each dimension of a tensor is referred to as a *mode*. In our notation, following [171], a boldface Euler script letter, boldface capital letter and boldface lower-case letter denote a tensor (e.g., $\mathcal{T}$), matrix (e.g., $\mathbf{M}$) and vector (e.g., $\mathbf{v}$), respectively. The entries of tensors, matrices and vectors (and scalar values in general) are indexed by italic lower-case letters. For example, the (p, q, r) entry of a $3^{rd}-$order (three dimensional) tensor $\mathcal{T}$ is denoted as t_{pqr}. Also, the nth element in a sequence is denoted by a superscript in parenthesis. For instance, $\mathbf{A}^{(n)}$ denotes the the nth matrix in a sequence of matrices. Please note that, in this study, we only deal with 3^{rd}-order tensors and all definitions and discussions are based on this assumption.

Tensor analysis involves a huge amount of indexing. Carelessness in dealing with indexes can lead to serious computation errors and undesired results. Luckily, there exists a convention using which the perils of indexing errors can be diminished. One of the most frequent practices in tensorial operations is extracting sub-data from tensor. This is useful both during the construction of the tensor and analysing it. Tensor sub-data can have the form of a sub-tensor or a sub-array. Since tensors have multiple dimensions, there are more than one way to access this sort of sub-data. Subtensors and subarrays are achieved by fixing a subset of indices. For example, given a 3^{rd}-order tensor $\mathcal{T}$ and fixing the second and third indices, one can extract *fibers* of the tensor (Figure 2.2(a)). Since the sub-array being extracted is corresponding to the first index (first mode) the vector $t_{:qr}$ is referred to as the mode-1 fiber. Similarly $t_{p:r}$ and $t_{pq:}$ are mode-2 and mode-3 fibers (figures 2.2(b) and 2.2(c)). Moving a step higher, one can extract *slices* of a tensor by fixing only one index. For example, fixing the first index results in a matrix $T_{p::}$ which is called the horizontal slice (Figure 2.2(d)). Lateral ($T_{:q:}$, Figure 2.2(e)) and frontal ($T_{::r}$, Figure 2.2(f)) slices can be achieved in similar fashion.

Apart from indexing methods, several tensorial operations exist which are frequently used in tensor-based analytic approaches. Covering all of these operations is out of the scope of this thesis as they are not directly used here. However, here we explain a relevant operation which is used during the factorisation procedure.

Tensor *matricisation* is a procedure to transform a tensor to a matrix. It is also known as *unfolding* and *flattering*. Similar to indexing operations, there are many ways to



(a) Mode-1 column fibers: $t_{:qr}$ (b) Mode-2 row fibers: $t_{p:r}$ (c) Mode-3 tube fibers: $t_{pq:}$

(d) Horizontal slices: $\mathbf{T}_{p::}$ (e) Lateral slices: $\mathbf{T}_{:q:}$ (f) Frontal slices: $\mathbf{T}_{::r}$

FIGURE 2.2: *Fibers* and *Slices* of a 3^{rd}-order tensor $\mathcal{T}$ (adopted from [171]).

matricize a tensor. However, the only way relevant to this thesis is the mode-n matricisation approach. Given a tensor $\mathcal{T}$, its mode-n matricisation is denoted by $\mathbf{T}_{(n)}$ which arranges the mode-n fibers of the tensor to be the columns of the resulting matrix. That is, the tensor element indexed as (i_1, i_2, i_3) maps to the (i_n, j) element in the resulting matrix where

$$j = 1 + \sum_{k=1, k\neq n}^{3} (i_k - 1) J_k \quad with \quad J_k = \prod_{m=1, m\neg n}^{k-1} I_m \tag{2.4}$$

This operation can be clarified better through an example. Suppose that $\mathcal{T} \in \mathbb{R}^{3\times4\times2}$ is a 3^{rd}-order tensor with the following two frontal slices:

$$\mathbf{T}_{::1} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \mathbf{T}_{::2} = \begin{bmatrix} 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{bmatrix}$$

Then, the following are matricisation of $\mathcal{T}$ in the first, second and the third modes respectively:

$$\mathbf{T}_{(1)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 13 & 14 & 15 & 16 \\ 5 & 6 & 7 & 8 & 17 & 18 & 19 & 20 \\ 9 & 10 & 11 & 12 & 21 & 22 & 23 & 24 \end{bmatrix}$$

$$\mathbf{T}_{(2)} = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{bmatrix}$$

$$\mathbf{T}_{(3)} = \begin{bmatrix} 1 & 5 & 9 & 2 & 6 & 10 & 3 & 7 & 11 & 4 & 8 & 12 \\ 13 & 17 & 21 & 14 & 18 & 22 & 15 & 19 & 23 & 16 & 20 & 24 \end{bmatrix}$$

Another useful tensorial operation is multiplication which includes tensor-tensor, tensor-matrix and tensor-vector multiplications. Though quite similar in principle, tensor multiplication can be much more complex than matrix/vector multiplication. There are various ways of multiplying tensor with other tensors of similar or different orders, however, here we only discuss the *n-mode matrix product of a tensor* which is the only multiplication operation directly relevant to this study. Given a N^{th}-order tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ and a matrix $\mathbf{M} \in \mathbf{R}^{J \times I_n}$, the n-mode multiplication (denoted by $\mathcal{T} \times_n \mathbf{M}$) results in a N^{th}-order tensor $I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. The resulting tensor can be expressed as in the following equation element wise.

$$(\mathcal{T} \times_n \mathbf{M})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} t_{i_1 \cdots i_N} m_{j i_n} \tag{2.5}$$

For instance, given the tensor $\mathcal{T}$ in the example above and the matrix $\mathbf{M}$ below:

$$\mathbf{M} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

The multiplication of tensor $\mathcal{T}$ and matrix $\mathbf{M}$ on the first mode ($\mathcal{T} \times_1 \mathbf{M}$) results in a new tensor $\mathcal{Y}$ with the following two frontal slices respectively:

$$\mathbf{Y}_{::1} = \begin{bmatrix} 61 & 70 & 79 & 88 \\ 76 & 88 & 100 & 112 \end{bmatrix}$$

$$\mathbf{Y}_{::2} = \begin{bmatrix} 169 & 178 & 187 & 196 \\ 220 & 232 & 244 & 256 \end{bmatrix}$$

One of the operators used in tensor calculations (particularly during factorisation) is the *Khatri-Rao* product ([174] and [171]). Where matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$ are given, their Khatri-Rao product (denoted by $\mathbf{A} \odot \mathbf{B}$) is a matrix of size $(IJ) \times K$ defined by:

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_2 \otimes \mathbf{b}_2 \quad \cdots \quad \mathbf{a}_K \otimes \mathbf{b}_K] \tag{2.6}$$

where $\otimes$ is the Kronecker product. An example of the Kronecker product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is given below.

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^T = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \\ a_4b_1 & a_4b_2 & a_4b_3 \end{bmatrix}$$

Further practical examples of the operators mentioned above and description of many other operators can be found in [175] and [171].

The operation widely used to extract the latent relationship between various modes of data is factorising (decomposing) the tensor into its *basic factors*. Basic factors are then used in different ways depending on the objective of the algorithm. The tensor decomposition methods are mainly generalisations of the Singular Value Decomposition (SVD) to higher dimensions. Higher Order SVD (HOSVD) [162], Tucker decomposition [163], Parallel Factor (a.k.a PARAFAC or CANDECOMP or CP) [164] and Non-negative Tensor Factorisation (NTF) [165] are among numerous factorisation methods proposed by researchers. A description of the two most popular factorisation methods, namely, the CP factorisation approach and the Tucker method is given in the following sections.

2.3.3.2 CP Factorisation

Assume that we have a 3^{rd}-order tensor denoted by $\mathcal{T}$ with a size $P \times Q \times R$. CP decomposition utilizes the Alternating Least Square (ALS) algorithm [176, 177] (Algorithm 5), in which tensor $\mathcal{T}$ is approximated by another tensor $\hat{\mathcal{T}}$ as in Equation 2.7.

$$\hat{\mathcal{T}} = \sum_{k=1}^{K} \lambda_k \ \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k \tag{2.7}$$

$\lambda_k \in \mathbb{R}_+$, $\mathbf{a}_k \in \mathbb{R}^P$, $\mathbf{b}_k \in \mathbb{R}^Q$ and $\mathbf{c}_k \in \mathbb{R}^R$ for $k = 1 \cdots K$, where K is the number of desired components. Each summand ($\lambda_k \ \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k$) is called a component while the individual vectors are called factors (Figure 2.3). λ_k is the weight of the kth component achieved by normalizing the vectors $\mathbf{a}_k$, $\mathbf{b}_k$ and $\mathbf{c}_k$. Most tensor toolboxes (including the Matlab Tensor Toolbox [206] used in this study) sort the resulting components in decreasing order of their weights.

Note that "$\circ$" is the outer product operator. The outer product of three vectors produces a 3^{rd}-order tensor. For instance, given $K = 1$, Equation 2.7 reduces to $\hat{\mathcal{T}} = \lambda \ \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$, in which $\hat{\mathcal{T}}$ is a 3^{rd}-order tensor which is obtained by the outer product of three vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. Subsequently, each tensor entry, denoted as $\hat{t}_{pqr}$ is computed through a simple multiplication like $a_p b_q c_r$.

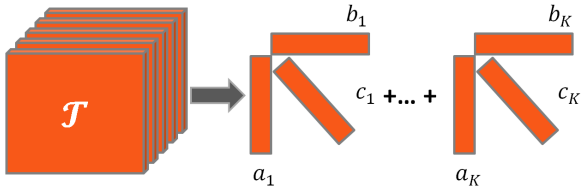


FIGURE 2.3: Factorising a tensor to K components.

The outer product $\mathbf{a}_k \circ \mathbf{b}_k$ in Equation 2.7 quantifies the relationship between the object pairs, i.e., score values indicating the "level of interaction" between object pairs in component k [171]. The purpose of the ALS algorithm is to minimize the error difference between the original tensor and the estimated tensor, denoted as ε as follows:

$$\varepsilon = \frac{1}{2} ||\mathcal{T} - \hat{\mathcal{T}}||_F^2 \tag{2.8}$$

where the subscript F refers to the Frobenious norm. That is:

$$||\mathcal{T} - \hat{\mathcal{T}}||_F^2 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \left(t_{pqr} - \hat{t}_{pqr}\right)^2 \tag{2.9}$$

The ALS algorithm (Algorithm 5) operates as follows. The algorithm takes a tensor and the desired number of components as input (lines 1 and 2). The algorithm can also be provided with a maximum number of iteration (line 3), though, modern ALS algorithms rarely need this input as they terminate the loop whenever there is no more

Algorithm 5: The Alternating Least Square algorithm - CP decomposition of a 3^{rd}-order tensor

1 In: $\mathcal{T} \in \mathbb{R}^{P \times Q \times R}$ (the tensor to be factorized);
2 In: K (number of desired components);
3 In: M (number of iterations);
4 $\mathbf{A}^{(0)} \in \mathbb{R}^{P \times K}, \mathbf{B}^{(0)} \in \mathbb{R}^{Q \times K}, \mathbf{C}^{(0)} \in \mathbb{R}^{R \times K} \leftarrow$ Initial guess;
5 **for** *(*$i = 1 \cdots M$*)* **do**
6 $\quad \mathbf{A}^{(i+1)} \leftarrow \mathbf{T}_{(1)} / (\mathbf{C}^{(i)} \odot \mathbf{B}^{(i)})^T$ (solving least square to update $\mathbf{A}$);
7 $\quad \mathbf{B}^{(i+1)} \leftarrow \mathbf{T}_{(2)} / (\mathbf{C}^{(i)} \odot \mathbf{A}^{(i+1)})^T$ (solving least square to update $\mathbf{B}$);
8 $\quad \mathbf{C}^{(i+1)} \leftarrow \mathbf{T}_{(3)} / (\mathbf{B}^{(i+1)} \odot \mathbf{A}^{(i+1)})^T$ (solving least square to update $\mathbf{C}$);
9 $\quad$ normalize columns of $\mathbf{A}^{(i+1)}, \mathbf{B}^{(i+1)}$ and $\mathbf{C}^{(i+1)}$ (storing norms as λ);
10 **end**
11 return $\lambda, \mathbf{A}^M, \mathbf{B}^M, \mathbf{C}^M$

improvement in the approximation error (Eq.2.8). The algorithm then generates initial guesses for the three factor matrices $\mathbf{A}^{(0)}$, $\mathbf{B}^{(0)}$ and $\mathbf{C}^{(0)}$. Starting from these initial guesses, the ALS algorithm fixes $\mathbf{B}$ and $\mathbf{C}$ and solves for $\mathbf{A}$ (line 6) where $\mathbf{T}_{(1)}$ is the mode-1 matricisation (definition: Section 2.3.3.1) of the given tensor and the solution to the least square problem is expressed by the following equation:

$$\mathbf{A}^{i+1} = \underset{\hat{A} \in \mathbb{R}^{P \times K}}{\operatorname{argmin}} \parallel \mathbf{T}_{(1)} - \hat{A}\big(\mathbf{C}^i \odot \mathbf{B}^i\big)^T \parallel_F^2 \tag{2.10}$$

Subsequently, the algorithm fixes the updated $\mathbf{A}^{i+1}$ and the existing $\mathbf{C}$ matrices to update the matrix $\mathbf{B}$ (line 7) with improved values where these values are achieved by solving the least square problem given in Equation 2.11.

$$\mathbf{B}^{i+1} = \underset{\hat{B} \in \mathbb{R}^{Q \times K}}{\operatorname{argmin}} \parallel \mathbf{T}_{(2)} - \hat{B}\big(\mathbf{C}^i \odot \mathbf{A}^{i+1}\big)^T \parallel_F^2 \tag{2.11}$$

Similar to the last two steps, updated $\mathbf{A}^{i+1}$ and $\mathbf{B}^{i+1}$ are fixed and used to solve the least square problem in Equation 2.12 to generate an updated $\mathbf{C}^{i+1}$ (line 8).

$$\mathbf{C}^{i+1} = \underset{\hat{C} \in \mathbb{R}^{R \times K}}{\operatorname{argmin}} \parallel \mathbf{T}_{(3)} - \hat{C}\big(\mathbf{B}^{i+1} \odot \mathbf{A}^{i+1}\big)^T \parallel_F^2 \tag{2.12}$$

This cycle continues until some convergence criterion is met or the maximum number of iterations is reached. At each iteration the factor vectors of each factor matrix are normalized to length 1 vectors and the weight of each factor is inserted into its corresponding location in the weight vectors λ.

After applying the factorisation method, in addition to the CP model, a measure of how well the original data is described by the factorized tensor can be obtained. This measure (*model fitness*), denoted as ϕ, is computed using the following equation:

$$\phi = 1 - \frac{||\mathcal{T} - \hat{\mathcal{T}}||_F}{||\mathcal{T}||_F} \tag{2.13}$$

where the value ϕ is in the range $[0, 1]$. The ϕ value is an indicator which shows how close the approximated tensor is to the original data. In other words, it measures the proportion of the original data represented in the approximated tensor $\hat{\mathcal{T}}$. A perfect factorisation (where $\mathcal{T} = \hat{\mathcal{T}}$) results in $\phi = 1$, while a poor factorisation results in $\phi = 0$, hence as the ϕ value increases, so does the factorisation accuracy. More on tensor decompositions and their applications can be found in [171].

As well as representing data in a concise and more generalizable manner which is immune to data anomalies (such as missing data), tensor factorisation methods offer additional interesting utilities. Those methods allow partitioning of the data into a set of more comprehensible sub-data which can be of specific use depending on the application according to various criteria. For example, in the field of computer vision, [178] and [153] separately show that factorisation of a 3^{rd}-order tensor of a video sequence results in some interesting basic factors. These basic factors reveal the location and functionality of human body parts which move synchronously.

2.3.3.3 Tucker Factorisation

Tucker decomposition was first introduced in [163], [179] and [180] and is a generalisation of the CP method with few modifications. Tucker decomposition is a higher order PCA and has it's applications in data compression [166], dimensionality reduction [149] and noise removal [167]. Unlike CP factorisation in which factorising a 3^{rd}-order tensor results in vectors as basic factors of each component, in the Tucker approach, decomposition of a 3^{rd}-order tensor results in a core tensor ($\mathcal{G}$) which is transformed by a matrix along each mode of the data (Figure 2.4). Thus, using Tucker decomposition method, a 3^{rd}-order tensor $\mathcal{T}$ of size $P \times Q \times R$ is approximated by the tensor $\hat{\mathcal{T}}$ as in the following equation.

$$\hat{\mathcal{T}} = \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r \tag{2.14}$$

where $\times_n$ refers to multiplication along mode n (definition: Section 2.3.3.1). $A \in \mathbb{R}^{I \times P}$, $B \in \mathbb{R}^{J \times Q}$ and $A \in \mathbb{R}^{K \times R}$ are factor matrices and can be thought of as the principal component in each mode. The entries of the core tensor $\mathcal{G} \in \mathbb{R}^{I \times J \times K}$, say g_{ijk}, represent the level of interaction between different principal components.

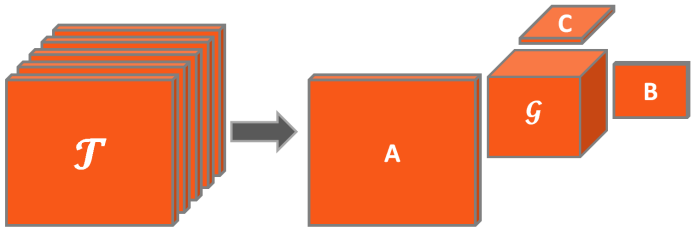


FIGURE 2.4: Factorising a tensor using the Tucker decomposition method.

2.3.3.4 Tucker vs. CP Decomposition

Perhaps the best way to compare the two popular factorisation approaches is to contrast their approximations of the original tensor. Contrasting Eq.2.7 (CP) with Eq.2.14 (Tucker), it is evident that the former provides a simple to understand approximation of the tensor. The CP factorisation produces three basic factors whereas the Tucker decomposition includes a core tensor ($\mathcal{G}$) which is difficult to understand and interpret. On the other hand, an interesting property of the CP decomposition method is that its unique (under mild conditions) [171]. That is, the rank one tensors achieved (i.e. each tensor in the right hand side of the Eq.2.7) are the only possible combination that sum up to the original tensor. This is while the Tucker decomposition does not provide a unique approximation of the original tensor.

In our study, as we shall see in future sections, we use tensor analysis for ranking and partitioning purposes. Thus, uniqueness of the basic factors is a most crucial condition. Moreover, compared to Tucker decomposition, CP factorisation produces *easy to interpret* basic factors. This is particularly desirable in the context of heuristics. Some applications (such as video, speech and text) produce easy-to-understand contents because there is a certain semantic concept naturally associated with the data. This is not the case in the data produced by hyper-heuristics. Therefore, it is very desirable to choose a decomposition method which offers some level of simplicity in the process of interpreting the basic factors. Therefore, we have chosen to use the CP decomposition as the method to factorize tensors in this study.

2.4 Machine Learning Improved Heuristic Optimisation

Using machine learning techniques to improve the performance of search algorithms is not a new strategy. In continuous optimisation where the variables in objective function take real numbers as values, machine learning approaches have been used to influence certain aspects of the optimisation algorithm. The learning can be applied in different ways influencing different components of the evolutionary algorithms such as population initialisation [181, 182] (using orthogonal experimental design and opposition-based learning respectively), modelling objective function [183] (using artificial neural networks), reducing number of function evaluations [184] (using Gaussian processes and surrogate models), problem scale reduction [185] (using principal component analysis), problem structure learning [186] (using Bayesian networks), parameter adaptation [187] (using cluster analysis) and operator adaptation [188] (using reinforcement learning respectively). More about integrating machine learning methods in optimisation algorithms in continuous domain can be found in [189].

In this study however, we focus on combinatorial (discrete) optimisation where the variables of the objective function take on discrete values. Optimisation algorithms in discrete domain have also considered using machine learning to improve their performance [30–35, 190, 191]. However, compared to the continuous domain, the number of studies in this area is a lot less. Although previous studies are certainly valuable introducing a very powerful technique in heuristic optimisation, and provide insights which have led to the work presented in this dissertation, they usually suffer from few drawbacks and fail to satisfy some of the criteria enlisted below.

- **Generality**: Often, machine learning algorithms are combined with heuristics to improve their performance on a sub-class of instances of a single problem domain. While this may achieve a satisfying result for a particular study there is no evidence of sufficient generality over various problem domains.
- **Domain Independent Data**: Data used for various machine learning algorithms are usually domain dependent [32, 120]. It is not clear if the same approach can be used when the dataset evolves even if the problem stays the same. It seems that in most cases, the learning algorithm changes as the data changes. For instance, following the sequence of research in [34, 35, 120]) it is clear that the mining technique needs revision when the problem/data changes. This on its own is not a deficiency and the studies in [34, 35, 120] present valuable conclusions which encourages us to choose the right technique in our approach.

- **Performance**: Sometimes, a machine learning technique is embedded into the heuristic optimisation process, however, it does not perform well at all [130, 136]. Looking at the lowest ranking hyper-heuristics of the CHeSC 2011 competition in Table 2.3 it becomes clear that some of the most well-known machine learning algorithms such as Reinforcement Learning or Markov Decision Processes fail to provide hyper-heuristics with a reasonably good level of adaptation. These algorithms are by no means considered as weak methodologies in the pattern recognition community. In fact, they are very powerful and numerous studies exist in the literature to prove this point and illustrate their popularity.

- **Agility**: Training episodes usually take a considerable time and even sometimes mixed with parameter tuning [117]. Long training scenarios most often result in more accurate predictive systems for the problem instances used in the training stage. However, this in turn can lead to over-fitting and a poor performance on unseen problem instances. Also, long training periods are not useful in real-time problems where a quick sub-optimal solution is preferred to a better solution achieved after a long run time.

- **Originality**: In some cases, the claim is that a specific machine learning approach is used, however, the learning methodology is over-simplified to the point that it no longer resembles/follows the original learning method. As an example, the study in [192] proposed a hyper-heuristic which uses reinforcement learning for adaptation. What the algorithm really does however, is simply exhibiting a greedy behaviour with respect to some rewarding scheme. Reinforcement learning in its true form can be found in [18] and compared to what has been proposed.

- **Data Abstraction Levels**: Machine learning algorithms used in the context of heuristics optimisation are not applied to data with various levels of abstraction. This is useful since it shows the flexibility of the learning approach. An off-the-shelf learning approach which can be integrated in a wide variety of heuristic optimisation methods regardless of their underlying design philosophy and improve their performance should be more appreciated than scattered use of different methods in different optimisation algorithms.

- **Novelty**: There are various novel and high performance machine learning approaches in the literature with very interesting properties. For example, apart from the technique which is used in this study for the first time, interesting advanced machine learning methods such as Conditional Random Fields [193], Deep Learning [194] and Deep Reinforcement Learning [195] could be investigated. Machine learning algorithms are getting smarter and more efficient every day. It would be scientifically interesting to use these new methods along with classical

mining techniques to see whether interesting patterns which were previously unknown exist. Most of the optimisation algorithms to date, rarely use these novel methods and often opt for the classical approaches.

A proper integration of machine learning approaches with heuristic optimisation techniques, should consider all of the above mentioned criteria and ideally satisfy them all. To further clarify these issues a review on the heuristic optimisation approaches which use machine learning at some stage during the search is given here.

In one of the rare studies where authors have tested their learning optimisation approach on multiple domains, [30] proposes an offline learning method to construct rules using which the algorithm escapes local optima. In order to construct the dataset, first, a set of randomly chosen local optima points are considered. The following procedure is then followed for each point A in the set of randomly chosen local optima points. For each local optima A, close local optima points B are chosen. The proximity is calculated using the Euclidean distance metric between two points and a distance threshold is used to select *close* local optima for each given point. Subsequently, pairs in the form of (A, B) are constructed for each local optima A and points B which are close to A. If the value of the objective function value improves when moving from A to B, then the pair (A, B) is regarded as an improving pair. Otherwise, it is labelled as a non-improving pair. Furthermore, if the number of local optima points close to A are less than a given lower bound then the point A is discarded all together. The reason for this strategy is that a local optima with few local optima in it's proximity does not offer much clue as to how to escape it. Constructing all the pairs for each given local optima results in a paired dataset. The learning process, based on binary classification approach, is formulated as a mathematical programming problem which outputs a set of guiding rules. Using these rules, the algorithm is guided to escape a local optimum in unseen problem instances. The experiments on two problems, namely, Constrained Task Allocation Problem and the matrix bandwidth minimisation problem show that by using the learned rules, a simple tabu search algorithm achieves competitive results compared to the state-of-the-art approaches in both problem domains.

In [31] several classification approaches are used to mine the trace of a Peckish hyper-heuristic [196] on instances of the training scheduling problem [197], [198]. The extracted knowledge is then tested on other runs of the same problem domain. The training set consists of several domain dependent features (domain barrier is not considered here) and is multi-labelled. The classifiers considered in this study are either variants of decision trees or associative classification algorithms. More specifically Partial Decision Trees (PART) [199], the RIPPER algorithm [200], Classification Based on Associations (CBA) approach [201], Multi-class Classification based on Association Rules (MCAR) [202]

algorithm and Multi-class Multi-label Associative Classification (MMAC) approach [203] are compared to each other. The results show that the MMAC approach performed best due to its acknowledgement of multiple class values per dataset record. Furthermore, it was observed that the decision tree approach constructs trees with unnecessary branches. The authors claim that this unnecessary branching is related to the unbalanced nature of their dataset. Also, it is not clear why a multi-label classification problem has been subjected to approaches such as decision trees while methods specifically designed for this sort of data could be considered and compared to the method proposed by the authors.

In [32] heuristic policies are learned using a reinforcement learning algorithm. That is, the bin packing problem is re-formulated as a temporal difference learning scheme [18]. A state description is provided for both online and offline bin packing problems. A value function representing the average performance of the learned heuristic over the problem domain is approximated using temporal difference learning. The experimental results show that this algorithm, though not very powerful when compared to some well known methods, produces good quality solutions.

In [33] machine learning is used to generate *behavioural search drivers* for a Genetic Programming (GP) method. It has been argued in this study that conventional fitness function in GP does not necessarily provide an ideal guidance. Instead, it has been proposed to use machine learning to complement the guidance provided by the fitness function. To this effect, a synthesized training dataset is produced. In order to produce the dataset synthetic program trees are generated. A simple random walk (using simple mutations) is applied to the tree which is assumed to be the optimal solution. Applying the random walk results in moving away from the original program tree. After the random walk is finished, the resulting tree is vectorized. This vector which naturally describes the program behaviour is treated as a feature vector and is added to the dataset. The label of the feature vector is the distance between the tree (which the feature vector represents) and the initial tree which is the expected output. Note that the label represents the expected number of steps to reach the optimal solution from the tree achieved after the random walk. The C4.5 algorithm [139] is applied on the dataset resulting in a classifier. The properties of the classifier is then used to define the behavioural fitness of unseen program trees. The experimental results show that the behavioural guidance achieves its objective.

Machine learning enhanced heuristics have also been considered for Constrained Satisfaction Problems (CSP). In a series of studies, Learning Classifier Systems (LCS) [35], back propagation neural networks citeOrtiz-BaylissTRC11 and logistic regression [34] are separately used to generate selection hyper-heuristic for CSP.

2.5 Summary

In this section, definitions, basic concepts and explanation regarding the techniques which have been used throughout this study were covered. This paves the way for studying the role of tensor analysis in heuristic optimisation. The first such application of tensor learning in the field of heuristic research is thus presented in the next chapter where tensor analysis is used to improve the performance of a simple hyper-heuristic.

Chapter 3

A Tensor-based Selection Hyper-heuristic for Cross-domain Heuristic Search

The search history formed by a heuristic, metaheuristic or hyper-heuristic methodology constitutes a multi-dimensional data. For example, when populations of several generations of individuals in a Genetic Algorithm (GA) are put together, the emerging structure representing the solutions and associated objective values changing in time is a third order tensor. Similarly, the interaction between low level heuristics as well as the interaction between those low level heuristics and the acceptance criteria under a selection hyper-heuristic framework are a couple of examples of various modes of functionality in a tensor representing the search history. This chapter represents a method which captures the trail of a hyper-heuristic in the form of a third order tensor and analyzes it to detect the latent relationship between low level heuristics and the hyper-heuristic. A multi-stage hyper-heuristic is then built which uses these latent patterns to perform search on various instances of several problem domains taken from the HyFlex framework.

The data captured from the hype-heuristic is highly abstract and is constructed using indexes of low level heuristics chosen by the underlying selection hyper-heuristic and the objective function values achieved during the search. Complex structures such as candidate solution representation and neighbourhood information are thus missing. During discussions on the experimental results we will show that the proposed method is able to deal with high level of abstraction in data and extract useful patterns using which the performance of a very primitive and simple hyper-heuristic is significantly improved.

3.1 Introduction

Many adaptive (hyper-/meta) heuristics make use of their search history to construct models which are used to improve their performance. Algorithms such as reinforcement-based hyper-heuristics or hyper-heuristics embedding late acceptance or tabu search components are few examples. The performance of such algorithms is usually confined with memory restrictions. Moreover, the memory often contains raw data, such as objective values or visited states and those components ignoring the hidden clues and information regarding the choices that influences the overall performance of the approach in hand.

In this chapter, a tensor-based selection hyper-heuristic is proposed. In the proposed approach, we represent the trail of a selection hyper-heuristic as a 3^{rd} order tensor to represent the search history of a hyper-heuristic. The first two modes of the tensor are indexes of subsequent low level heuristics selected by the underlying hyper-heuristic while the third mode is the time. Having such a tensor filled with the data acquired from running the hyper-heuristic for a short time and decomposing it, hopefully reveals the indices of low level heuristics which are performing well with the underlying hyper-heuristic and acceptance criteria. This is very similar to what has been done in human action recognition in videos using tensor analysis [153], except that, instead of examining the video of human body motion and looking for different body parts moving in harmony, we records the trace of a hyper-heuristic (body motion) and look for low level heuristics (body parts) performing harmoniously. Naturally, our ultimate goal is to exploit this knowledge for improving the search process.

Tensor analysis is performed during the search process to detect the latent relationships between the low level heuristics and the hyper-heuristic itself. The feedback is used to partition the set of low level heuristics into two equal subsets where heuristics in each subset are associated with a separate move acceptance method. Then a multi-stage hyper-heuristic combining a random heuristic selection with two simple move acceptance methods is formed. While solving a given problem instance, heuristics are allowed to operate only in conjunction with the corresponding move acceptance method at each alternating stage. This overall search process can be considered as a generalized and a non-standard version of the iterated local search [204] approach in which the search process switches back and forth between diversification and intensification stages. More importantly, the heuristics (operators) used at each stage are fixed before each run on a given problem instance via the use of tensors. To the best of our knowledge, this is the first time tensor analysis of the space of heuristics is used as a data science approach to improve the performance of a selection hyper-heuristic in the prescribed manner.

The empirical results across six different problem domains from the HyFlex benchmark indicate the success of the proposed hyper-heuristic mixing different acceptance methods.

3.2 Proposed Approach

The low level heuristics in HyFlex are divided into four groups as described in Section 2.2.3. The heuristics belonging to the mutational (MU), ruin-recreate (RR) and local search (LS) groups are unary operators requiring a single operand. This is while, the crossover operators have two operands requiring two candidate solutions to produce a *new* solution. In order to maintain simplicity as well as coping with the single point search nature of our framework, crossover operators are ignored in this chapter and MU, RR and LS low level heuristics are employed. The set of all available heuristics (except crossover operators) for a given problem domain is denoted by a lower-case bold and italic letter $\boldsymbol{h}$ throughout this section. Moreover, from now on, we refer to our framework as Tensor-Based Hybrid Acceptance Hyper-heuristic (TeBHA-HH) which consists of five consecutive phases: (i) noise elimination (ii) tensor construction, (iii) tensor factorisation, (iv) tensor analysis, and (v) hybrid acceptance as illustrated in Figure 3.1. The noise elimination filters out a group of low level heuristics from $\boldsymbol{h}$ and then a tensor is constructed using the remaining set of low level heuristics, denoted as $\boldsymbol{h}^{-}$. After tensor factorisation, sub-data describing the latent relation between low level heuristics is extracted. This information is used to divide the low level heuristics into two partitions: $\boldsymbol{h}_{NA}$, $\boldsymbol{h}_{IE}$. Each partition is then associated with a move acceptance method, that is naive move acceptance with $\alpha = 0.5$ (NA) or improving and equal moves (IE) respectively. This is equivalent to employing two selection hyper-heuristics, Simple Random-Naive move acceptance (SR-NA) and Simple Random-Improving and Equal (SR-IE). Each selection hyper-heuristic is invoked in a round-robin fashion for a fixed duration of time (t_s) using only the low level heuristics associated with the move acceptance component of the hyper-heuristic at work ($\boldsymbol{h}_{NA}$ and $\boldsymbol{h}_{IE}$, respectively) until the overall time limit (T_{max}) is reached. This whole process is repeated at each run while solving a given problem instance. All the problems dealt with in this chapter are minimising problems. A detailed description of each phase is given in the subsequent sections.

3.2.1 Noise Elimination

We model the trace of the hyper-heuristic as a tensor dataset and factorize it to partition the heuristic space. Tensor representation gives us the power to analyse the latent relationship between heuristics. But this does not mean that any level and type of noise is welcome in the dataset. The noise in the dataset may even obscure the existing latent

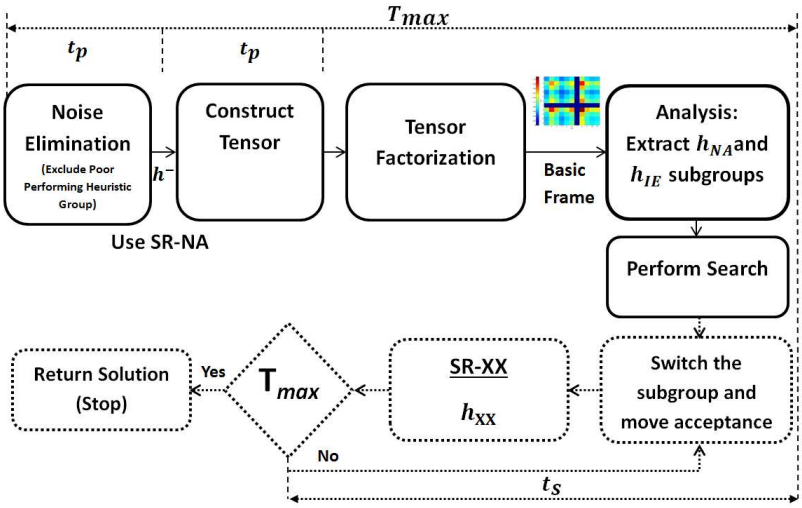


FIGURE 3.1: The schematic of our proposed framework.

structures. Thus, reducing the noise is a necessary step in constructing datasets. Under the selection hyper-heuristic framework in which low level heuristics are chosen randomly, if the heuristics of a certain type (e.g. mutation, ruin-recreate, etc) consistently generate highly worsening solutions, then such a heuristic group is considered as *poor performing*, causing partial re-starts which is often not a desirable behaviour. Hence, the tensor dataset produced while heuristics belonging to such a heuristic group are used can be considered *noisy* (noise is generally defined as undesired data) and that heuristic type can be treated as source of the noise. Thus, in our proposed approach, using the methodology explained below, we identify the heuristic group which causes noise at the start and eliminate it to create a less noisy dataset.

The type of noise happens to be very important in many data mining techniques and tensor factorisation is not an exception. CP factorisation method which is one of the most widely used factorisation algorithms, assumes a Gaussian type noise in the data. It has been shown that CP is very sensitive to non-Gaussian noise types [205]. In hyper-heuristics, change in the objective value after applying each heuristic follows a distribution which is very much dependent on the problem domain and the type of the heuristic, both of which are unknown to a hyper-heuristic and unlikely to follow a Gaussian distribution. To the best of our knowledge, while there are not many factorisation methods which deal with various types of noise in general, there is no method tailored for heuristics. Thus, it is crucial to reduce the noise as much as possible prior to any analysis of the data. This is precisely the aim of the first phase of our approach.

Excluding the crossover heuristics leaves us with three heuristic groups (MU, RR and LS). A holistic strategy is used in the noise elimination phase for getting rid of poor heuristics. An overall pre-processing time, denoted as t_p is allocated for this phase.

Except the LS group, applying heuristics belonging to all other groups may lead to worsening solutions. Hence, the *worst* of the two remaining heuristic groups (MU and RR) is excluded from the subsequent step of the experiment. In order to determine the group of low level heuristics to eliminate, SR-NA is run using the MU and LS low level heuristics for a duration of $t_p/2$, which is followed by another run using RR and LS low level heuristics for the same duration. Performing the pre-processing tests in this manner also captures the interaction between perturbative and local search heuristics under the proposed framework for improvement. During each run, the search process is initiated using the same candidate solution for a fair comparison. The quality of the solutions obtained during those two successive runs are then compared. Whichever group of low level heuristics generates the worst solution under the described framework gets eliminated. The remaining low level heuristics, denoted as $\boldsymbol{h}^-$ is then fed into the subsequent phase for tensor construction.

3.2.2 Tensor Construction and Factorisation

We represent the trail of SR-NA as a 3^{rd}-order tensor $\mathcal{T} \in \mathbb{R}^P \times \mathbb{R}^Q \times \mathbb{R}^R$ in this phase, where $P = Q = |\boldsymbol{h}^-|$ is the number of available low level heuristics and $R = N$ represents the number of tensor frames collected in a given amount of time. Such a tensor is depicted in Figure 3.2. The tensor $\mathcal{T}$ is a collection of two dimensional matrices ($\mathbf{M}$) which are referred to as *tensor frames*. Therefore each tensor frame is a frontal slice as in Figure 2.2(f). A tensor frame is a two dimensional matrix of heuristic indices. Column indices in a tensor frame represent the index of the current heuristic whereas row indices represent the index of the heuristic chosen and applied before the current heuristic.

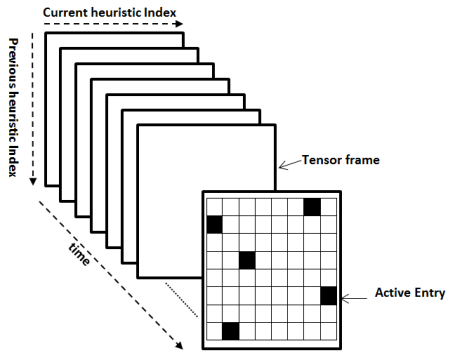


FIGURE 3.2: The tensor structure in TeBHA-HH. The black squares (also referred to as active entries) within a tensor frame highlight heuristic pairs invoked subsequently by the underlying hyper-heuristic.

The tensor frame is filled with binary data as demonstrated in Algorithm 6. The bulk of the algorithm is the SR-NA hyper-heuristic (starting at the while loop in line 4).

Iteratively, a new heuristic is selected randomly (line 12) and applied to the problem instance (line 14). This action returns a new objective value f_{new} which is used together with the old objective value f_{old} to calculate the immediate change in the objective value (δ_f). The algorithm then checks if $\delta_f > 0$ indicating improvement, in which case the solution is accepted. Otherwise, it is accepted with probability 0.5 (line 21). While accepting the solution, assuming that the indices of the current and previous heuristics are $h_{current}$ and $h_{previous}$ respectively, the tensor frame $\mathbf{M}$ is updated symmetrically: $m_{h_{previous},h_{current}} = 1$ and $m_{h_{current},h_{previous}} = 1$. The frame entries with the value 1 are referred to as active entries. At the beginning of each iteration, the tensor frame $\mathbf{M}$ is checked to see if the number of active entries in it has reached a given threshold of $\lfloor |\boldsymbol{h}|/2 \rfloor$ (line 5). If so, the frame is appended to tensor and a new frame is initialized (lines 6 to 9). This whole process is repeated until a time limit is reached which the same amount of time allocated for the pre-processing phase; t_p (line 4).

Algorithm 6: The tensor construction phase

```
1  In: h = h⁻;
2  Initialize tensor frame M to 0;
3  counter = 0;
4  while t < t_p do
5      if counter = ⌊|h|/2⌋ then
6          append M to T;
7          set frame label to Δ_f;
8          Initialize tensor frame M to 0;
9          counter = 0;
10     end
11     h_previous = h_current;
12     h_current = selectHeuristic(h);
13     f_current = f_new;
14     f_new =applyHeuristic(h_current);
15     δ_f = f_current − f_new;
16     if δ_f > 0 then
17         m_{h_previous,h_current} = 1;
18         counter + +;
19     else
20         if probability > 0.5 then
21             m_{h_previous,h_current} = 1;
22             if δ_f = 0 then
23                 m_{h_current,h_previous} = 1;
24             end
25             counter + +;
26         end
27     end
28 end
```

While appending a frame to the tensor, each tensor frame is labelled by the Δ_f it yields,

where Δ_f is the overall change in objective value caused during the construction of the frame. Δ_f is different from δ_f in the sense that the former measures the change in objective value inflicted by the collective application of active heuristic indexes inside a frame. The latter is the immediate change in objective value caused by applying a single heuristic. In other words, in a frame with a total of $\lfloor |\boldsymbol{h}|/2 \rfloor$ active entries (heuristic calls), Δ_f is calculated according to the following equation.

$$\Delta_f = \sum_{i=1}^{\lfloor |\boldsymbol{h}|/2 \rfloor} \delta_{f_i} \tag{3.1}$$

The aforementioned process constructs an initial tensor which contains all the tensor frames. However, we certainly do want to emphasize on intra-frame correlations as well. That is why, after constructing the initial tensor, the tensor frames are scanned for consecutive frames of positive labels($\Delta_f > 0$). In other words, a tensor frame is chosen and put in the final tensor only if it has a positive label and has at least one subsequent frame with a positive label. The final tensor is then factorized using the CP decomposition to the basic frame ($K = 1$ in Equation 2.7). The following section illustrates how the basic frame is used in our approach.

It is worth mentioning that the design of the tensor frames in this section is not the only possible one. Indeed, numerous other designs, including non-binary values for frame entries and different ways of associating search semantics with each mode of the tensor are possible. The influence of these design issues can (and perhaps should) be studied. However, since this work is the first to introduce tensor analysis to the field of heuristic optimisation, rather than dealing with design issues, we are more interested in investigating the effects that this learning technique has in improving the performance of a (hyper-/meta) heuristic. As such, we leave investigation of different tensor designs and their implications in heuristic optimisation to a different study in a future time.

3.2.3 Tensor Analysis: Interpreting The Basic Frame

Following the tensor factorisation process, the tensor is decomposed into basic factors. $K = 1$ in our case, thus, the Equation 2.7 reduces to the following equation:

$$\hat{\mathcal{T}} = \lambda \, \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \tag{3.2}$$

The outer product in the form $\mathbf{a} \circ \mathbf{b}$ produces a basic frame which is a matrix. Assume that, in line with our approach, we have constructed a tensor from the data obtained

from the hyper-heuristic search for a given problem instance in a given domain which comes with 8 low level heuristics, 6 of them being mutation and 2 of them being local search heuristics. Moreover, the factorisation of the tensor produces a basic frame as illustrated in Figure 3.3. The values inside the basic frame are the scores of each pair of low level heuristics when applied together. Since we are interested in pairs of heuristics which perform well together, we locate the pair which has the maximum value/score. In this example, the pair $(LS0, LS1)$ performs the best since the score for that pair is the highest. We regard these two heuristics as operators which perform well with the NA acceptance mechanism under the selection hyper-heuristic framework. The heuristics in the column index of this maximum pair (denoted by y in Algorithm 7) are then sorted (line 8 in Algorithm 7) to determine a ranking between the heuristics. Since heuristics $LS0$ and $LS1$ are already the maximising pair, they are considered to be the top two elements of this list. In the basic frame of Figure 3.3 the sorted list is then ($LS0$,$LS1$,$MU3$,$MU2$,$MU5$,$MU4$,$MU1$,$MU0$). The top/first $\lfloor |\boldsymbol{h}|/2 \rfloor$ elements of this list is then selected as those heuristics which perform well under NA acceptance mechanism ($\boldsymbol{h}_{NA}$). The remaining low level heuristics including the eliminated heuristics (e.g., RR heuristics) are associated with IE ($\boldsymbol{h}_{IE}$). The pseudo-code for partitioning is given in line 9 of Algorithm 7.

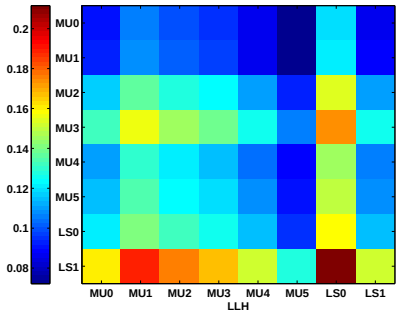


FIGURE 3.3: A sample basic frame. Each axis of the frame represents heuristic indexes. Higher scoring pairs of heuristics are darker in color.

3.2.4 Final Phase: Hybrid Acceptance

Selection hyper-heuristics have been designed mainly in two ways: ones which require the nature of low level heuristics to be known, while the others discard that information. For example, VNS-TW [122] and ML [123] assume that whether a given low level heuristic is mutational (or ruin-recreate) or local search is known, since they use only the relevant heuristics to be invoked at different parts of those algorithms. On the other hand, AdapHH [121] operated without requiring that information. Our hyper-heuristic

approach is of former type. Additionally, we ignore the crossover heuristics as many of the other previously proposed hyper-heuristics.

In our approach, we have considered a multi-stage selection hyper-heuristic which uses simple random heuristic selection and hybridizes the move acceptance methods, namely NA and IE (lines 10-27 of Algorithm 7). These selection hyper-heuristic components with no learning are chosen simply to evaluate the strength of the tensor-based approach as a machine learning technique under the proposed framework. This is the first time the proposed approach has been used in heuristic search as a component of a selection hyper-heuristic. In this phase, the best solution found so far is improved further using the proposed hyper-heuristic which switches between SR-NA and SR-IE. Since the same simple random heuristic selection method is used at all times, the proposed selection hyper-heuristic, in a way, hybridizes the move acceptance methods under the multi-stage framework. Each acceptance method is given the same amount of time; t_s to run. SR-NA operates using $\boldsymbol{h}_{NA}$, while SR-IE operates using $\boldsymbol{h}_{IE}$ as the low level heuristics. This search process continues until the time allocated to the overall hyper-heuristic (Algorithm 7) expires.

There are many cases showing that explicitly enforcing diversification (exploration) and intensification (exploitation) works in heuristic search. For example, there are many applications indicating the success of iterated local search (ILS) [204] and memetic algorithms (MAs) [78, 82]. Those metaheuristic approaches explicitly enforce the successive use of mutational and local search heuristics/operators in an attempt to balance diversification and intensification processes. The choice of NA and IE is also motivated by the reason that under the proposed framework, SR-NA can be considered as a component allowing diversification, while SR-IE focuses on intensification. Similar to ILS and MAs, the proposed approach also explicitly maintains the balance between diversification and intensification with the differences that SR-NA and SR-IE is employed in stages (not at each iteration) for a fixed period of time during the search process and the best subset of heuristics/operators that interact well to be used in a stage is determined through learning by the use of tensor analysis. Putting low level heuristics performing poorly with SR-NA under SR-IE somewhat ensures that those low level heuristics cause no harm misleading the overall search process.

3.3 Experimental Results

The experiments are performed on an Intel i7 Windows 7 machine (3.6 GHz) with 16 GB RAM. This computer was given 438 seconds (corresponding to 600 nominal seconds on the competition machine) as the maximum time allowed (T_{max}) per instance by the

Algorithm 7: Tensor-Based Hyper-heuristic with Hybrid Acceptance Strategy

let $\boldsymbol{h}$ be the set of all low level heuristics;
let t be the time elapsed so far;
$\boldsymbol{h}_x^- =$ exclude XO heuristics;
$\boldsymbol{h}^- =$ preProcessing$(t_p, \boldsymbol{h}_x^-)$;
$\mathcal{T} =$ constructTensor$(t_p, \boldsymbol{h}^-)$;
$\mathbf{a}, \mathbf{b}, \mathbf{c} = \text{CP}(\mathcal{T}, K = 1)$, $\mathbf{B} = \mathbf{a} \circ \mathbf{b}$;
$y = max(\mathbf{B})$;
$\boldsymbol{h}_s = sort(\mathbf{B}_{i=1:|\boldsymbol{h}^-|,y})$;
$\boldsymbol{h}_{NA} = (\boldsymbol{h}_s)_{i=1:\lfloor|\boldsymbol{h}_s|/2\rfloor}$, $\boldsymbol{h}_{IE} = \boldsymbol{h}_x^- - \boldsymbol{h}_{NA}$;
while $t < T_{max}$ **do**
  if $acceptance = NA$ **then**
    h = selectRandomHeuristic$(\boldsymbol{h}_{NA})$;
  else
    h = selectRandomHeuristic$(\boldsymbol{h}_{IE})$;
  end
  $s_{new}, f_{new} =$ applyHeuristic$(h, s_{current})$;
  $\delta = f_{old} - f_{new}$;
  updateBest(δ, f_{new});
  if $acceptanceTimer \geq t_s$ **then**
    toggle acceptance mechanism;
  end
  if $switch = true$ **then**
    $s_{new}, s_{old} =$NA$(s_{new}, f_{new}, s_{current}, f_{current})$;
  else
    $s_{new}, s_{old} =$IE$(s_{new}, f_{new}, s_{current}, f_{current})$;
  end
end

benchmarking tool provided by the CHeSC 2011 organizers. This is to ensure a fair comparison between various algorithms. We used the Matlab Tensor Toolbox [206] [1] for tensor operations. The HyFlex, as well as the implementation of our framework is in Java. Hence, subsequent to tensor construction phase, Matlab is called from within Java to perform the factorisation task. Throughout the chapter, whenever a specific setting of the TeBHA-HH framework is applied to a problem instance, the same experiment is repeated for 31 times, unless mentioned otherwise.

3.3.1 Experimental Design

One of the advantages of the TeBHA-HH framework is that it has considerably few parameters. To be more precise there are two parameters governing the performance of our tensor-based hybrid acceptance approach. The first parameter is the time allocated to pre-processing and tensor construction phases. This time boundary is equal

[1] `http://www.sandia.gov/~tgkolda/TensorToolbox/`

for both phases and is denoted as t_p. The second parameter is the time allowed to an acceptance mechanism in the final phase. During the final phase, the proposed selection hyper-heuristic switches between the two move acceptance methods (and their respective heuristic groups). Each move acceptance method is allowed to be in charge of solution acceptance for a specific time which is the second parameter and is denoted as t_s.

Note that all our experiments are conducted on the set of instances provided by CHeSC 2011 organizers during the final round of the competition. HyFlex contains more than a dozen of instances per problem domain. However, during the competitions 5 instances per domain were utilized. These are the set of instances which were employed in this chapter.

A preliminary set of experiments are performed in order to show the need for the noise elimination phase and more importantly to evaluate the performance of Automatic Noise Elimination (AUNE) in relation to the factorisation process fixing t_p and t_s values. This preliminary experiment along with experiments involving evaluation of various values of parameters t_p and t_s are only conducted on the first instance of each problem domain. After this initial round of tests in which the best performing values for the parameters of the framework are determined, a second experiment, including all the competition instances is conducted and the results are compared to that of the CHeSC 2011 competitors. Regarding the parameter t_p, values $15, 30$ and 60 seconds are tested. For t_s, values *nil* (0), 500, 1000 and 1500 milliseconds are experimented.

3.3.2 Pre-processing Time

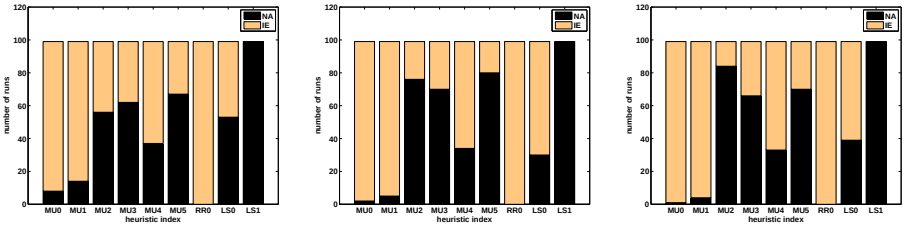
The experiments in this section concentrates on the impact of the time (t_p) given to the first two phases (noise elimination and tensor construction) on the performance of the overall approach. During the first phase in all of the runs, the *RR* group of heuristics have been identified as source of noise for Max-SAT and VRP instances. This is while, *MU* has been identified as source of noise for BP, FS and TSP instances. As for the PS domain, due to small number of frames collected (which is a result of slow speed of heuristics in this domain), nearly half of the time *RR* has been identified as the source of noise. In the remaining runs *MU* group of heuristics are excluded as noise. Our experiments show that for a given instance, the outcome of the first phase is persistently similar for different values of t_p.

The t_p value also determines the number of tensor frames recorded during the tensor construction phase. Hence, we would like to investigate how many tensor frames are *adequate* in the second phase. We expect that an adequate number of frames would result in a stable partitioning of the heuristic space regardless of how many times the

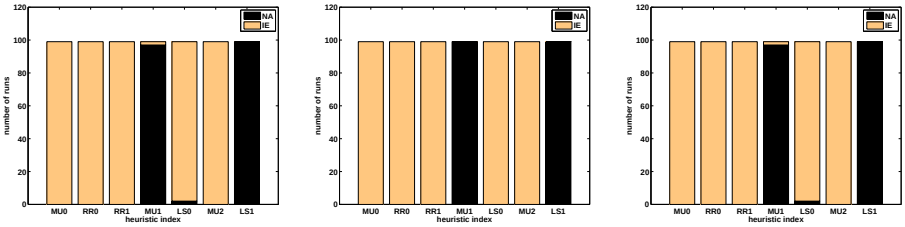
algorithm is run on a given instance. As mentioned in Section 3.3.1, three values (15, 30 and 60 seconds) are considered. Rather than knowing how the framework performs in the maximum allowed time, we are interested to know whether different values for t_p result in tremendously different heuristic subsets at the end of the first phase. Thus, in this stage of experiments, for a given value of t_p, we run the first two phases only. That is, for a given value of t_p we run the simple noise elimination algorithm. Subsequently, we construct the tensor for a given instance. At the end of these first two phases, the contents of $\boldsymbol{h}_{NA}$ and $\boldsymbol{h}_{IE}$ are recorded. We do this for 100 runs for each instance. At the end of the runs for a given instance, a histogram of the selected heuristics is constructed. The histograms belonging to a specific instance and achieved for various values of t_p are then compared to each other. Figures 3.4 compare the histograms of three different t_p values for the first instances of the 6 problem domains in HyFlex.

The histograms show the number of times a given heuristic is chosen as $\boldsymbol{h}_{NA}$ or $\boldsymbol{h}_{IE}$ within the runs. For instance, looking at the histograms corresponding to the Max-SAT problem (Figures 3.4(a), 3.4(b) and 3.4(c)), one could notice that the heuristic $LS1$ is always selected as $\boldsymbol{h}_{NA}$ in all the 100 runs. This is while $RR0$ is always assigned to $\boldsymbol{h}_{IE}$. The remaining heuristics are assigned to both sets although there is a bias towards $\boldsymbol{h}_{NA}$ in case of heuristics $MU2, MU3$ and $MU5$. A similar bias towards $\boldsymbol{h}_{IE}$ is observable for heuristics $MU0, MU1, MU4$ and $LS0$. This shows that the tensor analysis together with noise elimination adapts its decision based on the search history for some heuristics while for some other heuristics definite decisions are made. This adaptation is indeed based on several reasons. For one thing, some heuristics perform very similar to each other leading to similar traces. This is while, their performance patterns, though similar to each other, varies in each run. Moreover, there is an indication that there is no unique optimal subgroups of low level heuristics under a given acceptance mechanism and hyper-heuristic. There indeed might exists several such subgroups. For instance, there are two (slightly) different NA subgroups ($\boldsymbol{h}_{NA} = \{MU2, MU5, LS0, LS1\}$ and $\boldsymbol{h}_{NA} = \{MU2, MU3, LS0, LS1\}$) for the Max-SAT problem domain which result in the optimal solution ($f = 0$). This is strong evidence supporting our argument about the existence of more than one useful subgroups of low level heuristics. Thus, it only makes sense if the factorisation method, having several good options (heuristic subsets), chooses various heuristic subsets in various runs.

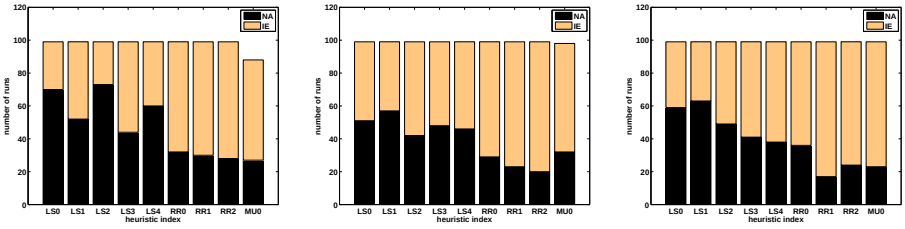
Interestingly, $RR0$ and $LS1$ are diversifying and intensifying heuristics respectively. Assigning $RR0$ to $\boldsymbol{h}_{IE}$ means that the algorithm usually chooses diversifying operations that actually improves the solution. A tendency to such assignments is observable for other problem instances, though not as strict as it is for BP and Max-SAT problem domains. While this seems to be a very conservative approach towards the diversifying



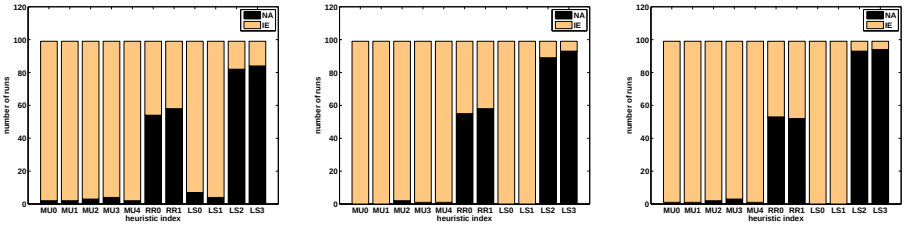
(a) Max-SAT, $t_p = 15sec$ (b) Max-SAT, $t_p = 30sec$ (c) Max-SAT, $t_p = 60sec$



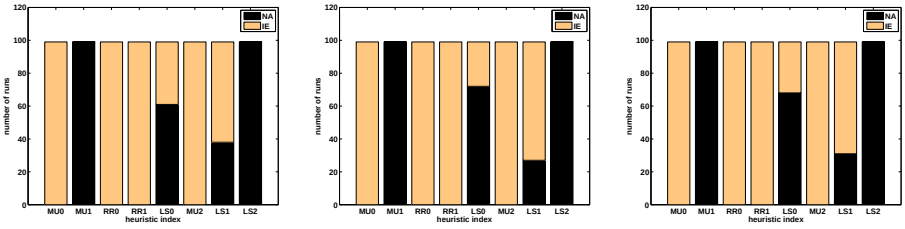
(d) BP, $t_p = 15sec$ (e) BP, $t_p = 30sec$ (f) BP, $t_p = 60sec$



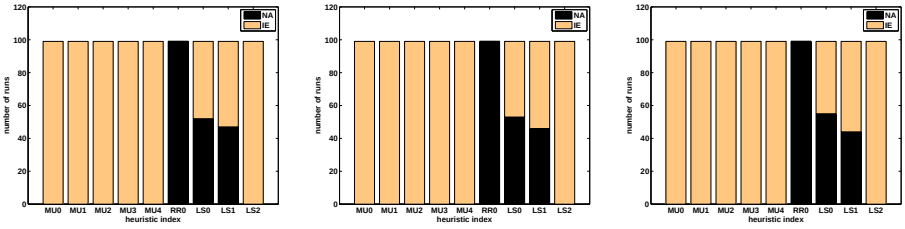
(g) PS, $t_p = 15sec$ (h) PS, $t_p = 30sec$ (i) PS, $t_p = 60sec$



(j) FS, $t_p = 15sec$ (k) FS, $t_p = 30sec$ (l) FS, $t_p = 60sec$



(m) VRP, $t_p = 15sec$ (n) VRP, $t_p = 30sec$ (o) VRP, $t_p = 60sec$



(p) TSP, $t_p = 15sec$ (q) TSP, $t_p = 30sec$ (r) TSP, $t_p = 60sec$

FIGURE 3.4: Histograms of heuristics selected as $\boldsymbol{h}_{NA}$ and $\boldsymbol{h}_{IE}$ for various t_p values across all CHeSC 2011 problem domains.

heuristics, as we will see later in this section, it often results in a *good* balance between intensification and diversification during the search.

In summary, the histograms show that the partitioning of the heuristic space is more or less the same regardless of the time allocated to t_p for a given problem instance. This pattern is observable across all CHeSC 2011 problem domains as illustrated in Figure 3.4. Longer run experiments, in which all the phases of the algorithm are included and the framework is allowed to run until the maximum allowed time is reached, confirms the conclusion that TeBHA-HH is not too sensitive to the value chosen for t_p. In Figure 3.5 a comparison between the three values for t_p is shown. The asterisk highlights the average performance. A comparison based on the average values shows that $t_p = 30$ is slightly better than other values.

Additionally, to quantify and evaluate the effectiveness of the proposed noise elimination strategy, we have performed further experiments and investigated into four possible scenarios/strategies for noise elimination: i) Automatic Noise Elimination (AUNE) (as described in Section 3.2) ii) No Noise Elimination (NONE) in which the first phase of our algorithm is entirely ignored iii) RR-LS in which ruin and recreate and Local Search heuristics only are participated in tensor construction and iv) MU-LS where mutation and local search heuristics are considered in tensor construction. Each scenario is tested on all CHeSC 2011 instances and during those experiments t_p is fixed as 30 seconds. After performing the factorisation, the ϕ value (Equation 2.13) is calculated for each instance at each run. Figure 3.6 provides the performance comparison of different noise elimination strategies based on the ϕ values averaged over 31 runs for each instance. It is desirable that the ϕ value, which expresses the model fitness, to be maximized in these experiments. Apart from the PS and FS domains, our automatic noise elimination scheme (AUNE) delivers the best ϕ for all other instances from the rest of the four domains. In three out of five FS instances, AUNE performs the best with respect to the model fitness. However, AUNE seems to be under-performing in the PS domain. The reason for this specific case is that the heuristics in this domain are extremely slow and the designated value(s) for t_p does not give the algorithm sufficient time to identify the source of noise properly and consistently. This is also the reason for the almost random partitioning of the heuristic space (figures 3.4(g), 3.4(h) and 3.4(i)). The low running speed of low level heuristics leads to a low number of collected tensor frames at the end of the second phase (tensor construction). Without enough information the factorisation method is unable to deliver useful and consistent partitioning of the heuristic space (like in other domains). That is why, the histograms belonging to the PS domain in Figure 3.4 demonstrate a half-half distribution of heuristics to $\boldsymbol{h}_{NA}$ and $\boldsymbol{h}_{IE}$ heuristic sets. Nevertheless, the overall results presented in this section supports the necessity of the noise elimination step and illustrates the success of the proposed simple strategy.

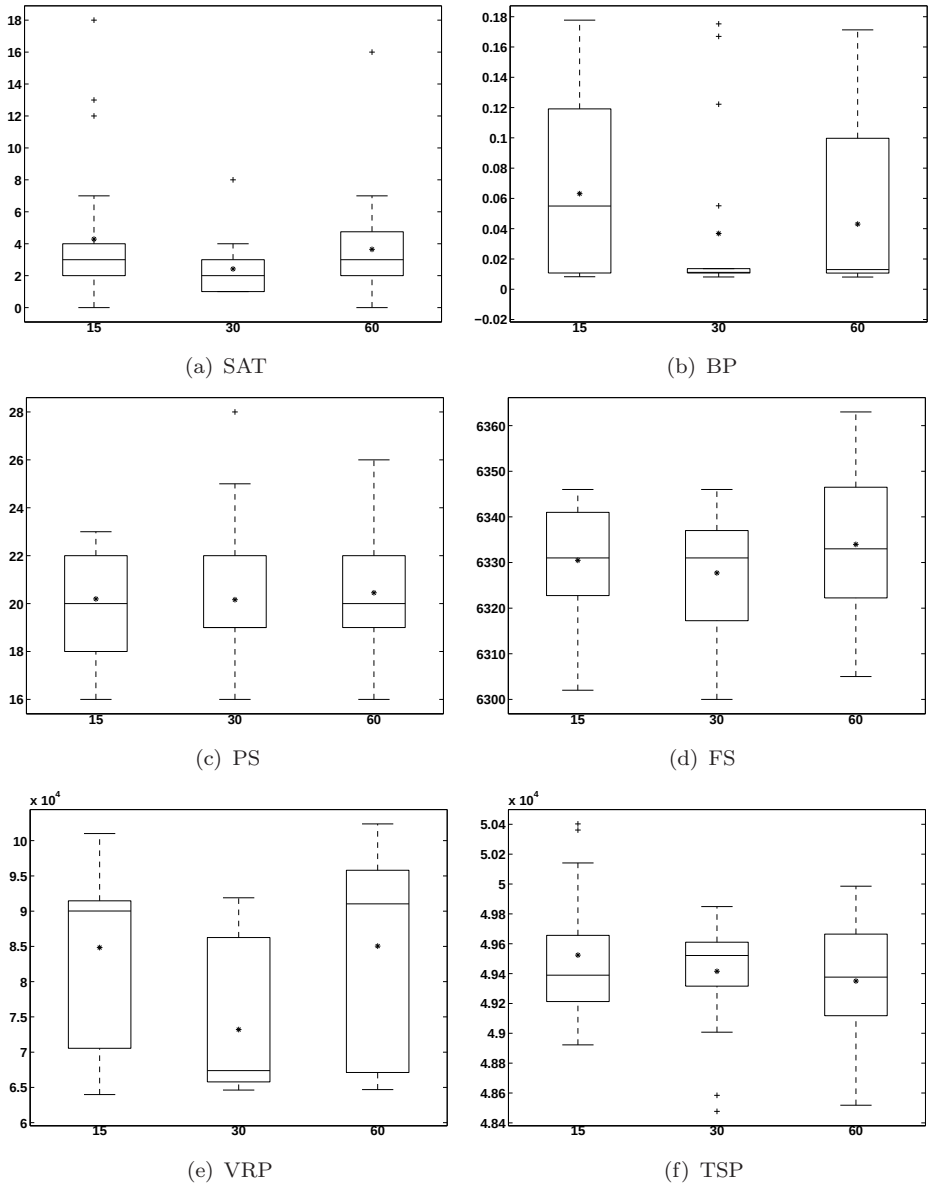
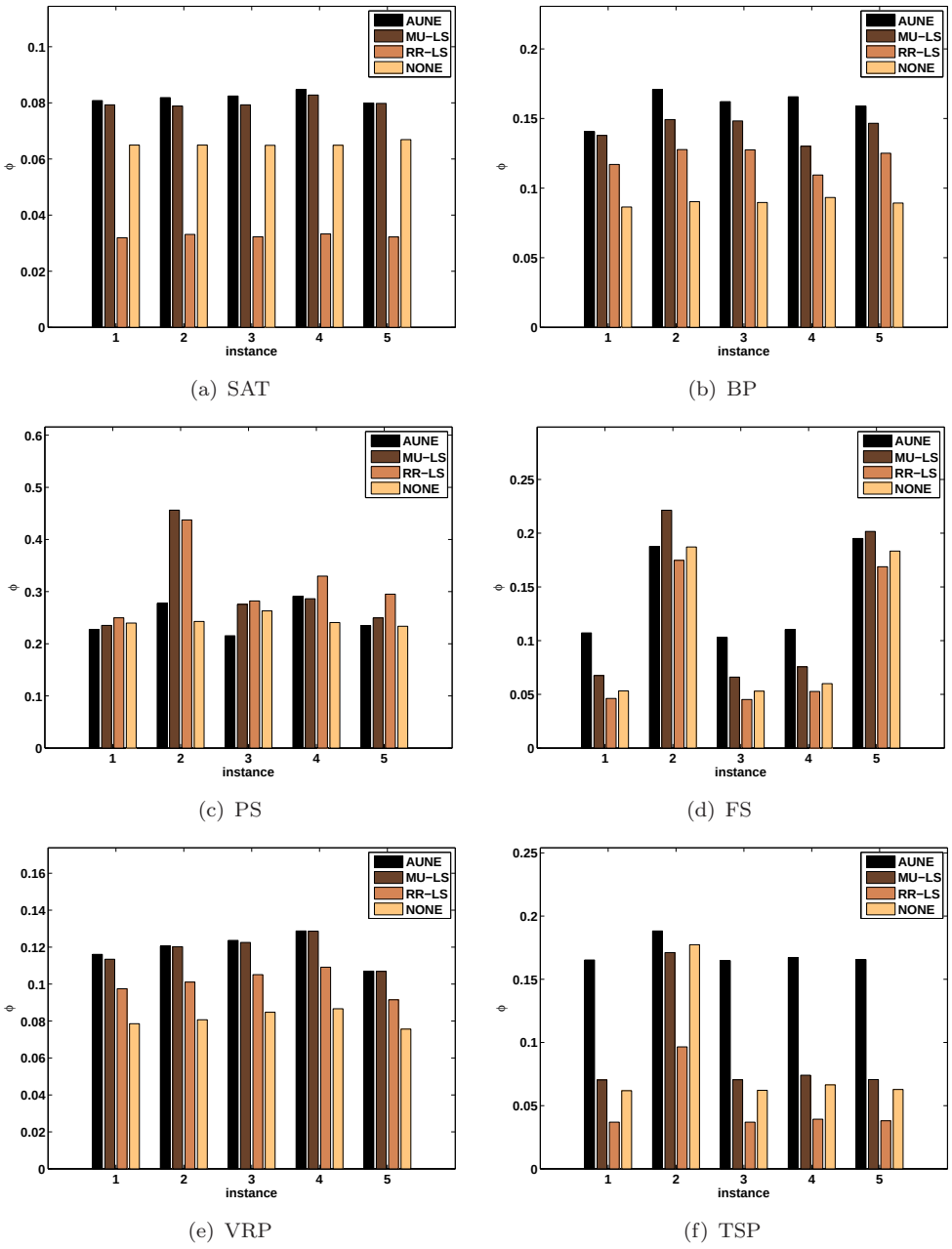


FIGURE 3.5: Comparing the performance of TeBHA-HH on the first instance of various domains for different values of t_p. The asterisk sign on each box plot is the mean of 31 runs.

3.3.3 Switch Time

The value assigned to t_s determines the frequency based on which the framework switches from one acceptance mechanism to another during the search process in the final phase. Four values: $nil, 500, 1000$ and 1500 milliseconds have been considered in our experiments. For $t_s = nil$, randomly chosen low level heuristic determines the move acceptance method to be employed at each step. If the selected heuristic is a member of $\boldsymbol{h}_{NA}$ or $\boldsymbol{h}_{IE}$, NA or IE is used for move acceptance, respectively. The value for t_p is fixed at 30 seconds and AUNE is used for noise elimination during all switch time experiments.



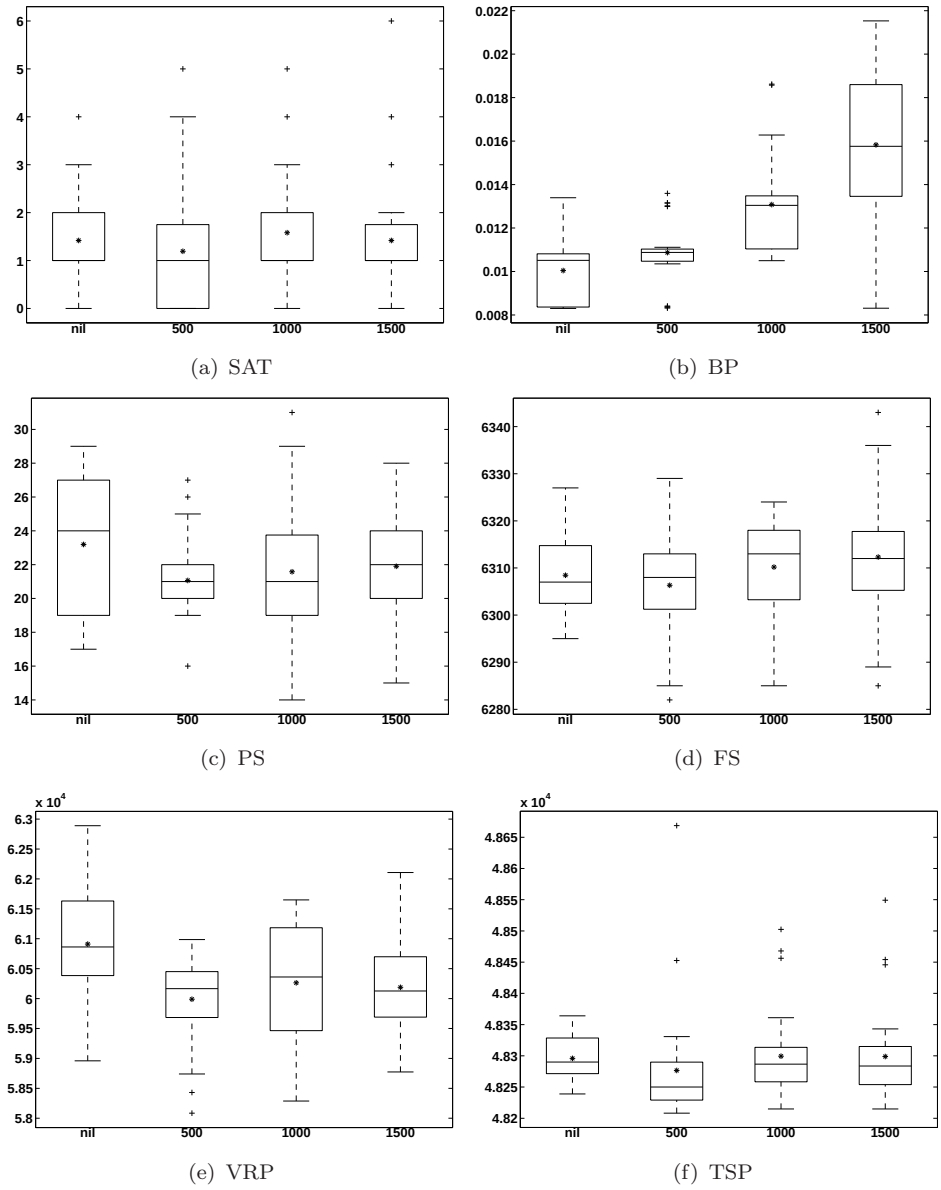
(a) SAT (b) BP

(c) PS (d) FS

(e) VRP (f) TSP

FIGURE 3.6: Comparing the model fitness in factorisation, ϕ (y axis of each plot), for various noise elimination strategies. Higher ϕ values are desirable. The x-axis is the ID of each instance from the given CHeSC 2011 domain.

A comparison between various values considered for t_s is given in Figure 3.7. Judging by the average performance (shown by an asterisk on each box), $t_s = 500$ msec performs slightly better than other values. Figure 3.8 shows the impact of the time allocated for the final phase on two sample problem domains and the efficiency that early decision making brings. $t_s = nil$ is also under performing. Similar phenomena are observed in the other problem domains.



(a) SAT (b) BP

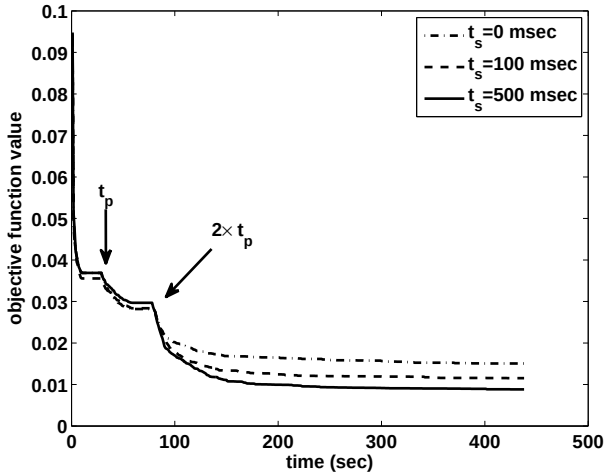
(c) PS (d) FS

(e) VRP (f) TSP

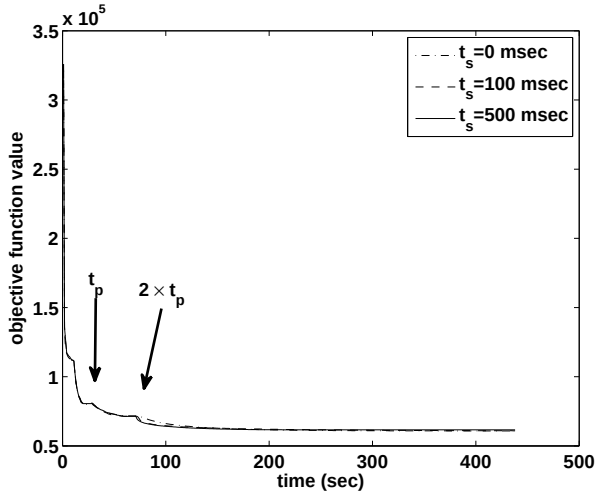
FIGURE 3.7: Comparing the performance (y axis) of TeBHA-HH on the first instance of various domains for different values of t_s (x axis). The asterisk sign on each box plot is the mean of 31 runs.

3.3.4 Experiments on the CHeSC 2011 Domains

After fixing the values for parameters t_p and t_s to best achieved values (30 seconds and 500 milliseconds, respectively) and using AUNE for noise elimination, we run another round of experiments testing the algorithm on all CHeSC 2011 instances. Table 3.1 summarises the results obtained using TeBHA-HH. The performance of the proposed hyper-heuristic is then compared to that of the two building block algorithms, namely SR-NA and SR-IE. Also, the current state-of-the-art algorithm, AdapHH [121] is included in the comparisons. Table 3.2 provides the details of the average performance



(a) BP



(b) VRP

FIGURE 3.8: Average objective function value progress plots on the (a) BP and (b) VRP instances for three different values of t_s where $t_p = 30$ sec.

comparison of TeBHA-HH to AdapHH. Clearly, TeBHA-HH outperforms AdapHH on PS and Max-SAT domains. A certain balance between the performance of the two algorithm is observable in VRP domain. In case of other problem domains, AdapHH manages to outperform our algorithm. The major drawback that TeBHA-HH suffers from is its poor performance on the FS domain. We suspect that ignoring heuristic parameter values such as depth of search or the intensity of mutation is one of the reasons.

The interesting aspect of TeBHA-HH is that, generally speaking, it uses a hyper-heuristic based on random heuristic selection, decomposes the low level heuristics into two subsets and again applies the same hyper-heuristic using two simple move acceptance methods.

TABLE 3.1: The performance of the TeBHA-HH framework on each CHeSC 2011 instance over 31 runs, where μ and σ are the mean and standard deviation of objective values. The bold entries show the best produced results compared to those announced in the CHeSC 2011 competition.

		Instances				
Problem		1	2	3	4	5
SAT	μ :	1.9	4.0	1.3	3.8	7.4
	min :	0	1	0	1	7
	σ :	2.3	3.5	1.0	3.0	0.8
BP	μ :	0.0116	0.0086	0.0107	0.1087	0.0173
	min :	**0.0083**	0.0035	0.0058	0.1083	0.0114
	σ :	0.0013	0.0032	0.0027	0.0007	0.0069
PS	μ :	20.8	9618.1	3241.8	1611.2	349.1
	min :	13	9355	3141	1458	310
	σ :	4.3	129.5	59.7	101.2	23.9
FS	μ :	6310.9	26922.2	6372.5	11491.3	26716.2
	min :	6289	26875	6364	11468	26606
	σ :	11.3	27.1	5.8	16.1	37.5
VRP	μ :	59960.6	13367.9	148318.6	20862.3	147540.2
	min :	**57715.2**	13297.9	143904.5	20656.2	145672.5
	σ :	30.7	5.3	44.8	20.0	31.1
TSP	μ :	48274.2	20799913.7	6874.7	66812.4	53392.0
	min :	48194.9	**20657952.5**	6851.1	66074.7	52661.2
	σ :	6.9	442.1	3.2	17.5	25.7

Despite this, the TeBHA-HH manages to perform significantly better than its building blocks of SR-IE and SR-NA as illustrated in Figure 3.9 on all almost all domains. The same behaviour is observed across the rest of the CHeSC 2011 instances.

3.3.4.1 Performance comparison to the competing algorithms of CHeSC 2011

The results obtained from the experiments as described in the previous section, are then compared to the results achieved by all CHeSC 2011 contestants. We used the Formula 1 scoring system provided by the organizers to determine the rank of our hyper-heuristic among all other competitors. Table 3.3 provides the ranking of all CHeSC 2011 hyper-heuristics including ours. Since Ant-Q received a score of 0, that hyper-heuristic is ignored. The details of ranking per domain, and the succeeding/preceding algorithms are shown in Figure 3.10. The TeBHA-HH ranks first in Max-SAT and VRP domains. It ranks 2nd in BP and 3rd in PS domains while it's ranking on the TSP domain is 4th. Our algorithm gained no score on the FS domain (a score of 0 equal to 10 other algorithms). Overall, TeBHA-HH ranks the second with a total score of 148 after AdapHH. As it is

TABLE 3.2: Average performance comparison of TeBHA-HH to AdapHH, the winning hyper-heuristic of CHeSC 2011 for each instance. Wilcoxon signed rank test is performed as a statistical test on the objective values obtained over 31 runs from TeBHA-HH and AdapHH. $\leq$ ($<$) denotes that TeBHA-HH performs slightly (significantly) better than AdapHH (within a confidence interval of 95%), while $\geq$ ($>$) indicates vice versa. The last column shows the number of instances for which the algorithm on each side of "/" has performed better.

	Instances					
Problem	1	2	3	4	5	TeBHA-HH/AdapHH
Max-SAT	$<$	$<$	$<$	$\leq$	$\leq$	5/0
BP	$<$	$>$	$>$	$>$	$>$	1/4
PS	$<$	$\leq$	$<$	$\leq$	$>$	4/1
FS	$>$	$>$	$>$	$>$	$>$	0/5
VRP	$<$	$\leq$	$>$	$>$	$\geq$	2/3
TSP	$>$	$\geq$	$>$	$>$	$\geq$	0/5

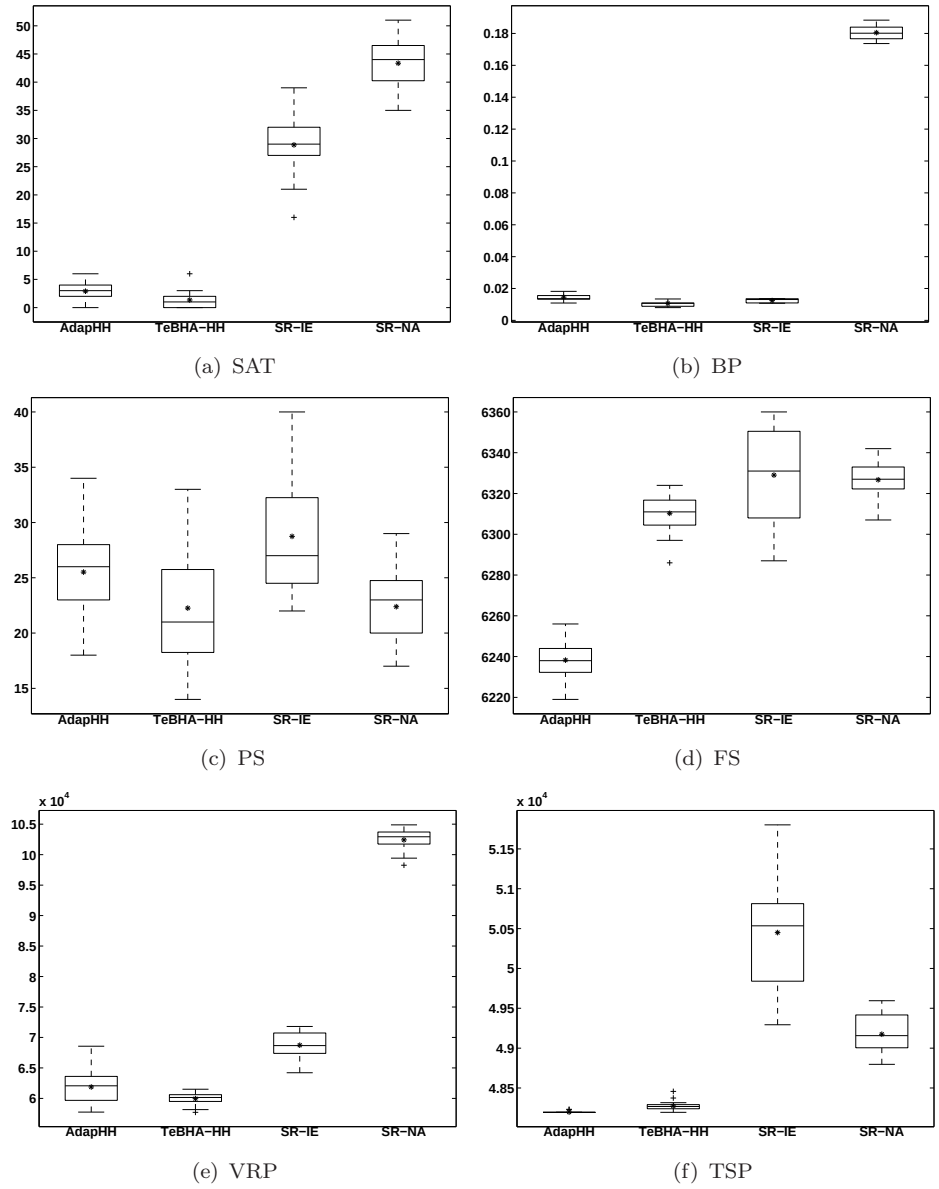
evident in Table 3.1, we produce the best results (compared to the results announced after the CHeSC 2011 competition) for the first instances of the BP and VRP domains as well as the best result for the second instances of the TSP domain (the bold entries in Table 3.1).

TABLE 3.3: Ranking of the TeBHA-HH among the selection hyper-heuristics that were competed in CHeSC 2011 with respect to their Formula 1 scores.

Rank	Name	Score	Rank	Name	Score
1	AdaptHH	162.83	11	HAEA	45
2	**TeBHA-HH**	**148**	12	ACO-HH	37
3	VNS-TW	118.83	13	Gen-Hive	32.5
4	ML	117.5	14	DynILS	22
5	P-Hunter	84.75	15	SA-ILS	18.75
6	EPH	83.25	16	AVEG-Nep	18.5
7	NAHH	68.5	17	XCJ	17.5
8	HAHA	65.58	18	GISS	13.25
9	ISEA	62.5	19	MCHH-S	3.25
10	KSATS-HH	52	20	Self Search	3

3.3.4.2 An Analysis of TeBHA-HH

In this section, an analysis of the TeBHA-HH algorithm is performed to gain some insight into its behaviour during the search process. The objective value progress plots in almost all cases look the same, hence we have chosen an instance of the BP problem from CHeSC 2011 for which the TeBHA-HH demonstrates a good performance, while it consistently produces the same heuristic space partitioning in the tensor analysis stage. It is clear



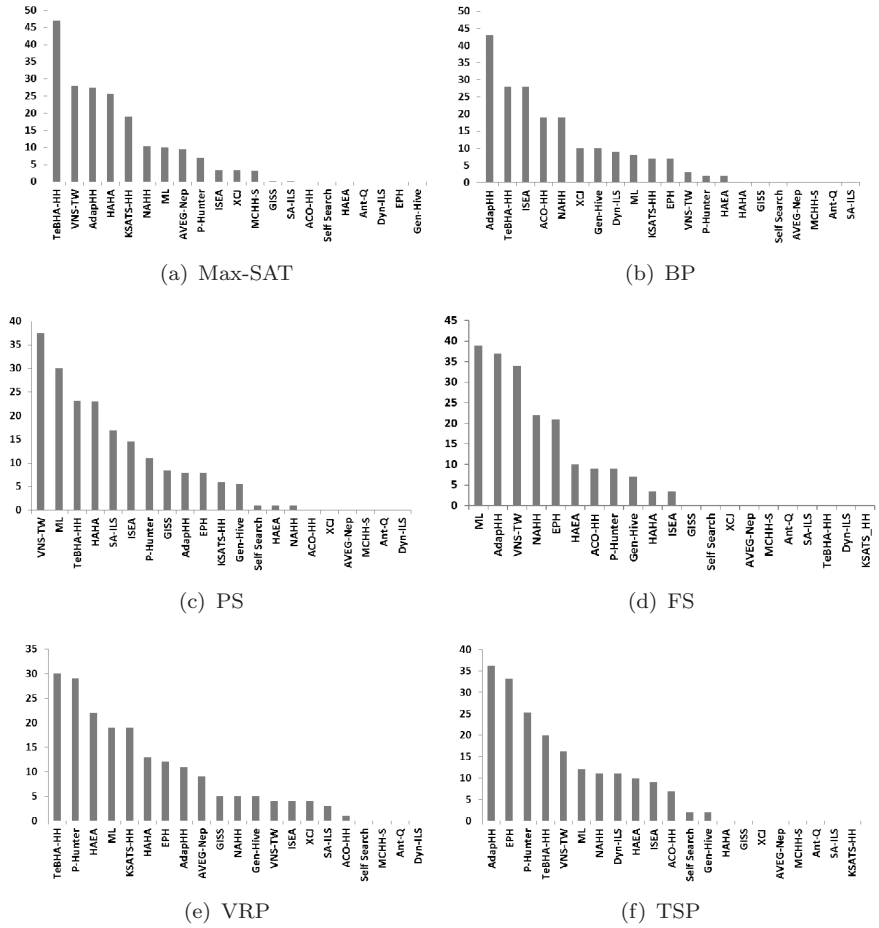
(a) SAT (b) BP

(c) PS (d) FS

(e) VRP (f) TSP

FIGURE 3.9: Box plots of objective values (y axis) over 31 runs for the TeBHA-HH with AdapHH, SR-NA and SR-IE hyper-heuristics on a sample instance from each CHeSC 2011 problem domain.

from Figure 3.4(e) that heuristics $MU1$ and $LS1$ are always assigned to the set $\boldsymbol{h}_{NA}$ while the rest of the heuristics (excluding the crossover heuristics) are assigned to the set $\boldsymbol{h}_{IE}$. In each previous experiment, we run our algorithm on the BP instance for 31 runs. The plot in Figure 3.11(a) shows how the average objective value changes (decreases) in time during the search process. We have divided the progress of the algorithm into 3 distinct stages which represent the early, medium and late stages of the search, respectively (not to be confused with algorithm phases/stages, this is simply dividing the run-time into 3 periods). Figure 3.11(b) shows a close-up of the same plot achieved for a single run within the early stage. The dark rectangle shapes correspond to the times when naive



(a) Max-SAT (b) BP

(c) PS (d) FS

(e) VRP (f) TSP

FIGURE 3.10: Ranking of the TeBHA-HH and hyper-heuristics which competed at CHeSC 2011 for each domain.

acceptance is in charge. It is obvious that an extensive diversification process takes place in the vicinity of the current objective value when NA is at work. It also seems that during the time when the IE acceptance mechanism is in charge, intensification is in order. This leads to the conclusion that the hybrid acceptance scheme approximates the actions of a higher level iterated local search while maintaining a balance between intensification and diversification. This is in-line with extracting *domain independent domain knowledge* as discussed in [207] where the knowledge encapsulates the heuristic indexes assigned to each acceptance mechanism. We have seen in Section 3.3.3 that the value $t_s = 500$ msec, results in slightly better objective function values, especially when compared to $t_s = nil$. The analysis given here clarifies this issue. When $t_s = nil$, less time remains for both diversification and intensification processes, hence leading to a poor interaction/balance between the two acceptance mechanisms.

Furthermore, regardless of the type of acceptance mechanism, the algorithm manages to update the best values as is shown in Figure 3.11(c). The share of each heuristic in updating the best-so-far solution is also demonstrated in Figure 3.11(c). Interestingly,

while the local search heuristic $LS1$ is responsible for most of improvements when NA is at work, a mutation operator ($MU1$) increasingly produces most of the improvements when IE is operational. In a way, hyper-heuristic using IE operates like a random mutation local search.

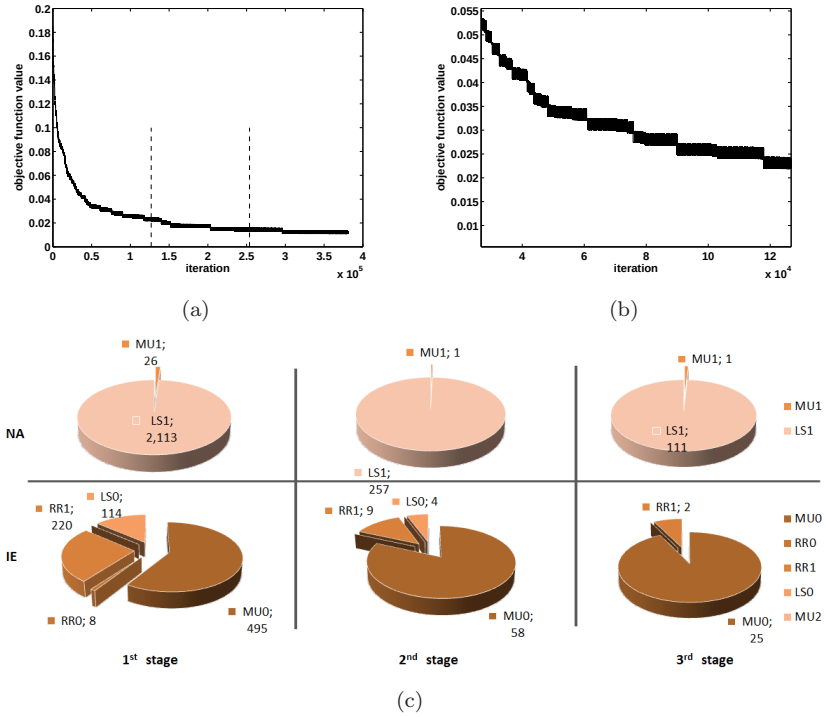


FIGURE 3.11: The interaction between NA and IE acceptance mechanisms: (a) The search process is divided into three sections, (b) a close-up look on the behaviour of the hybrid acceptance mechanism within the first section in (a), (c) the share of each acceptance mechanism in the overall performance stage-by-stage.

3.4 Summary

Machine learning is an extremely important component of the adaptive search methodologies, such as hyper-heuristics. The use of a learning mechanism becomes even more crucial considering that hyper-heuristics aim to "raise the level of generality" by their design and implementation which is expected be applicable to different problem domains, rather than a single domain. Reinforcement learning, learning automata, genetic programming and classifier systems are some of the online and offline learning techniques that have been used within or as hyper-heuristics [11]. In this chapter, we have introduced a novel selection hyper-heuristic embedding a tensor-based approach as a machine learning technique and combining random heuristic selection with the naive (NA), and improving and equal (IE) move acceptance methods. The tensor-based approaches have

been successfully applied in other areas of research, such as computer vision [149], but they have never been used in heuristic search, previously.

In the proposed approach, tensors represent the interaction between low level heuristics in time under a certain selection hyper-heuristic framework. Then the gained knowledge is used to improve the overall search process later by hybridising move acceptance methods in relation to the low level heuristics. In order to be able to evaluate the behaviour of the proposed approach, we have ensured that the building blocks of the framework are in their simplest forms. For example, the default settings for the HyFlex low level heuristics are used and NA and IE are employed as move acceptance components. Nevertheless, the proposed tensor-based framework proved to be very powerful demonstrating an outstanding performance. Using NA and IE move acceptance in a multi-stage manner switching between them enforces diversification and intensification balance. Despite all the simplicity of its components, our hyper-heuristic ranked the second among the contestants of the CHeSC 2011 across six problem domains, even beating the average and best performance of the winning approach in some problem instances, particularly from bin packing, maximum satisfiability and the vehicle routing problem.

The approach which was presented in this chapter consists of a single episode of learning. A learning episode is the process of collecting the data, analysing it and applying the results of the analysis step to the search. Extending the proposed approach into a multi-episode learning system will further clarify whether or not the learning mechanism is capable of detecting useful patterns in the long term. Furthermore, there are several acceptance criteria available in the literature (as discussed in Section 2.2.1.1). However, the approach proposed here can only consider one acceptance criteria to construct a tensor. Given the number of different acceptance methods available from the literature, it seems necessary to extend the approach and enable it to consider more than one method. Based on these reasons, in the next chapter, we present a modified variant of the tensor-based approach which caters for multiple acceptance criteria and apply it to instances of the Nurse Rostering problem instances in a multiple episode fashion.

Chapter 4

A Tensor-based Selection Hyper-heuristic for Nurse Rostering

The approach proposed in Chapter 3 used a single episode of learning to extract patterns and achieved very good results. In this chapter we investigate whether the proposed approach is capable of extracting good results continuously. That is, we conduct multiple episode experiments during long runs to see if the proposed approach extracts useful patterns throughout the time. The benefits of this property is discussed further in this chapter. Apart from investigating the multi episode behaviour of the proposed approach, extensions to the original approach have been proposed here which enable the algorithm to embrace a virtually infinite number of acceptance criteria. Also, it has been shown here that the tensor-based approach can also be used to tune the parameters of heuristics. The abstraction level in data is precisely as it was in Chapter 3 and is considered to be high.

4.1 Introduction

Nurse rostering is a highly constrained scheduling problem which was proven to be NP-hard [208] in its simplified form. Solving a nurse rostering problem requires assignment of shifts to a set of nurses so that 1) the minimum staff requirements are fulfilled and 2) the nurses' contracts are respected [209]. The problem can be represented as a constraint optimisation problem using 5-tuples consisting of set of nurses, days (periods) including the relevant information from the previous and upcoming schedule, shift types, skill types and constraints.

In this chapter, a tensor-based selection hyper-heuristic approach is employed to tackle the nurse rostering problem. The proposed framework (which is an extension to the framework described in Chapter 3) is a single point-based search algorithm which fits best in the online learning selection hyper-heuristic category, even if it is slightly different to the other online learning selection hyper-heuristics.

Our proposed approach consists of running the simple random heuristic selection strategy in four stages. In the first stage the acceptance mechanism is NA while in the second stage we use IE as acceptance mechanism. The trace of the hyper-heuristic in each stage is represented as a $3^{rd}-$order tensor. After each stage commences, the respective tensor is factorized which results in a score value associated to each heuristic. The space of heuristics is partitioned into two distinct sets, each representing a different acceptance mechanism (NA and IE respectively) and lower level heuristics associated to it. Subsequently, a hyper-heuristic is created which uses different acceptance methods in an interleaving manner, switching between acceptance methods periodically. In the third stage, the parameter values for heuristics is extracted by running the hybrid hyper-heuristic and collecting tensorial data similar to the first two stages. Subsequently, the hybrid hyper-heuristic equipped with heuristic parameter values is run for a specific time. The above mentioned procedure continues until the maximum allowed time is reached.

Compared to the method proposed in Chapter 3, the framework here has few modifications. First, the previous framework has been extended to accommodate for an arbitrary number of acceptance criteria to be involved in the framework. That is, in contrast to the work in Chapter 3 where tensor data was collected for one acceptance criteria and the space of heuristics was partitioned into two disjoint sets, in this chapter, data collection and tensor analysis is performed for each hyper-heuristic separately. Moreover, low level heuristics are partitioned dynamically, rather than only once which was the case in Chapter 3 where ten (nominal) minute runs were considered. Mining search data periodically allows us to investigate whether the framework is capable of extracting new knowledge as the search makes progress. This could be useful in a variety of applications (i.e. life-long learning as in [210], [211] and [212] or apprenticeship learning as in [45] and [42]). Finally, the framework here is different than the one proposed in Chapter 3 when parameter control for each low level heuristic is considered. While no parameter control was done in Chapter 3, in this chapter, parameters of each heuristic is tuned using tensor analysis. The good results achieved in this chapter shows that tensor analysis can also play a parameter control role.

4.2 Nurse Rostering

In this section, we define the nurse rostering problem dealt with. Additionally, an overview of related work is provided.

4.2.1 Problem Definition

The constraints in the nurse rostering problem can be grouped into two categories: (i) those that link two or more nurses and (ii) those that only apply to a single nurse. Constraints that fall into the first category include the cover (sometimes called demand) constraints. These are the constraints that ensure a minimum or maximum number of nurses are assigned to each shift on each day. They are also specified per skill/qualification levels in some instances. Another example of a constraint that would fall into this category would be constraints that ensure certain employees do or do not work together. Although these constraints do not appear in most benchmark instances (including those used here), they do occasionally appear in practise to model requirements such as training/supervision, carpooling, spreading expertise etc. The second group of constraints model the requirements on each nurse's individual schedule. For example, the minimum and maximum number of hours worked, permissible shifts, shift rotation, vacation requests, permissible sequences of shifts, minimum rest time and so on.

In this chapter, our aim is to see whether any improvement is possible via the use of machine learning, particularly tensor analysis. We preferred using the benchmark provided at [213] as discussed in the next section. These benchmark instances are collected from a variety of workplaces across the world and as such have different requirements and constraints, particularly the constraints on each nurse's individual schedule. This is because different organisations have different working regulations which have usually been defined by a combination of national laws, organisational and union requirements and worker preferences. To be able to model this variety, in [214] a regular expression constraint was used. Using this domain specific regular expression constraint allowed all the nurse specific constraints found in these benchmarks instances to be modelled. The model is given below.

Sets

E = Employees to be scheduled, $e \in E$

T = Shift types to be assigned, $t \in T$

D = Days in the planning horizon, $d \in \{1, \cdots |D|\}$

R_e = Regular expressions for employee e, $r \in R_e$

W_e = Workload limits for employee e, $w \in W_e$

Parameters

r_{er}^{max} = Maximum number of matches of regular expression r in the work schedule of employee e.
r_{er}^{min} = Minimum number of matches of regular expression r in the work schedule of employee e.
a_{er} = Weight associated with regular expression r for employee e.
v_{ew}^{max} = Maximum number of hours to be assigned to employee e within the time period defined by workload limit w.
v_{ew}^{min} = Minimum number of hours to be assigned to employee e within the time period defined by workload limit w.
b_{ew} = Weight associated with workload limit w for employee e.
s_{td}^{max} = Maximum number of shifts of type t required on day d.
s_{td}^{min} = Minimum number of shifts of type t required on day d.
c_{td} = Weight associated with the cover requirements of shift type t on day d.

Variables
$x_{etd} = 1$ if employee e is assigned shift type t on day d, 0 otherwise.
n_{er} = The number of matches of regular expression r in the work schedule of employee e.
p_{ew} = The number of hours assigned to employee e within the time period defined by workload limit w.
q_{td} = The number of shifts of type t assigned on day d.

Constraints
Employees can be assigned only one shift per day.

$$\sum_{t \in T} x_{etd} \leq 1, \quad \forall e \in E,\ d \in D \tag{4.1}$$

Objective Function

$$Minf(s) = \sum_{e \in E} \sum_{i=1}^{4} f_{e,i}(x) + \sum_{t \in T} \sum_{d \in D} \sum_{i=5}^{6} f_{t,d,i}(x) \tag{4.2a}$$

$where$

$$f_{e,1}(x) = \sum_{e \in R_e} max\{0, (n_{er} - r_{er}^{max})a_{er}\} \tag{4.2b}$$

$$f_{e,2}(x) = \sum_{e \in R_e} max\{0, (r_{er}^{min} - n_{er})a_{er}\} \tag{4.2c}$$

$$f_{e,3}(x) = \sum_{w \in W_e} max\{0, (p_{ew} - v_{ew}^{max})b_{ew}\} \tag{4.2d}$$

$$f_{e,4}(x) = \sum_{w \in W_e} max\{0, (v_{ew}^{min} - p_{ew})b_{ew}\} \tag{4.2e}$$

$$f_{e,5}(x) = max\{0, (s_{td}^{min} - q_{td})c_{td}\} \tag{4.2f}$$

$$f_{e,6}(x) = max\{0, (q_{td} - s_{td}^{max})c_{td}\} \tag{4.2g}$$

To facilitate comparing results and to remove the difficulties in comparing infeasible solutions, the benchmark instances were designed with only one hard constraint 4.1 which is always possible to satisfy. Every other constraint is modelled as a soft constraint, meaning that it becomes part of the objective function. If in practice, in one of the instances, a soft constraint should really be regarded as a hard constraint then it was given a very high weight (the Big M method). The objective function is thus given in equation 4.2a. It consists of minimising the sum of equations 4.2b to 4.2g. Equations 4.2b and 4.2c ensure that as many of the regular expression constraints are satisfied as possible. These constraints model requirements on an individual nurse's shift pattern. For example, constraints on the length of a sequence of consecutive working days, or constraints on the number of weekends worked, or the number of night shifts and so on. Equations 4.2d and 4.2e ensure that each nurse's workload constraints are satisfied. For example, depending on the instance, there may be a minimum and maximum number of hours worked per week, or per four weeks, or per month or however the staff for that organisation are contracted. Finally, equations 4.2f and 4.2g represent the demand (sometime called cover) constraints to ensure there are the required number of staff present during each shift. Again, depending upon the instance, there may be multiple demand curves for each shift to represent, for example, the minimum and maximum requirements as well as a preferred staffing level. The weights for the constraints are all instance specific because they represent the scheduling goals for different institutions.

Instance	Best Known	No. of Staff	Shift Types	No. of Shifts	Ref.
BCV-A.12.2	1875	12	5	31	[127]
BCV-A.12.1	1294	12	5	31	[215]
CHILD-A2	1095	41	5	42	[127]
ERRVH-A	2135	51	8	48	[127]
ERRVH-B	3105	51	8	48	[127]
ERMGH-B	1355	41	4	48	[127]
MER-A	8814	54	12	48	[127]
Valouxis-1	20	16	3	28	[216]
Ikegami-3Shift-DATA1.2	3	25	3	30	*
Ikegami-3Shift-DATA1.1	3	25	3	30	*
Ikegami-3Shift-DATA1	2	2	3	30	*
ORTEC01	270	16	4	31	[214]
ORTEC02	270	16	4	31	[214]
BCV-3.46.1	3280	46	3	26	[215]

TABLE 4.1: Instances of nurse rostering problem and their specifications (best known objective values corresponding to entries indicated by * are taken from private communication from Nobuo Inui, Kenta Maeda and Atsuko Ikegami).

4.2.2 Related Work

There are various benchmarks for nurse rostering problems. In [213], a comprehensive benchmark is available where the latest best known results together with approaches yielding these results are available. The characteristics of the benchmark instances from [213] used in the experiments are summarized in Table 4.1.

There is a growing interest in challenges and the instances used during those challenges and resultant algorithms serve as a benchmark afterwards. The last nurse rostering competition was organised in 2010 [217] which consisted of three tracks where each track differed from others in maximum running time and size of instances. Many different algorithms have been proposed since then ([218], [219], [220] and etc). Since it has been observed that the previous challenge did not impose much difficulty for the competitors [214], other than developing a solution method in limited amount of time, a second challenge has been organised which is ongoing [221]. In the second nurse rostering competition, the nurse rostering problem is reformulated as a multi-stage problem with fewer constraints where a solver is expected to deal with consecutive series of time periods (weeks) and consider longer planning horizon.

In [214] a branch and price algorithm and an ejection chain method has been used for nurse rostering problem instances collected (from thirteen different countries) by the authors [1]. Branch and price method is based on the branch and bound technique with the difference that each node of the tree is a linear programming relaxation and is solved through column generation. The column generation method consists of two parts: the restricted master problem and the pricing problem. The former is solved using a linear programming method while the latter is using a dynamic programming approach. Some of the latest results and best-known solutions regarding the instances is provided by this chapter. Also, a general problem modelling scheme has been proposed in [214] which is also adopted here due to its generality over many problem instances.

In [216] a generic two-phase variable neighbourhood approach has been proposed for nurse rostering problems. After determining the value of the parameters which govern the performance of the algorithm, a random population of candidate solutions is generated. In the first phase of the algorithm, assigning nurses to working days is handled. Subsequent to this phase, in the second phase, assigning nurses to shift types is dealt with. Though, the proposed approach has been applied to few publicly available instances, the chosen instances are significantly different from one another.

In [222] a method based on mixed integer linear programming is proposed to solve four of the instances also in [214] and [213], namely, `ORTEC01`, `ORTEC02`, `GPost` and `GPost-B`. The method is able to solve these instances to optimality very quickly. The idea of implied penalties has been introduced in this study. Employing implied penalties avoids accepting small improvements in the current rostering period at the expense of penalising larger penalties on the next rostering period.

In [223] the nurse rostering problem has been identified as an over-constrained one and it is modelled using soft global constraints. A variant of Variable Neighbourhood Search (VNS), namely VNS/LDS+CP [224], is used as a metaheuristic to solve the problem instances. The proposed approach has been tested on nine different instances (available in [213]). The experimental results show that the method is relatively successful, though the authors have suggested to use specific new heuristic for instances such as `Ikegami` to improve the performance of the algorithm.

In [225] a hybrid multi-objective model has been proposed to solve nurse rostering problems. The method is based on Integer Programming (IP) and Variable Neighbourhood Search (VNS). The IP method is used in the first phase of the algorithm to produce intermediary solutions considering all the hard constraints and a subset of soft constraints. The solution is further polished using the VNS method. The proposed approach is then applied to the `ORTEC` problem instances and compared to a commercial hybrid Genetic

[1] These instances as well as other nurse rostering instances can be found at [213]

Algorithm (GA) and a hybrid VNS [226]. The computational results show that the proposed approach outperforms both methods in terms of solution quality.

In [127], a hyper-heuristic method inspired by pearl hunting is proposed and applied to various nurse rostering instances. The proposed method is based on repeated intensification and diversification and can generally be described as a type of Iterated Local Search (ILS). Their experiment consists of running the algorithm on various instances for several times, where each run is 24 CPU hours long. The algorithm discovered 6 new best-known results.

Numerous other approaches have been proposed to solve the nurse rostering problem. In [227] a Scatter Search (SS) is proposed to tackle the nurse rostering problem. A shift sequence-based approach was proposed in [228]. In [229] the nurse rostering problem is modelled using 0-1 Goal Programming Model.

4.3 Proposed Approach

The proposed approach consists of the consecutive iteration of four stages as depicted in Algorithm 8 and Figure 4.1. In all stages, *simple* hyper-heuristic algorithms operating on top of a fixed set of low level heuristics (move operators) are used. Those low level heuristics are exactly the same low level heuristics implemented for the personnel scheduling problem domain [230] under the Hyper-heuristic Flexible Framework (HyFlex) v1.0 [36]. The low level heuristics in HyFlex are categorized into four groups: mutation (MU), ruin and re-create (RR), crossover (XO) and local search (LS). Consequently, one mutation operator is available for the nurse rostering problem domain which is denoted here by $MU0$. This operator randomly un-assigns a number of shifts while keeping the resulting solution feasible. Three ruin and re-create heuristics are available which are denoted by $RR0$, $RR1$ and $RR2$. These operators are based on the heuristics proposed in [226] and they operate by un-assigning all the shifts in one or more randomly chosen employees' schedule followed by a rebuilding procedure. These operators differ in the size of the perturbation they cause in the solution. Five local search heuristics, denoted by $LS0$, $LS1$, $LS2$, $LS3$ and $LS4$ are also used where the first three heuristics are hill climbers and the remaining two are based on variable depth search. Also, three different crossover heuristics are used which are denoted by $XO0$, $XO1$ and $XO2$. The crossover operators are binary operators and applied to the current solution in hand and the best solution found so far (which is initially the first generated solution). More information on these heuristics can be found in [230].

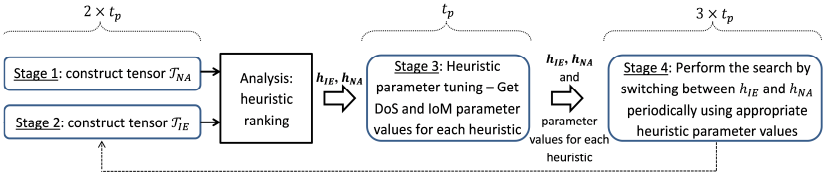


FIGURE 4.1: Overall approach with various stages.

During the first two stages (line 2 and 3), two different tensors are constructed. The tensor $\mathcal{T}_{NA}$ is constructed by means of an SR-NA algorithm and tensor $\mathcal{T}_{IE}$ is constructed from the data collected from running an SR-IE algorithm. At the end of the second stage (line 4 and 5), each tensor is subjected to factorisation to obtain basic frames ($\mathbf{B}_{NA}$ and $\mathbf{B}_{IE}$) and score values ($\mathbf{S}_{NA}$ and $\mathbf{S}_{IE}$) corresponding to each tensor. Using all the information we have on both tensors, the heuristic space is partitioned (line 6) to two distinct sets: $\mathbf{h}_{NA}$ and $\mathbf{h}_{IE}$. Subsequently, in the third stage, a hybrid algorithm is executed for a limited time (t_p) with random heuristic parameter values (depth of search and intensity of mutation). The hybrid algorithm consists of periodically switching between the two acceptance mechanisms NA and IE. Depending on the chosen acceptance method, the heuristics are chosen either from $\mathbf{h}_{NA}$ or $\mathbf{h}_{IE}$. In fact the hybrid algorithm is very similar to the algorithm in the final stage except that during the search process in this stage, the heuristic parameters are chosen randomly and a tensor using the heuristic parameter settings is constructed. Factorising this tensor and obtaining the basic frame (similar to what is done in previous steps) results in good parameter value settings for heuristics. Hence this stage can be considered as a parameter tuning phase for the heuristics. The final (fourth) stage consists of running the previous stage for a longer time ($3 \times t_p$) and assigning values achieved in the previous stage to heuristic parameters. After the time specified for the fourth stage is consumed, the algorithm starts over from stage one. This whole process continues until the maximum time allowed (T_{max}) for a given instance is reached. Figure 4.1 illustrates this process.

Algorithm 8: Tensor-based hyper-heuristic

1 **while** $t < T_{max}$ **do**
2 $(\mathcal{T}_{NA})$ = ConstructTensor($\mathbf{h}$, NA, t_p);
3 $(\mathcal{T}_{IE})$ = ConstructTensor($\mathbf{h}$, IE, t_p);
4 $(\mathbf{B}_{NA}, \mathbf{S}_{NA})$ = Factorisation($\mathcal{T}_{NA}$,$\mathbf{h}$);
5 $(\mathbf{B}_{IE}, \mathbf{S}_{IE})$ = Factorisation($\mathcal{T}_{IE}$,$\mathbf{h}$);
6 $(\mathbf{h}_{NA}, \mathbf{h}_{IE})$ = Partitioning($\mathbf{B}_{NA}, \mathbf{B}_{IE}, \mathbf{S}_{NA}, \mathbf{S}_{IE}$);
7 $(\mathbf{P})$ = ParameterTuning();
8 Improvement($\mathbf{h}_{NA}, \mathbf{h}_{IE}, \mathbf{P}, \mathbf{S}_{NA}, \mathbf{S}_{IE}$);
9 **end**

4.3.1 Tensor Analysis for Dynamic Low Level Heuristic Partitioning

During first and second stages, given an acceptance criteria (such as NA or IE), a hyper-heuristic with a simple random heuristic selection and the given acceptance methodology is executed. During the run, data, in the form of a tensor is collected. Hence the collected tensor data ($\mathcal{T}_{NA}$ or $\mathcal{T}_{IE}$ depending on the acceptance criteria) is a 3^{rd}-order tensor of size $\mathbb{R}^{|\mathbf{h}|} \times \mathbb{R}^{|\mathbf{h}|} \times \mathbb{R}^{t}$, where $|\mathbf{h}|$ is the number of low level heuristics and t represents the number of tensor frames collected in a given amount of time. Each tensor is a collection of two dimensional matrices ($\mathbf{M}$) which are referred to as *tensor frames*. A tensor frame is a two dimensional matrix of heuristic indices. Column indices in a tensor frame represent the index of the current heuristic whereas row indices represent the index of the heuristic chosen and applied before the current heuristic. Algorithm 9 shows the tensor construction procedure.

The core of the algorithm is the SR selection hyper-heuristic (starting at the while loop in line 4) combined with the acceptance criteria which is given as input. Repeatedly, a new heuristic is selected at random (line 12) and is applied to the problem instance (line 14). The returned objective function value (f_{new}) is compared against the the old objective function value and the immediate change in objective function value is calculated as $\delta_f = f_{old} - f_{new}$. The method `Accept` (line 16) takes the δ_f value as input and returns a decision as to whether accept the new solution or reject it. In case the new solution is accepted , assuming that the indices of the current and previous heuristics are $h_{current}$ and $h_{previous}$ respectively, the tensor frame $\mathbf{M}$ is updated symmetrically: $m_{h_{previous},h_{current}} = 1$ and $m_{h_{current},h_{previous}} = 1$. The tensor frame $\mathbf{M}$ is only allowed to have $\lfloor |\mathbf{h}|/2 \rfloor$ elements with a value of 1 (line 5). Whenever this threshold is reached the tensor frame is added to the tensor and a new frame is initialized (lines 6 to 9). Assuming that the objective function value before updating the frame for the first time is f_{start} and assuming that the objective function value after the last update in the tensor frame is f_{end}, then the frame is labelled as $\Delta_f = f_{start} - f_{end}$ (line 7). In other word, the label of a frame (Δ_f) is the overall change in the objective function value caused during the construction of the frame. Δ_f (given in Eq. 3.1) is different from δ_f in the sense that the former measures the change in objective value inflicted by the collective application of active heuristic indexes inside a frame. The latter is the immediate change in objective value caused by applying a single heuristic. This whole process is repeated until a time limit (t_p) is reached (line 4). This procedure, creates a tensor of binary data for a given acceptance method. In order to prepare the tensor for factorisation and increase the chances of gaining good patterns from the data, the frames of the constructed tensor are scanned for consecutive frames of positive labels($\Delta_f > 0$). Only these frames are kept in the tensor and all other frames are removed. This adds an extra emphasis on

intra-frame correlations which is important in discovering good patterns. The resulting tensor for each acceptance criteria is then fed to the factorisation procedure.

Algorithm 9: The tensor construction phase

```
1  In: h, acceptance criteria, P, t_p;
2  Initialize tensor frame M to 0;
3  counter = 0;
4  while t < t_p do
5      if counter = ⌊|h|/2⌋ then
6          append M to T;
7          set frame label to Δ_f;
8          Initialize tensor frame M to 0;
9          counter = 0;
10     end
11     h_previous = h_current;
12     h_current = selectHeuristic(h);
13     f_current = f_new;
14     f_new =applyHeuristic(h_current);
15     δ_f = f_current − f_new;
16     if Accept(δ_f, acceptance criteria) then
17         m_{h_previous,h_current} = 1;
18         m_{h_current,h_previous} = 1;
19         counter + +;
20     end
21 end
22 Construct final tensor T from collected data;
```

In the factorisation stage, a tensor $\mathcal{T}$ is fed to the factorisation procedure (Algorithm 10). This could be $\mathcal{T}_{NA}$ or $\mathcal{T}_{IE}$ depending on who calls the factorisation procedure. Using the CP factorisation, basic factors of the input tensor $\mathcal{T}$ are obtained (line 2). Using these basic factors the basic frame $\mathbf{B}$ is computed (line 3). To obtain the basic frame, Equation 2.7 is used where $K = 1$ and basic factors $\mathbf{a}$ and $\mathbf{b}$ represent previous and current heuristic indexes respectively (Figure 4.2). The values in the basic frame quantify the relationship between the elements along each dimension (basic factor). To make use of the basic frame, the maximum entry is pinpointed and the column corresponding to this entry is sorted. This results in a vector $\mathbf{S}$ which contains the score values achieved for heuristics.

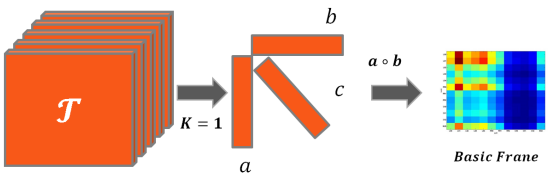


FIGURE 4.2: Extracting the basic frame for $K = 1$ in Eq.2.7.

The factorisation stage is applied to both $\mathcal{T}_{NA}$ and $\mathcal{T}_{IE}$ tensors. That is, the procedures tensor construction (Algorithm 9) and factorisation (Algorithm 10) are executed twice independently. Starting from an initial solution, first we execute the two procedure assuming NA as acceptance criteria. Consequently, we obtain a basic frame $\mathbf{B}_{NA}$ and a set of score values $\mathbf{S}_{NA}$. Following this, the two procedure (Algorithms 9 and 10) are executed assuming IE as acceptance criteria. This results in a basic frame $\mathbf{B}_{IE}$ and score values $\mathbf{S}_{IE}$. Consequently, for a given heuristic, there are two score values, one obtained from factorising $\mathcal{T}_{NA}$ and the other obtained from factorising $\mathcal{T}_{IE}$. The obtained score values are then fed into the partitioning procedure (Algorithm 11).

Algorithm 10: Factorisation

1 In: $\mathcal{T}, \mathbf{h}$;
2 $\mathbf{a}, \mathbf{b}, \mathbf{c} = \mathrm{CP}(\mathcal{T}, K = 1)$;
3 $\mathbf{B} = \mathbf{a} \circ \mathbf{b}$;
4 $x, y = max(\mathbf{B})$;
5 $\mathbf{S} = sort(\mathbf{B}_{i=1:|\mathbf{h}|,y})$ //Scores;

Algorithm 11 is used to partition the space of heuristics. In lines 2 and 3 of the algorithm, the two score values for a given heuristic are compared to one another. The heuristic is assigned to the set $\mathbf{h}_{NA}$ if its score is higher (or equal) in the basic frame achieved from $\mathcal{T}_{NA}$ (that is, if $\mathbf{S}_{NA}(h) \geq \mathbf{S}_{IE}(h)$). Otherwise it is assigned to $\mathbf{h}_{IE}$. Note that, equal scores (say, $\mathbf{S}_{NA}(h) = \mathbf{S}_{IE}(h)$) rarely happens. At the end of this procedure, two distinct sets of heuristics, $\mathbf{h}_{NA}$ and $\mathbf{h}_{IE}$, are achieved where each group is associated to NA and IE acceptance methods respectively.

Algorithm 11: Partitioning

1 In: $\mathbf{B}_{NA}, \mathbf{B}_{IE}, \mathbf{S}_{NA}, \mathbf{S}_{IE}$;
2 $\mathbf{h}_{NA} = \{h \in \mathbf{h} \mid \mathbf{S}_{NA}(h) \geq \mathbf{S}_{IE}(h)\}$;
3 $\mathbf{h}_{IE} = \{h \in \mathbf{h} \mid \mathbf{S}_{IE}(h) > \mathbf{S}_{NA}(h)\}$;

4.3.2 Parameter Control via Tensor Analysis

The next two stages of the framework (Algorithm 12 and 13 respectively) are very similar. The only detail which distinguishes the two is that, the first stage (Algorithm 12) is run for a shorter time with randomly chosen heuristic parameter values. These values are chosen from the range {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8} denoted by $p \in \mathbf{P}$. Depending on the nature of a given heuristic (e.g. mutational/ruin-recreate or local search) the value of this parameter determines the degree to which the heuristic affects the candidate solution. For example, given a mutational heuristic, this parameter defines the intensity of mutation (e.g. the number of bits flipped), while in a local search heuristic, this

parameter determines the depth of search (e.g. the number of iterations the heuristic will run over a candidate solution). The goal in this section is to construct a tensor which contains selected heuristic parameter values per heuristic index. Factorising this tensor would then help in associating each heuristic with a parameter value. This is while the final stage of the algorithm (Algorithm 13) runs for a longer time and uses these parameter values for each heuristic instead of choosing them randomly. Despite their similarity, each stage is described in detail here to provide the readers with a clearer picture of the logic

In Algorithm 12, the two sets of heuristics achieved in the previous stage together with their respective score values per heuristic are employed to run a hybrid acceptance hyper-heuristic. For the selected heuristic, a random parameter value is chosen and set (lines 11-12), the heuristic is applied and the relevant acceptance criteria is checked (lines 14-15). The heuristic selection is based on tournament selection. Depending on the tour size, few heuristics are chosen from the heuristic set corresponding to the acceptance mechanism and a heuristic with highest score (probability) is chosen and applied. In case of acceptance, the relevant frame entry is updated (line 16). Since this is a hybrid acceptance algorithm, each acceptance criteria has a budget which is expressed as the number of heuristic calls allocated to the acceptance method. If the acceptance criteria has used its budget, then a new random acceptance criteria is selected (lines 18-20). After continuing this process for a time t_p, the final tensor ($\mathcal{T}_{Param}$) is constructed from collected frames and factorized (exactly in the same manner as in Algorithm 9), the basic frame is computed and the parameter values are extracted as suggested in line 25.

4.3.3 Improvement Stage

In the next phase of the algorithm, the two sets of heuristics achieved in previous stages, together with their score and parameter values are employed to run a hybrid acceptance hyper-heuristic (Algorithm 13). Each acceptance method is given a budget in terms of the maximum number of heuristic calls it is allowed to perform (`callCounter`). Whenever, the acceptance method uses its budget, the algorithm switches to a randomly chosen acceptance method, resetting the budget (line 10-12). Depending on the acceptance criteria in charge, a heuristic is selected (using the tournament selection method discussed above) from the corresponding set (lines 2-5). For instance, if NA is in charge a heuristic is selected from $\mathbf{h}_{NA}$. Later, depending on the nature of the heuristic (mutation, hill climbing or none) the parameter value of the heuristic is assigned (line 6) and the heuristic is applied (line 7). The achieved objective function value is then controlled for acceptance (line 9). This process continues until a time limit ($3 \times t_p$) is reached.

Algorithm 12: Parameter Control

In: $\mathbf{h}, \mathbf{h}_{NA}, \mathbf{h}_{IE}, t_p$;
Initialize tensor frame $\mathbf{M}$ to 0;
$counter = 0$;
while $t < t_p$ **do**
  if $counter = \lfloor |\mathbf{h}|/2 \rfloor$ **then**
    append frame and initialize;
  if *acceptance criteria* $= NA$ **then**
    $h =$ SelectHeuristic($\mathbf{h}_{NA}$);
  else
    $h =$ SelectHeuristic($\mathbf{h}_{IE}$);
  $p_{current} = rand(\{0.1, 0.2, \cdots, 0.8\})$;
  setHeuristicParameter($p_{current}$);
  $f_{current} = f_{new}$;
  $f_{new} =$applyHeuristic($h_{current}$) , $\delta_f = f_{current} - f_{new}$;
  if *Acceptance(δ_f, acceptance criteria)* **then**
    $m_{h,p} = 1$, $counter++$;
  end
  if $callCounter > c$ **then**
    callCounter = 0;
    acceptance criteria = selectRandomAcceptance();
end
Construct final tensor $\mathcal{T}_{Param}$ from collected data;
$\mathbf{a}, \mathbf{b}, \mathbf{c} = \text{CP}(\mathcal{T}_{Param}, K = 1)$;
$\mathbf{B} = \mathbf{a} \circ \mathbf{b}$;
$x, y = max(\mathbf{B})$;
$\mathbf{P} = sort(\mathbf{B}_{i=1:|\mathbf{h}|,y})$;

Algorithm 13: Improvement

while $t < (3 \times t_p)$ **do**
  if *acceptance criteria* $= NA$ **then**
    $h =$ SelectHeuristic($\mathbf{h}_{NA}$);
  else
    $h =$ SelectHeuristic($\mathbf{h}_{IE}$);
  setHeuristicParameter($\mathbf{P}(h)$);
  $f_{new} =$ ApplyHeuristic(h);
  $\delta_f = f_{current} - f_{new}$;
  Acceptance(δ_f, acceptance criteria);
  if $callCounter > c$ **then**
    callCounter = 0;
    acceptance criteria = selectRandomAcceptance();
end

4.4 Experimental Results

4.4.1 Experimental Design

The algorithm proposed here is a multi-stage algorithm where in each stage data samples are collected from the search process in form of tensors. Various approaches can be considered for data collection. While each stage can collect the data and ignore those collected in previous corresponding stages the data collected from various (corresponding) stages can be appended to one another. The former data collection approach has the advantage that collected data reflect the current search status independent from previous search stages allowing the algorithm to focus on the current state. However, ignoring previous data means discarding the knowledge that could have been extracted from experience. In order to assess the two data collection approaches, we employ both data collection approaches. That is, two methods are investigated here, both using the same algorithm (as in Algorithm 7). The only difference between them is that one algorithm (TeBHH 1) the data collection phase of the algorithm ignores previously collected data and over-writes the dataset. In the second algorithm (TeBHH 2) the data collected at each stage is appended to those collected in the same previous stage. Please note that, this does not mean that the data collected in the third stage is appended to those collected in the second stage. Each stage maintains its own dataset and e.g. stage 2 appends its data to the dataset designated for the same stage index.

Regardless of the data collection strategy they employ, both TeBHH 1 and TeBHH 2 have three configurable parameters, namely, the time allocated for data collection phase (t_p), the budget allocated to each acceptance criterion (`callCounter` in Algorithm 13) and the tournament size (employed in Algorithms 12 and 13). For each variant, a range of values are considered. Values considered for the variable t_p are $\{75, 125, 175\}$ seconds. For the call budget, values in $\{|\mathbf{h}|, 2 \times |\mathbf{h}|, 3 \times |\mathbf{h}|\}$ are considered. Also, three different tour sizes have been investigated: $\{2, \frac{|\mathbf{h}|}{2}, |\mathbf{h}|\}$. Each experiment performed with the proposed approach with each combination of those parameter values as the initial setting are indexed successively using the ordering as provided, starting from 1 denoting (75, $|\mathbf{h}|$, 2) to 27 denoting (175,3 $\times$ $|\mathbf{h}|$, $|\mathbf{h}|$).

4.4.2 Selecting The Best Performing Parameter Setting

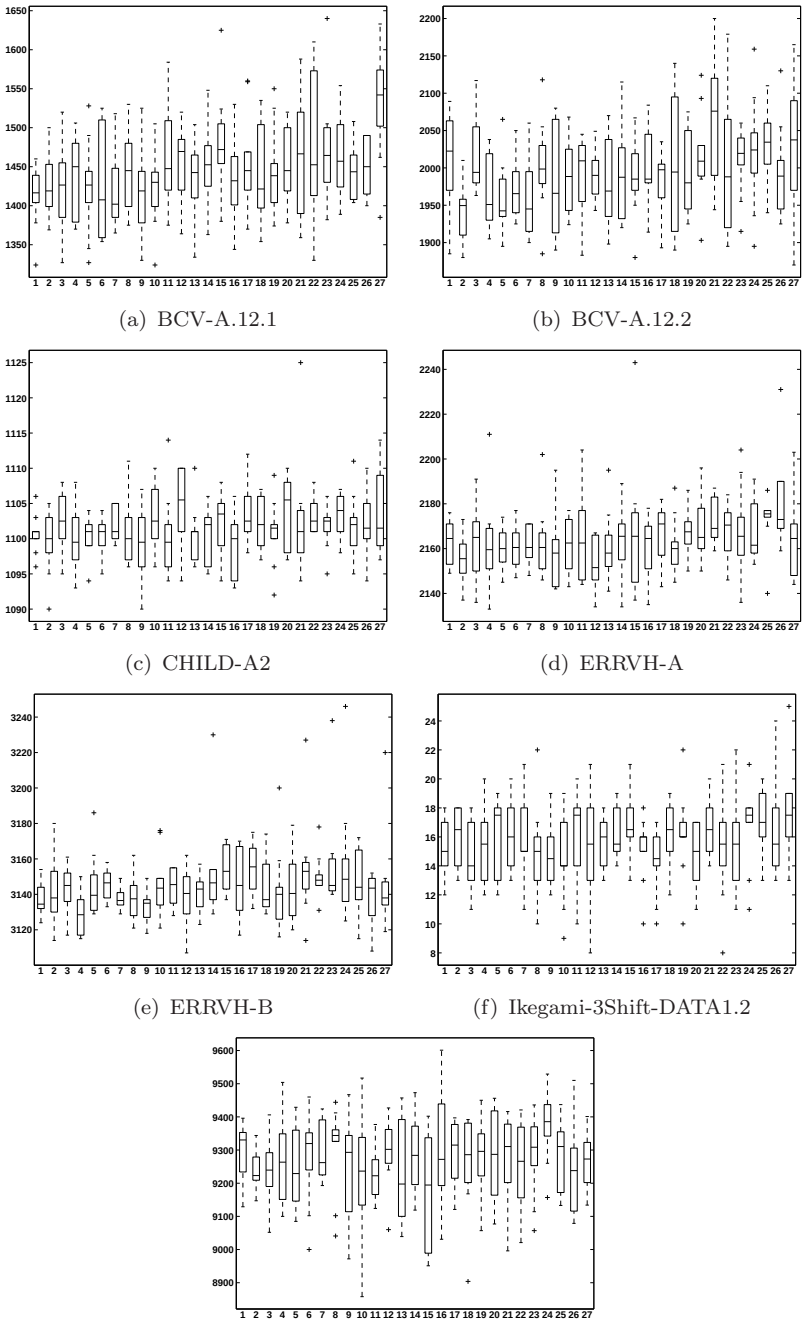
In order to determine the best performing parameter setting, each variant of the algorithm with a parameter value combination was run 10 times where each run terminates after two hours. Apart from detecting the best performing parameter configuration, we would like to know how sensitive the framework is with respect to the parameter settings.

Seven instances were chosen for these experiments which would hopefully cover and represent a whole range of available instances. The chosen instances are: `BCV-A.12.1`, `BCV-A.12.2`, `CHILD-A2`, `ERRVH-A`, `ERRVH-B`, `Ikegami-3Shift-DATA1.2` and `MER-A`.

Figure 4.3 shows the results from these experiments for the TeBHH 1 variant. Although most of the configurations seem to achieve similar performances, there is no other parameter configuration which performs significantly better than the configuration with index 9 (for which $t_p = 175$ seconds, acceptance budget of $3 \times |\mathbf{h}|$ and tournament size of 2) on any of the cases, which is confirmed via a Wilcoxon signed rank test. Ranking all configurations based on the average results across the instances shows that this configuration performs slightly better than the others in the overall. Therefore, these values are chosen for the TeBHH 1 variant. A similar analysis shows that the parameter configuration with index 4 (for which $t_p = 75$ seconds, acceptance budget of $2 \times |\mathbf{h}|$ and tournament size of 2) is more suitable for the TeBHH 2 variant. It has been observed that tournament size of 2 is constantly a winner over the other values for tour size. The apparent conclusion is that both algorithms are very much sensitive to the value chosen for this parameter. Also, a shorter time for data collection in the TeBHH 2 variant makes sense, since it preserves the data collected in previous data collection sessions. The same is not true for TeBHH 1 which overwrites the old data in each stage. Thus, a longer data collection time in case of TeBHH 1 also makes sense. However, when the performance of variants with different data collection time values are compared, the emerging conclusion is that both algorithms are not very sensitive to the chosen value. A similar conclusion can be reached for the value of the acceptance budget.

4.4.3 Comparative Study

In the first round of experiments, an analysis is made as to compare the performance of the two proposed algorithms, namely, TeBHH 1 and TeBHH 2. The second column in Table 4.2 shows the result of this comparison. The statistical test, based on Wilcoxon signed rank test, reveals that, the performance disparity between the two algorithms can vary from one instance to another. For instance, TeBHH 1 outperforms TeBHH 2 on 3 instances significantly. This is while on 2 other instances the situation is the opposite. Also, on 9 instances there is no statistically significance difference between the performance of the two algorithms. This makes sense since the only difference between the two algorithms is the way the dataset is treated throughout the time. While TeBHH 1 overwrites the data with newly collected dataset, TeBHH 2 appends the new data to the old dataset. Thus, it is natural that TeBHH 2 performs similarly to TeBHH 1 since much of their collected data can be similar. Also, the heuristics on all nurse rostering



(a) BCV-A.12.1

(b) BCV-A.12.2

(c) CHILD-A2

(d) ERRVH-A

(e) ERRVH-B

(f) Ikegami-3Shift-DATA1.2

(g) MER-A

FIGURE 4.3: Parameter configuration experiments using TeBHH 1. Each value on the X axis represent the index of the parameter setting of the approach as described at the end of Section 4.4.1. The Y axis represents the objective function values.

Instance	TeBHH 1 vs TeBHH 2	TeBHH 1 vs SRNA	TeBHH 1 vs SRIE
BCV-A.12.2	<	>=	>
BCV-A.12.1	>	>	>
CHILD-A2	>	>	>=
ERRVH-A	>	>	>
ERRVH-B	>=	>	>
ERMGH-B	>=	>	>
MER-A	>=	>	>
Valouxis-1	>=	>	>
Ikegami-3Shift-DATA1.2	>=	>	>
Ikegami-3Shift-DATA1.1	<=	>	>
Ikegami-3Shift-DATA1	<=	>	>
ORTEC01	<	<	>
ORTEC02	>=	<=	>
BCV-3.46.1	<=	>=	>

TABLE 4.2: Statistical Comparison between TeBHH 1, TeBHH 2 and their building block components (SRIE and SRNA). Wilcoxon signed rank test is performed as a statistical test on the objective function values obtained over 20 runs from both algorithms. Comparing algorithm x versus y (x vs. y) $\geq$ ($>$) denotes that x (y) performs slightly (significantly) better than the compared algorithm (within a confidence interval of 95%), while $\leq$ ($<$) indicates vice versa.

instances are quite slow and therefore there is a lack of data which is more the reason that the two algorithms perform similarly.

Overall, combining the entries in Table 4.2 and the minimum objective function value achieved by each algorithm (Table 4.3), it would be fair to say that TeBHH 1 performs slightly better than TeBHH 2. It is to say that it would be safer to refresh the dataset once in a while and handle the current search landscape independent from the experience achieved from other regions of the search landscape.

Subsequent to this conclusion another statistical experiment is conducted to compare the performance of the TeBHH 1 to its building block components, namely, SR-NA and SR-IE. The third and fourth columns in Table 4.2 shows that, given equal values as run time, TeBHH1 performs always better than the SR-IE hyper-heuristic. On only one instance, TeBHH 1 performs slightly (and not significantly) better. As for the comparison between TeBHH 1 and SR-NA, although TeBHH 1 still performs significantly better than SR-NA on the majority of instances, on `ORTEC` instances it performs very poorly.

The results of applying the two proposed algorithms on various nurse rostering instances is shown in Table 4.3. The two algorithms are also compared to various well-known

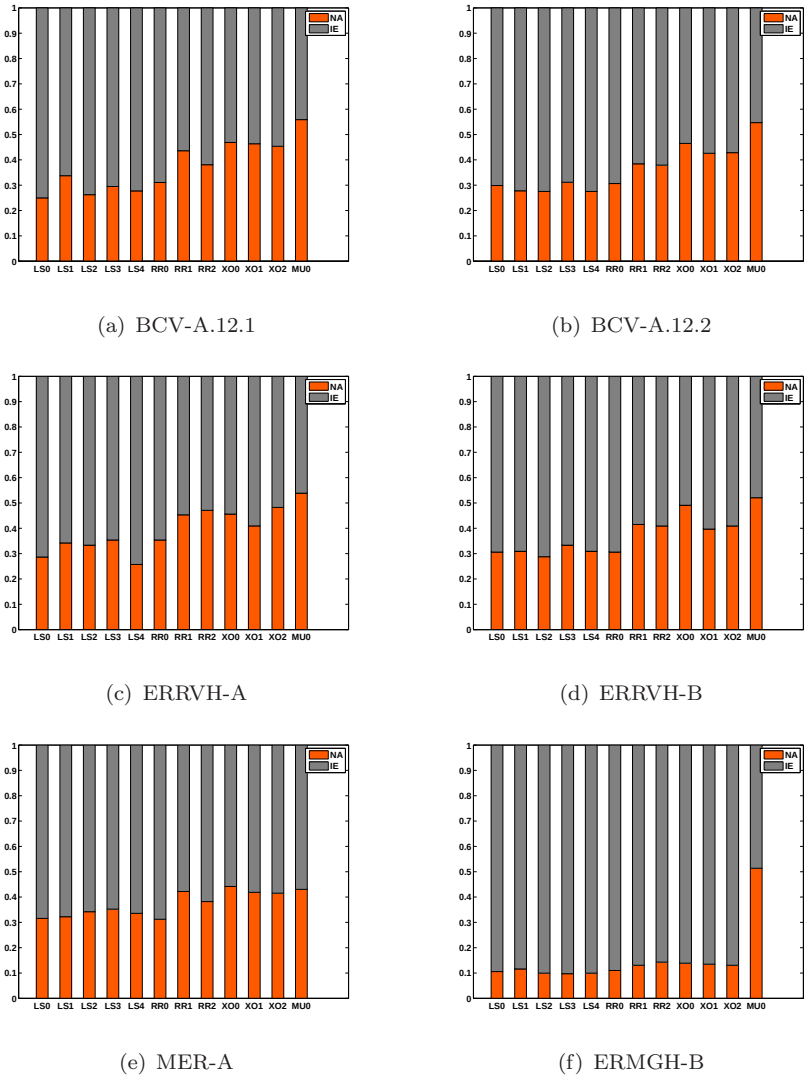
Instance	TeBHH 1	TeBHH 2	Time	Best Known	$\text{Time}_{BestKnown}$
BCV-A.12.2	1858	**1844**	9080	1875	86400
BCV-A.12.1	**1270**	1280	41443	1294	13914
CHILD-A2	**1087**	1089	8229	1095	86400
ERRVH-A	**2118**	2127	9175	2135	86400
ERRVH-B	**3090**	3095	10629	3105	86400
ERMGH-B	1217	**1214**	5	1355	86400
MER-A	8810	**8779**	22008	8814	86400
Valouxis-1	**20**	**20**	3184	**20**	17
Ikegami-3Shift-DATA1.2	8	9	-	**3**	2820
Ikegami-3Shift-DATA1.1	6	8	-	**3**	2820
Ikegami-3Shift-DATA1	6	3	-	**2**	21600
ORTEC01	285	280	-	**270**	69
ORTEC02	290	290	-	**270**	105
BCV-3.46.1	3282	3283	-	**3280**	20764

TABLE 4.3: Comparison between the two proposed algorithms and various well-known (hyper-/meta)heuristics. The second and third columns contain the best objective function values achieved by TeBHH 1 and TeBHH 2 respectively. Fourth column gives the earliest time (seconds) among all the runs (20) in which the reported result has been achieved. Same quantities (minimum objective function values and earliest time it has been achieved) are also reported for compared algorithms in columns five and six.

algorithms. While some of these algorithms (like the one in [127]) are general-purpose search algorithms, some others are specifically designed to solve the given instance.

On the first seven instances in Table 4.3, both TeBHH 1 and TeBHH 2 outperform compared algorithms in terms of minimum objective function value. On the instance `Valouxis-1`, both algorithm can achieve the best known result (20), although much later than the state-of-the-art [216]. Similarly, on `Ikegami` and `ORTEC` instances as well as `BCV-3.46.1`, the state-of-the-art performs better. The algorithms which solve aforementioned instances are instance-specific and designed to solve a group of highly related instances, such as those in the `Ikegami` family. Overall, the two algorithms perform well on provided instances and produce new best known results for some of them (the first seven instances).

Figure 4.4 shows the distribution of heuristics to disjoint sets $\mathbf{h}_{NA}$ and $\mathbf{h}_{IE}$ throughout the 20 runs for some of the problem instances. Each run consists of up to 27 stages and in each stage, the set of heuristics is partitioned using tensor factorisation. The histograms in Figure 4.4 is built by counting the number of times a heuristic is associated with the NA and IE move acceptance methods throughout all the runs for a given instance. The histograms vary from one instance to another. The difference between histograms are sometimes minor (as it is between histograms of `BCV-A.12.1` and `BCV-A.12.2`) and



(a) BCV-A.12.1

(b) BCV-A.12.2

(c) ERRVH-A

(d) ERRVH-B

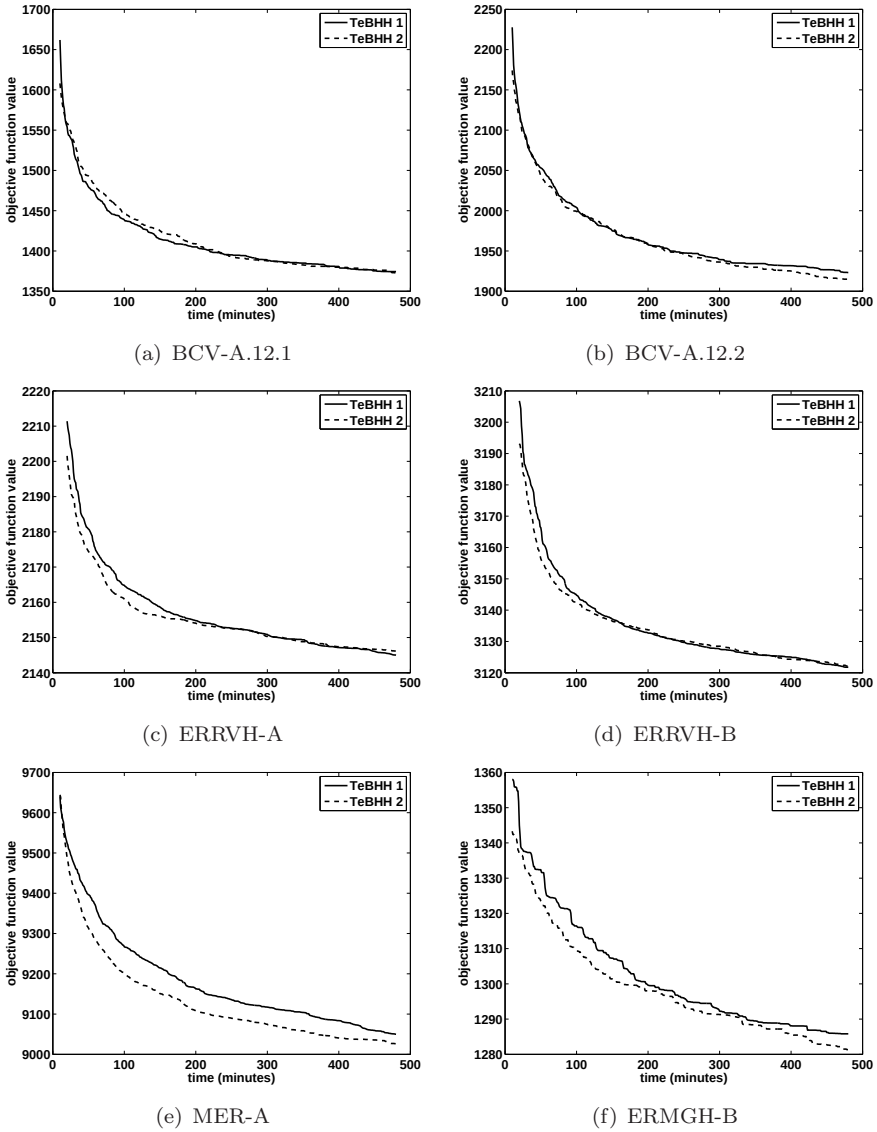
(e) MER-A

(f) ERMGH-B

FIGURE 4.4: Distribution of heuristics in $\mathbf{h}_{NA}$ and $\mathbf{h}_{IE}$ partitions.

sometimes major (as is the case for the instance `MER-A` compared to the rest). However, the common pattern among most of these partitions is that the heuristic MU0 has been equally associated to both sets. Although the framework clearly shows the tendency to assign heuristics more to the $\mathbf{h}_{IE}$ set rather than $\mathbf{h}_{NA}$, Ruin Recreate and Crossover heuristics are likelier to be assigned to $\mathbf{h}_{NA}$ compared to local search heuristics. Since the heuristics in nurse rostering domain all deliver feasible solutions, it makes sense that the framework tries to increase the possibility of diversification by assigning diversifying heuristics to NA acceptance method.

During the improvement stage (Algorithm 13), the algorithm allocates a time budget to each acceptance method. Whenever this budget is consumed, the algorithm switches to a randomly chosen acceptance criteria. Since the tensor analysis is likelier to assign diversifying heuristics to $\mathbf{h}_{NA}$ (keeping the intensifying heuristics in $\mathbf{h}_{IE}$), it thus performs



(a) BCV-A.12.1 (b) BCV-A.12.2

(c) ERRVH-A (d) ERRVH-B

(e) MER-A (f) ERMGH-B

FIGURE 4.5: The progress of the objective function value on average, obtained from 20 runs of TeBHH 1 and TeBHH 2.

similar to a higher level Iterated Local Search (ILS) algorithm where each intensification step is followed by a diversification one. That in turn results in continuous improvement of the solution as is confirmed in Figure 4.5 for TeBHH 1 and TeBHH 2 respectively.

The progress plots corresponding to TeBHH 1 and TeBHH 2 (Figure 4.5) show that on many instances (particularly on `BCV-A.12.1`, `BCV-A.12.2` and `ERMGH-B`) both algorithm are rarely stuck in local optima. This is a good behaviour showing that given longer run times (similar to the experiments in [127]) there is a high likelihood that the algorithms proposed here provide better results with even lower objective function values.

4.5 Summary

Nurse rostering is a real-world NP-hard combinatorial optimisation problem. A hyper-heuristic approach which benefits from an advanced data science technique, namely, tensor analysis is proposed in this chapter to tackle a nurse rostering problem. The proposed approach embedding a tensor-based machine learning algorithm is tested on well-known benchmark problem instances collected from hospitals across the world. Two different remembering mechanisms, *memory length*s are used within the learning algorithm. One of them remembers all relevant changes from the start of the search process, while the other one refreshes its memory every stage. The results indicate that 'forgetting' is slightly more useful than remembering all. Hence, a strategy that decides on the memory length adaptively would be of interest as a future work. In this chapter, the tensor-based hyper-heuristic with memory refresh generated new best solutions for four benchmark instances and a tie on one of the benchmark instance.

The proposed approach cycles through four stages continuously and periodically, employing machine learning in the first three stages to configure the algorithm to be used in the final stage. The final stage approach itself is an iterated bi-stage algorithm cycling through two successively invoked hyper-heuristics, namely SR-NA and SR-IE. Depending on the problem instance and even a trial, the nature of the low level heuristics allocated to each stage (hence the move acceptance) could change. However, experiments indicate that mutational heuristics often can get allocated to either of the hyper-heuristics. SR-NA allows worsening moves while SR-IE does not. Hence, the final stage component of the tensor-based hyper-heuristic acts as a high level Iterated Local Search algorithm [38], providing a neat balance between intensification and diversification using the appropriate low level heuristics which are determined automatically during the search process, resulting in continuous improvement in time. The overall approach is enabled to extract fresh knowledge periodically throughout the run time, which is an extremely desired behavior in life-long learning. Thus, the tensor-based hyper-heuristic proposed here can be considered in life-long learning applications.

So far, we have coupled the tensor learning approach to hyper-heuristics. Hyper-heuristics operate as high level decision making strategies and leave traces which are highly abstract. The experimental results in both this chapter and the previous chapter (Chapter 4) indicate that the proposed approach performs very well on highly abstract data. To continue with our assessment of this learning approach and decide whether or not tensor analysis can be applied to trace data with lower levels of abstraction, in the next chapter, it has been used to analyse the trace of a hyper-heuristic which is somewhat more detailed (and hence less abstract) compared to those extracted from the hyper-heuristics in this chapter as well as the previous chapter.

Chapter 5

A Tensor Analysis Improved Genetic Algorithm for Online Bin Packing

In this chapter, we move lower in data abstraction level. We use tensor analysis to mine the data collected from the trace of a standard genetic algorithm hyper-heuristic when applied to the one dimensional bin packing problem. Compared to the hyper-heuristics in previous chapters, the hyper-heuristic considered here has full access to the heuristic design. Indeed, the genetic algorithm hyper-heuristic evolves/generates heuristics rather than selecting available low level heuristics. The tensor analysis approach in this chapter is employed to extract useful patterns from heuristics generated by the hyper-heuristic. Thus, the amount of information available to the factorisation procedure is more compared to previous chapters and the tensorial data is lower in abstraction level. The patterns extracted by the hyper-heuristic are used to adjust mutation probabilities in the genetic algorithm.

5.1 Introduction

In many situations, decisions must be made despite lack of knowledge of the future allowing the computation of the full effects of the decisions. In such cases, it is usual to have some kind of heuristic 'dispatch policy' to make decisions. Usually, such heuristics are produced by an expert in a domain carefully designing some decision procedure. Often, even an expert requires a great deal of trial and error - though the errors are rarely reported, and so a misleading impression is given suggesting that creation of heuristics is not a time-consuming process. Of course, such difficulties are well-known,

and so there have been various attempts to automate the production of heuristics (e.g. for some recent work see [14, 15, 83]).

In [29], an approach for the automatic creation of heuristics is given that might be viewed as a form of parameter tuning [231], but applied with a much larger number of parameters than usually considered. The large number of parameters arise from a 'brute force' representation of the heuristic as a matrix covering the various potential decisions. That is, it defines a policy in terms of the 'features' available at each decision point. This is done in the style of an 'index policy' (e.g., [232]) in that each potential outcome is given a score separately of other outcomes and the largest score is selected.

In this chapter, we particularly study the well-known online bin-packing problem [233, 234], creating a policy that is based on using a (large) matrix[1] of 'heuristic scores'. The policy matrix can be viewed as a heuristic with many parameters. Alluding to this, the framework in [29] allows the use of an optimiser for 'Creating Heuristics viA Many Parameters' (CHAMP) and online bin packing simulator that can be used as an evaluation function for a given policy on a given problem instance. Packing problem instances are specified in terms of a specified bin capacity and a stochastically generated sequence of item sizes taken from a specified range. For specific instance generators, good policies are found using a Genetic Algorithm (GA) as the optimiser under the CHAMP framework to search the space of matrices, with the matrix-based policies being evaluated directly by packing a (large) number of items.

In this chapter, we take the GA optimiser of the CHAMP framework as the basis to investigate the role of tensor analysis in heuristic optimisation. We propose the integration of the tensor analysis approach into the CHAMP framework to generate mutation probabilities for each locus of a chromosome, also referred to as individual, representing a candidate solution. In our approach, within the GA algorithm in CHAMP, the trail of high quality solutions, where each solution has a matrix form, is represented as a 3^{rd} order tensor. Factorising such a tensor reveals the latent relationship between various chromosome locations through identifying common subspaces of the solutions where mutation is more likely to succeed in producing better offspring. In addition to subspace learning, one would expect a powerful data mining approach to discover the related genes. Possession of such information should naturally result in having similar probability values for closely related genes. The experiments in this chapter show that tensor factorisation achieves this objective and identifies genes which should have similar mutation likelihoods due to their close relationship.

[1]Here, the term 'matrix' is used as a convenience for a 2-d array; there is no implication of it being used for matrix/linear algebra

The tensor analysis approach is applied to a range of bin packing problems and the results are compared to those achieved by the original CHAMP framework [29] and subsequent studies. In this chapter, first the one dimensional online bin packing problem and its instances are described in Section 5.2. Since the policy matrix representation of candidate solutions is used in the proposed approach, this representation is discussed in Section 5.3 followed by a description of the CHAMP framework in full detail in Section 5.4. Subsequent to these preliminaries, the tensor-based approach and the way it has been integrated to the CHAMP framework is described in Section 5.6. Experimental results and discussion are also provided in Sections 5.7 and 5.8.

5.2 Online Bin Packing Problem

In online one dimensional bin packing, each bin has a capacity $C > 1$ and each item size is a scalar in the range $[1, C]$. More specifically, each item can be chosen from the range $[s_{min}, s_{max}]$ where $s_{min} > 0$ and $s_{max} \leq C$. The items arrive sequentially, meaning that the current item has to be assigned to a bin before the size of the next item is revealed. A new empty bin is always available. That is, if an item is placed in the empty bin, it is referred to as an open bin and a new empty bin is created. Moreover, if the remaining space of an open bin is too small to take in any new item, then the bin is said to be closed.

The uniform bin packing instances produced by a parametrised stochastic generator are represented by the formalism: UBP(C, s_{min}, s_{max}, N) (adopted from [29]) where C is the bin capacity, s_{min} and s_{max} are minimum and maximum item sizes and N is the total number of items. For example, UBP$(15, 5, 10, 10^5)$ is a random instance generator and represents a class of problem instances. Each problem instance is a sequence of 10^5 integer values, each representing an item size drawn independently and uniformly at random from $\{5, 6, 7, 8, 9, 10\}$. The probability of drawing exactly the same instance using a predefined generator of UBP(C, s_{min}, s_{max}, N) is $1/(s_{max} - s_{min} + 1)^N$, producing an extremely low value of $6^{-100000}$ for the example. Note that there are various available instances in the literature [235, 236], however, these instances are devised for offline bin packing algorithms and usually consist of a small number of items.

There are two primary ways of utilising random instance generators. A common usage is to create a generator and then generate around a few dozen instances which then become individual public benchmarks. Consequently, methods are tested by giving results on those individual benchmark instance. In our case, the aim is to create heuristics that perform well on average across all instances (where an instance is a long sequence of item sizes) from a given generator. (Hence, for example, we believe it would not serve any

useful purpose to place our specific training instance sequences on a website.) A related note is that it is important to distinguish two quite different meanings of 'instance': either a specific generator, or a specific sequence of items. An instance in the sense of a generator generally contains a Psuedo-Random Number Generator (PRNG) which needs be supplied with a seed in order to create an instance in the sense of a specific sequence of item sizes.

There are well established heuristics for this problem among which First Fit (FF), Best Fit (BF) and Worst Fit (WF) [237–239]. The FF heuristic tends to assign items to the first open bin which can afford to take the item. The BF heuristic looks for the bin with the least remaining space to which the current item can be assigned. Finally, WF assigns the item to the bin with the largest remaining space. Harmonic-based online bin packing algorithms [240, 241] provide a worst-case performance ratio better than the other heuristics. Assuming that the size of an item is a value in (0,1], the Harmonic algorithm partitions the interval (0,1] into non-uniform subintervals and each incoming item is packed into its category depending on its size. Integer valued item sizes can be normalised and converted into a value in (0,1] for the Harmonic algorithm.

Although we often refer to the choices for UBP as instances it should be remembered that they are instances of distributions and not instances of a specific sequence of items; the actual sequence is variable and depends on the seed given to the random number generator used within the item generator. That is, within the instance generator one can use different seed values to generate a different sequence of items each time the same UBP is generated. Indeed, this is the case when we test our approach as it will be seen in the coming sections.

There are various criteria with which the performance of a bin packing solution can be evaluated. Some of these are enlisted below.

Bins-Used, B: The number of bins that are used. B is an integer value which tends to increase as larger number of items (N) are considered.

Average-Fullness, F_{af}: Considering that bin t has a fullness equal to f_t, $t \in \{1, \ldots, B\}$ then F_{af} is the value of the occupied space, averaged over the number of used bins.$F_{af} = 1/B \sum_t f_t$

Average-Generic-Fullness, F_{gf}: This value gives some insight into the variation of resulting fullness between bins. $F_{gf} = 1/B \sum_t f_t^2$

Average-Perfection, F_{ap}: This measure is an indication of how successful the heuristic is in packing the bins perfectly. $F_{ap} = 1/B \sum_{t,f_t=1} f_t$

In our study, average bin fullness (F_{af}) is considered as the fitness (evaluation/objective) function.

5.3 Policy Matrix Representation

A packing policy can be defined by a matrix of heuristic values (called policy matrix). That is, we have a matrix structure in which for each pair (r,s) a score $W_{r,s}$ is provided which gives the priority of assigning the current item size s to a remaining bin capacity r. Given such a matrix, our approach is to simply scan the remaining capacity of the existing feasible open bins and select the first one to which the highest score is assigned in the matrix (Algorithm.14). A feasible open bin is an open bin with enough space for the current item size, say $r \geq s$ and includes the always-available new empty bin. The integer scores are chosen from a specific range $W_{r,s} \in [w_{min}, w_{max}]$.

Algorithm 14: Applying a policy matrix on a bin packing instance

```
In : W : score matrix;
for each arriving item size s do
    maximumScore = 0;
    for each open bin i in the list with remaining size k do
        if k > s then
            if W_{k,s} > maximumScore then
                maximumScore = W_{k,s};
                maximumIndex = i;
            end
        end
    end
    assign the item to the bin maximumIndex;
    if maximumIndex is the empty bin then
        open a new empty bin and add to the list;
    end
    update the remaining capacity of maximumIndex by subtracting s from it;
    if remaining capacity of maximumIndex is none then
        close the bin maximumIndex;
    end
end
```

It is clear that the policy matrix is a lower triangular matrix as elements corresponding to $s > r$ do not require a policy (such an assignment is simply not possible). Therefore, only some elements of the policy matrix which correspond to relevant cases for which a handling policy is required are considered. We refer to these elements as active entries while the rest are inactive elements. Inactive entries represent a pair of item size and remaining capacity which either can never occur or are irrelevant.

The active entries along each column of the policy matrix represent a policy with respect to a specific item size and the scores in each column is independent from that of other columns as the policy for a certain item size can be quite different than that of other item sizes.

In order to further clarify how a policy matrix functions, an example is given here. The policy matrix in Figure 5.1 is evolved to solve packing instances generated by $UBP(15,5,10)$. Assume that, during the packing process, an item of size 5 arrives. This item size corresponds to the fifth column in the given policy matrix. The entries of this column represent the set of scores which are associated to each possible remaining bin capacity for the current item size. Assume that, currently, only bins with remaining capacities of 9 and 10 are open. As always, the empty bin is also available for item placement. The scores associated with remaining bin capacities 9 and 10 are 4 and 1 respectively. The empty bin has a score of 2. Since the bin with the remaining capacity 9 has the highest score, the item is placed in this bin.

In all policy matrices, the last row represents the scores assigned to the empty bin for different item sizes. Suppose that, in the previous example, the score associated to the empty bin is 7 (instead of 2 in Figure 5.1). In this case, the item would be no longer put in bin with remaining capacity 9. Instead it would be placed in the empty bin (bin with remaining capacity 15) and a new empty bin would be opened immediately.

Ties can occur and the tie breaking strategy employed here is first fit. As an example, assume that the arriving item has a size 8. Therefore, in order to determine which bin to choose for item placement, the scores in column 8 will be investigated. Assume that currently there are open bins with all possible remaining bin capacities as well as the always available empty bin. Scanning the scores, bins with remaining capacities 8 and 10 emerge as top scoring ones because they both have the highest score which is 7. However, due to the first fit tie breaking strategy, the first bin from the top is chosen and the item is put in the bin with remaining capacity 8.

5.4 A Framework for Creating Heuristics via Many Parameters (CHAMP)

A policy matrix represents a heuristic (scoring function). Changing even a single entry in a policy matrix creates a new heuristic potentially with a different performance. Assuming that each active entry of a policy matrix is a parameter of the heuristic, then a search is required to obtain the best setting for many parameters (in the order of $O(C^2)$).

r\s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5:	.	.	.	.	6	.	.	.	.	.	.	.	.	.	.
6:	.	.	.	.	3	7	.	.	.	.	.	.	.	.	.
7:	.	.	.	.	7	4	2	.	.	.	.	.	.	.	.
8:	.	.	.	.	1	2	2	7	.	.	.	.	.	.	.
9:	.	.	.	.	4	3	6	4	5	.	.	.	.	.	.
10:	.	.	.	.	1	7	3	7	2	4	.	.	.	.	.
11:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
12:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
13:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
14:	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15:	.	.	.	.	2	5	6	5	3	4	.	.	.	.	.

FIGURE 5.1: An example of a policy matrix for $UBP(15, 5, 10)$

In this chapter, we use the framework for creating heuristics via many parameters (CHAMP) consisting of two main components operating hand in hand: an *optimiser* and a *simulator* as illustrated in Figure 5.2. CHAMP separates the optimiser that will be creating the heuristics and searching for the best one from the simulator for generality, flexibility and extendibility purposes. The online bin packing simulator acts as an evaluation function and measures how good a given policy is on a given problem.

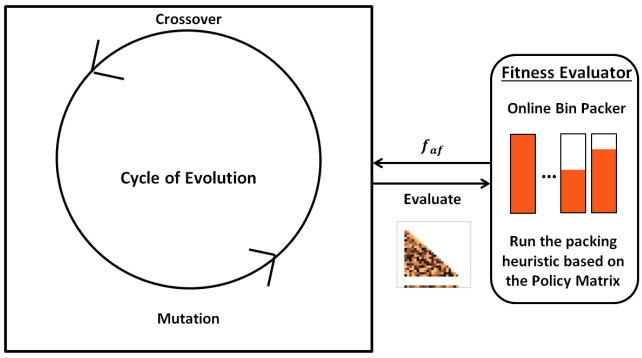


FIGURE 5.2: CHAMP framework for the online bin packing problem.

As is evident from Figure 5.2, policy matrices are evolved using a Genetic Algorithm (GA) as the optimiser component of the CHAMP framework. Each individual in the GA framework represents the active entries of the score matrix and therefore each gene carries an allele value in $[w_{min}, w_{max}]$. The population of these individuals undergoes the usual cyclic evolutionary process of selection, recombination, mutation and evaluation. Each individual is evaluated by applying it to the bin packing problem instance as was shown in Algorithm 14 and the fitness value is one (or more) of the measures in Section 5.2. The settings for the GA optimiser is given in Table.5.1.

The GA and the fitness evaluator communicate through the matrices; the GA saves an individual into a matrix and invokes the online bin packing program. The packing algorithm uses the matrix as a policy and evaluates its quality using an instance produced

TABLE 5.1: Standard GA parameter settings used during training

Parameter	Value
No. of iterations	200
Pop. size	$\lceil C/2 \rceil$
Selection	Tournament
Tour size	2
Crossover	Uniform
Crossover Probability	1.0
Mutation	Traditional
Mutation Rate	$1/ChromosomeLength$
No. of trials	1

by the instance generator UBP(C,s_{min},s_{max},10^5). The total number of bins used while solving each training case is accumulated and then saved as the fitness of the individual into another file for GA to read from. The initial population is randomly generated unless it is mentioned otherwise and the training process continues until a maximum number of iterations is exceeded.

A hyper-heuristic operates at a domain-independent level and does not access problem specific information (e.g. see [15]), thus, the framework we use, as shown in Figure 5.2, follows the same structure. However, in contrast to the hyflex implementation of hyper-heuristics, here, the hyper-heuristic has access to the heuristic design. Therefore, the domain barrier is considered to be breached and the amount of information available to the higher level strategy is less abstract compared to the same in HyFlex. In this chapter, in contrast to the previous work [29], several instance generators for the one dimensional online bin packing problem have been considered for experiments and N is kept the same during training and testing phases. Moreover, several variants of the policy matrix evolution scheme has been considered each differing with others in the initialisation scheme, and the upper bound for score range (w_{max}).

5.5 Related Work on Policy Matrices

There is a growing interest on automating the design of heuristics (e.g. for some recent work see [14, 15]). In [29], a GA framework was proposed in which policy matrices as described above were evolved, resulting in automatic generation of heuristics in form of index policies. In addition to this original study, there has been a number of studies related to this topic. For example, in [242], an approach based on policy matrices was proposed for analysing the effect of the mutation operator in Genetic Programming (GP) in a regular run using online bin packing.

In [243] dimensionality reduction was considered for policy matrices in that they were derived from one dimensional vectors (say, policy vectors). Evolving policy vectors in a fashion similar to the evolution of policy matrices was shown to produce high quality solutions. In [244], policy matrices are seen as heuristics with many parameters and are approached from a parameter tuning perspective. The Irace package was used to tune policies during training. Trained policies were then tested on unseen instances with performances close to that of the GA framework and significantly better than the man-made heuristics. In this chapter, we use the same GA with the same settings as described in [29], however, we present a tensor-based approach for improving its performance via an adaptive locus-based mutation operator, instead of using a generic one.

5.5.1 Apprenticeship Learning for Generalising Heuristics Generated by CHAMP

In [45], Apprenticeship Learning (AL) method (as in Chapter 2.3.2) was proposed to increase the generality level of the CHAMP framework. Although the policy matrix approach in the CHAMP framework [29] was effective at generating heuristics with better performance than the standard ones, it had the drawback of directly only applying to a specific set of values for the bin capacity and range of item sizes. The AL method in [45] collects the data from applying a high quality heuristic generated by the CHAMP framework on small instances. The apprenticeship learning method (described in Section 2.3.2) is then used to build a generalisable model of the data. The model is then used as a packing policy (heuristic) on different, larger instances. The tensor-based approach proposed here is compared to the AL-based method. Also, the AL-based method is interesting considering that it generalises a hyper-heuristic using machine learning, similar to the tensor-based approach of this study. Therefore, it seems suitable to give a brief introduction to the AL-based approach in this section.

In the AL-based method, each search state can be seen and described as a feature set (as in Eq. 2.1 or 2.2) with which a generalized model can be constructed. In order to achieve a desirable performance, the extracted features should be instance independent. That is, they should not be dependent on the absolute values of the item size (s), bin capacity (C) and minimum or maximum item size (s_{min} or s_{max}), but rather to depend on relative sizes. Table 5.2 shows the list of considered features along with their formal and verbal descriptions. The features in Table 5.2 are extracted for each open bin on the arrival of each new item. In the dataset, each record is labelled as either 1 (if the bin is selected) or 0 (if the bin is rejected).

TABLE 5.2: Features of the search state. Note that the UBP instance defines the constants C, s_{min}, and s_{max} whereas the variables are s the current item size, and r the remaining capacity in the bin considered, and r' is simply $r - s$.

feature	description
$(s - s_{min})/(s_{max} - s_{min})$	normalized current item size
r/C	normalized remaining capacity of the current bin
s/C	ratio of item size to bin capacity
s/r	ratio of item size to the current bin's remaining capacity
r'/C	normalized remaining capacity of the current bin after a feasible assignment
$r'/(s_{max} - s_{min})$	ratio of remaining capacity of the current bin after a feasible assignment to the range of item size

Having formulated the feature representation, it is now possible to use expert policies to extract features and their corresponding labels for each search state. That is, we assume that we are in possession of a set of n expert policies $\{\pi_e^1, ..., \pi_e^n\}$ in one dimensional on-line bin packing problem domain. These expert policies are obtained by the policy generation method discussed in Section.5.4. Each expert policy corresponds to a certain UBP. We run each expert policy once, on it's corresponding UBP for a certain and fixed number of items $N = 10^5$. While running, expert features, ϕ_e^t given in Table 5.2, are extracted for each state of the search (t). Here, ϕ_e^t is a r dimensional vector of features where r is the number of features representing a search state. At the end of each run for a policy π_e^i we will have a set of demonstrations like:

$$\mathcal{D}_{\pi_e^i} = \{(\phi_e^t, a_t) | \pi_e^i\} \tag{5.1}$$

where a_t is the action at step t. The demonstration sets for all training policies are then merged together to form a dataset.

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_{\pi_e^i} \tag{5.2}$$

Having the feature vectors and their associated labels, we employ a k-means clustering algorithm (Section 2.3.1) to cluster the feature vectors of each class. The generated clusters constitute a generalised model of the actions of various expert heuristics.

For an unseen problem instance (a new, unseen UBP with different range of item sizes and bin capacity), at each state of the search, say, on the arrival of each new item, for each open bin, the state features are extracted ($\phi^{t'}$) and the closest matching centroid to the current feature vector in terms of cosine similarity is found. In case the centroid

has a label 1 the bin is selected for the item assignment according to a probability. The probability is chosen to be 0.99 and is considered to introduce randomness to the decision making process. Eq.5.3 illustrates the decision making mechanism of the generalized policy, given a feature vector for a bin and a set of centroids.

$$\pi_g = \{a_{x_j} \in \{0,1\} \mid \underset{j}{\operatorname{argmin}}\, d(\phi_{x_j}, \phi^{t'}) \ , \ \phi_{x_j} \in \mathcal{D}\} \tag{5.3}$$

Here, π_g is the generalized policy, the subscript x_j indicates the jth centroid obtained by the k-means clustering algorithm, a_{x_j} is the action (label) which is associated to the centroid j and d is the distance metric which is given in Eq.5.4.

$$d(\phi_{x_j}, \phi^{t'}) = 1 - \frac{\sum_r \phi_{x_j} \cdot \phi^{t'}}{\sqrt{\sum_r {\phi_{x_j}}^2} \cdot \sqrt{\sum_r {\phi^{t'}}^2}} \tag{5.4}$$

5.6 Proposed Approach

There is a wide variety of population-based approaches, solving computationally hard problems, which are referred to as 'knowledge-based' evolutionary computation methods. Knowledge can be extracted and used in many ways in various stages of the evolutionary process. For instance, Knowledge-Based Genetic Algorithm (KBGA) [245] used problem domain knowledge to produce an initial population and guide the operators of a Genetic Algorithm (GA) using that knowledge at all the stages of the evolution. In [246] problem specific 'knowledge' was represented in form of ground facts and training examples of Horn clauses. This knowledge is exploited in a GA for inductive concept learning and is used in mutation and crossover operators to evolve populations of if-then rules. [247] employed prior problem specific 'knowledge' to generate locus level bias probabilities when selecting allele for crossover, resulting in a Knowledge-Based Nonuniform Crossover (KNUX). In [248], a mutation operator was designed based on knowledge capturing the distribution of candidate solutions in Extremal Optimisation context. This method was successfully applied to PID tuning. The approach proposed in [249] utilized rough set theory to explore hidden knowledge during the evolutionary process of a GA. The extracted knowledge is then used to partition the solution space into subspaces. Each subspace is searched using a separate GA. In this section, we use tensor analysis to extract prolem specific knowledge using which mutation probabilities in the genetic algorithm of the CHAMP framework are enhanced/improved.

Evolutionary algorithms are among many approaches which produce high dimensional data. The search history formed by GA can be turned into multi-dimensional data in

a similar fashion to the search history of a hyper-heuristic as described in Chapters 3 and 4. For example, collecting high quality individuals (candidate solutions) from the populations in several successive generations while GA operates, naturally, yields a 3^{rd}-order tensor, representing the changing individuals in time. Moreover, the candidate solution in the CHAMP framework are two dimensional matrices. Put together, these matrices naturally form a 3^{rd}-order tensor. This is precisely what has been done here as is described below.

In the original framework [29], policy matrices are produced using GA in a train and test fashion. The evolutionary cycle is performed for a given stochastic sequence generator (UBP) resulting in a policy matrix for that UBP. At the beginning, a random population of policy matrices is generated. At each generation, mutation and crossover are applied to the individuals (policy matrices). Each individual representing a packing policy/heuristic is then handed over to a separate evaluator (bin packer) which applies the policy to a stream of items, returning the fitness (F_{af}) as feedback. The cycle of evolution continues until the stopping criterion is met. Our method modifies the training procedure as illustrated in Figure 5.3. During every 5 generations, a tensor ($\mathcal{T}$) containing the top 20% individuals (policy matrices) is constructed and factorized into its basic factor, producing a basic frame. The elements of the basic frame are used as mutation probabilities for the next 5 generations from which a new tensor is constructed. Subsequent to training, the best individual is then tested on several unseen instances for evaluation. Throughout this chapter, the original CHAMP framework in [29] will simply be denoted by GA whereas the tensor-based variant proposed here will be denoted by GA+TA.

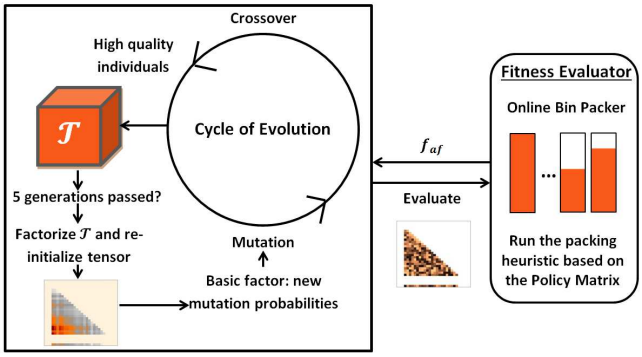


FIGURE 5.3: The GA+TA framework

The tensor $\mathcal{T}$ has the size $C \times C \times R$ where R is the number of the top 20% individuals and C is the bin capacity as described in Section 5.2. The order according to which the policy matrices are put into the tensor is precisely the order in which they are generated by the GA framework. This tensor is then factorized where K in Eq.2.7 is set to 1 resulting in a simplified expression of the factorisation (Equation 5.5). That is, the

original tensor $\mathcal{T}$ is approximated by $\hat{\mathcal{T}}$ as the following

$$\hat{\mathcal{T}} = \lambda \, \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \tag{5.5}$$

where the length of the vectors **a**, **b** and **c** are C, C and R respectively. As depicted in Figure 5.4, the outer product of vectors **a** and **b** results in a basic frame **B** which is exactly the shape of a policy matrix with the size $C \times C$ is used with $k = 1$ to produce **B**). The difference between **B** and a policy matrix is that instead of containing integer score values in the range of $[w_{min}, w_{max}]$, it contains real values between 0 and 1. These values point towards regions in policy matrices where change of score values has been a common pattern among good quality matrices.

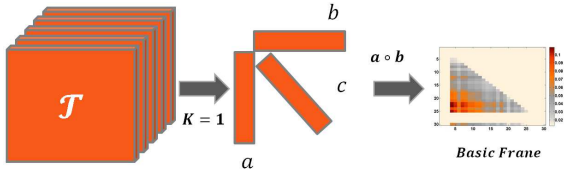


FIGURE 5.4: Extracting the basic frame for $K = 1$ in Eq.2.7.

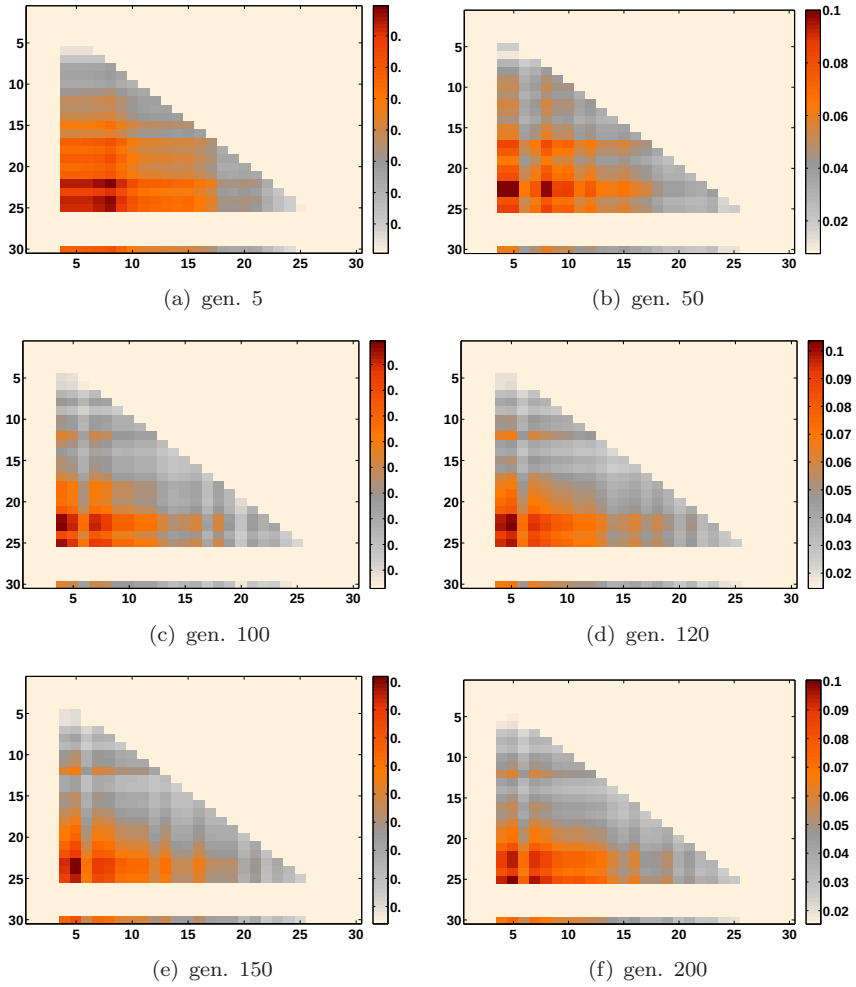
Thus, the values in **B** are perceived as mutation probability of each locus for the next 5 generations. That is, during the next 5 generations, a gene indexed (i, j) is mutated with a probability $\mathbf{B}(i, j)$. The initial mutation probabilities are fixed as $\frac{1}{chromosomeLength}$ for the first 5 generations. Data collection for tensor construction occurs at the same time when the generated basic frame **B** has been applied.

5.7 Experimental Results

5.7.1 Experimental Design

The setting used for the GA framework is illustrated in Table 5.1. As discussed in Section 5.3, the scores in policy matrices are chosen from the range $[w_{min}, w_{max}]$. In our experiments $w_{min} = 1$ and w_{max} is equal to the maximum number of active entries along the columns of the policy matrix (i.e. for the policy matrix in Figure 5.1, $w_{max} = 7$).

For tensor operations, Matlab Tensor Toolbox [206] has been used. The GA framework is implemented in the C language. In order to use the toolbox, the Matlab DeployTool has been used to generate an executable of the Matlab code. This executable is then called when necessary from the C code without a need to load the Matlab environment. The approach proposed in this chapter is compared to the original CHAMP framework



(a) gen. 5 (b) gen. 50

(c) gen. 100 (d) gen. 120

(e) gen. 150 (f) gen. 200

FIGURE 5.5: Various basic frames achieved in different stages of the search for an instance generated by $UBP(30, 4, 25)$. The basic frame in 5.5(d) is the probability matrix using which the best policy is achieved (gen 121)

and some of its recent extensions (discussed in Section 5.5). The experiments regarding the original GA framework have been repeated here (instead of using the results reported elsewhere) and the train-test conditions (seeding etc.) for both GA and GA+TA are the same.

5.7.2 Basic Frames: An Analysis

As discussed in Section 5.6, the GA+TA algorithm frequently constructs and factorizes a tensor of high quality candidate solutions. The factorisation process results in the basic frame which is used as a mutation probability. This section is dedicated to the analysis of these basic frames and the manner with which they evolve along side the main cycle of evolution.

Figure 5.5 illustrates the gradual change in probabilities produced by tensor analysis throughout the generations. The instance generator on which the policies were trained in Figure 5.5 is $UBP(30, 4, 25)$. The basic frame generated after the first factorisation (Figure 5.5(a)) is notably less detailed compared to the ones generated in later generations. However, during the time, a common pattern seems to emerge. This pattern reveals that, for this UBP, good quality matrices tend to frequently change the score values corresponding to small item sizes and large remaining bin capacities. Thus, subjecting these locus to mutation more frequently would probably result in better packing performance.

Different UBP's indicate different patterns though. For instance, Figure 5.6 shows one of the basic frames produced for the instance generator $UBP(40, 10, 20)$. Using this basic frame, the best policy matrix was found during training. The pattern here is certainly different from those in Figure 5.5 indicating a whole different group of items sizes and remaining bin capacities as the most frequently changing genes. It also is less focused and more disconnected compared to the basic frames in Figure 5.5(d).

A closer look at Figure 5.5 shows another interesting aspect of the generated basic frame. It seems that in addition to finding common changing locus in the chromosome, the basic frame also identifies groups of different genes with similar (if not equal) probabilities. In other words, basic frames seems to partition genes into groups (with no clear border) where genes within each group are related. This is no surprise and it is one of the achievements of the ALS algorithm. In Eq.5.5, the factor **a** captures the gene patterns corresponding to bin remaining capacity while **b** does the same for gene patterns concerning the item size. The factor **c** captures the temporal profile of the patterns in the first two factors. Hence, our approach is able to detect recurring gene patterns along each dimension (remaining capacity and item size). Moreover, when constructing the tensor, only *good* quality solutions were allowed in the tensor. Thus, any pattern detected along each dimension is equally promising. The basic frame is calculated from the outer product of **a** and **b**, combining the gene patterns related to each dimension of the tensor. It has been observed in many studies (such as [250] and [153]) that the basic frame quantifies the relationship between the elements of the two factors. Hence, the relationship between any gene pattern detected along the first and the second dimensions is scored in the basic frame. Thus, if there are regions with similar score values [2] in the basic frame (as it is visible in both figures 5.5 and 5.6), the genes are considered to be *related*.

[2]Not to be confused with the scores in the policy matrix. The score here refers to the quantity achieved from the factorisation procedure.

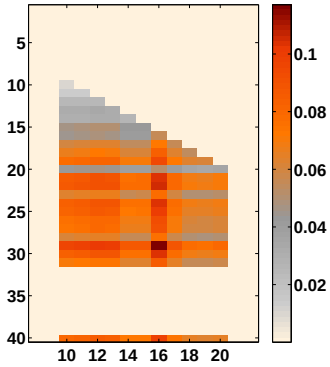


FIGURE 5.6: The basic frame for $UBP(40, 10, 20)$ using which the best policy matrix was found.

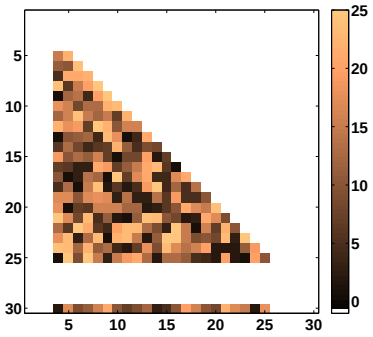


FIGURE 5.7: The best policy matrix obtained by GA+TA for $UBP(30, 4, 25)$ using the basic frame entries (see Figure 5.5(d)) as mutation probabilities.

It is important to stress the fact that the produced basic frames are in no way representing the index scores generated by the GA framework. That is, we are not trying to infer score values in the policy matrix from the corresponding elements of a basic frame. The policy matrix in Figure 5.7 is generated using the probabilities in Figure 5.5(d) and solves instances generated by $UBP(30, 4, 25)$ instance generator. It is evident from the figures that although the two matrices are similar in dimensions, they are not similar at all when it comes to the contents. The rough structure of the policy matrix itself compared to the smooth structure of the basic frame confirms that there is little correlation between scores and mutation probabilities.

5.7.3 Comparative Study

The experimental results show that our algorithm (GA+TA) outperforms the original GA framework on almost all instances significantly. A Wilcoxon sign rank test is performed to confirm this. Table 5.3 summarizes the results. The only instance generator on

TABLE 5.3: Performance comparison of the GA+TA, GA, the generalized policy achieved by the AL method, BF and harmonic algorithms for each UBP over 100 trials. The 'vs' column in middle highlights the results of the Wilcoxon sign rank test where $>$ ($<$) means that GA+TA is significantly better (worse) than the compared method to the method in the left and right column within a confidence interval of 95%. Similarly, $\geq$ shows that GA+TA performs slightly better than the compared method (with no statistical significance). The sign = refers to equal performance.

Instance	**GA** [29]	**vs**	**GA+TA**	**vs**	**AL** [45]	**BF**	**Harmonic**
UBP$(6,2,3)$	99.99	$=$	99.99	-	-	92.29	-
UBP$(15,5,10)$	99.58	$\leq$	**99.62**	-	-	99.62	74.24
UBP$(20,5,10)$	97.89	$<$	**98.28**	$>$	94.32	91.55	90.04
UBP$(30,4,20)$	99.09	$<$	**99.53**	-	-	96.84	73.82
UBP$(30,4,25)$	98.39	$<$	**99.53**	$>$	97.69	98.38	74.21
UBP$(40,10,20)$	96.08	$<$	**96.27**	-	-	90.23	89.10
UBP$(60,15,25)$	**99.68**	$>$	99.47	$>$	93.83	92.55	85.18
UBP$(75,10,50)$	98.27	$<$	**98.53**	$\geq$	98.50	96.08	71.59
UBP$(80,10,50)$	98.07	$<$	**98.66**	$>$	98.17	96.39	72.96
UBP$(150,20,100)$	97.78	$<$	98.22	$\leq$	**98.32**	95.82	71.97

which GA+TA seems to be under-performing is UBP$(60,15,25)$. On all other instances, GA+TA outperforms the GA framework.

Our studies show that it is very hard to increase the performance of the GA algorithm even slightly. Nevertheless, the GA+TA algorithm has improved the performance substantially. One major reason that contributes to the success of the GA+TA algorithm is the representation. Tensor factorisation algorithms are designed for high dimensional data where it is expected that various dimensions of data are correlated. Perhaps the study in [153] is a very good example confirming this argument. Thus, matrix representation of packing policies prepares a suitable ground for analytic algorithms with an expectation of existing relations between various dimensions of data. The fact that the first dimension of a policy matrix is dedicated to remaining bin capacities and the second to item sizes fits very well to the factorisation algorithm. This is something which couldn't be achieved if the policies were vectorized (as in [45]). Therefore, the representation matters and it has a great contribution to the performance of GA+TA. However, apart from representation, the strength of tensor analytic approaches has also a great impact on the performance. In previous study [37] introduced the use of these approaches in heuristic research for the first time. The impressive results achieved in this chapter confirms this and is encouragement for further research in transferring tensor analytic approaches to the field of (meta)heuristic optimisation.

5.8 Summary

An advanced machine learning technique (tensor analysis) is integrated into a GA framework [29] for solving an online bin packing problem. Online bin packing policies with matrix representation enables construction of a 3^{rd}-order tensor as the high quality candidate solutions vary from one generation to another under the genetic operators in GA. This construction process is repeated periodically throughout the evolutionary process. At the end of each period, the obtained tensor is factorized into its basic factors. Then those basic factors are used to identify recurring gene patterns identifying the frequency with which genes are modified in high quality solutions. This information is directly used to set the mutation probability for each gene, accordingly. Our empirical experimental results show that the proposed tensor analysis approach is capable of adaptation at the gene level during the evolutionary process yielding a successful locus-based mutation operator. Furthermore, the results indicate that tensor analysis embedded into GA significantly improves the performance of the generic GA with standard mutation on almost all online bin packing instance classes used during the experiments. Since the data provided to the factorisation procedure is less abstract here (compared to those in Sections 3 and 4), we conclude that the tensor analysis approach is capable of handling data with lower abstraction levels. Finally, due to the multi-episode nature of the tensor learning in this chapter (as well as in the previous chapter), we can confirm that tensor analysis is capable of continuous pattern recognition in heuristic search algorithms.

Solution representation can perhaps be considered as data with lowest level of abstraction. So far in this study, the tensor frames consist of highly abstract data. In the next chapter, we will push the proposed approach to the limits by embedding it in an agent-based metaheuristic approach. The tensor frames will be the candidate solutions for the Flowshop Scheduling problem instances. Therefore, we subject our algorithm to data with the lowest possible level of abstraction. By performing the experiments in the next chapter we will have our algorithm tested on every possible range of data abstraction and heuristic design philosophy.

Chapter 6

A Tensor Approach for Agent Based Flow Shop Scheduling

In this chapter, the proposed approach is applied to the permutation flow shop scheduling problem under an agent-based framework. Multiple agents run in parallel each applying it's own (meta)heuristic to a given problem instance. Time to time, one of the agents initiates a line of communication to the other agents asking for their best solutions. These solutions are appended to each other to form a third order tensor. The constructed tensor thus contains candidate solutions retrieved from the problem domain implementation and is considered to have a very low level of abstraction. The tensor is factorized and the emerging pattern is sent back to all the agents where each agent uses the pattern to construct a better solution.

6.1 Introduction

In this chapter, we use tensors for online learning in a *multi-agent* system to solve the permutation flow shop scheduling problem (PFSP). A multi-agent system provides means for cooperative search, in which (meta)heuristics are executed in parallel as agents with ability to share information at various points throughout the search process. The interest into cooperative search has been rising, considering that, nowadays, even home desktop computers have multiple processors enabling relevant technologies. For example agent-based approaches have been proposed to enable computer dead time be utilised to solve complex problems or used in grid computing environments [251, 252].

Cooperating metaheuristic systems have been proposed in various forms by[253–256]. Several frameworks have been proposed recently, incorporating meta-heuristics, as in

[255, 257, 258], or hyper-heuristics, as in Ouelhadj and Petrovic [259]. Also, Hachemi et al. [260] explore a general agent-based framework for solution integration where distributed systems use different heuristics to decompose and then solve a problem. Other studies have focussed on swarm intelligence, such as, Aydin [261] and Khouadjia et al. [262].

In our multi-agent system based on the framework provided by Martin et al. [263, 264] for PFSP, each agent instantiates the well known NEH algorithm [4] and performs a search starting from a different point in the search space. During this process, each agent builds its own tensor from the incumbent solutions, which is then shared with the other agents. One agent then concatenates all the tensors received from the other agents and factorises it. A solution to a permutation flow shop scheduling problem is a tour formed of edges on a graph where each node is visited once. The resultant factor matrix is used to form a list of edges identified as likely to be members of an overall good solution to the problem in hand. Each agent then uses these edges to build new improved incumbent solutions. The process repeats until a stopping criteria is met and an overall best solution is found.

We tested this approach on the benchmarks of Taillard [265] and Vallada et al. [3], performing better than standard heuristics and delivering a competitive performance to the state-of-the-art. To the best of authors' knowledge, this is the first time a tensor approach is used employing domain knowledge in heuristic optimisation as well as this being its first application in the context of agent-based distributed search.

This chapter is organised as follows. Section 6.2 overviews the permutation flow shop scheduling problem. Section 6.3 describes the proposed multi-agent system which embeds tensor analysis for solving the permutation flow shop scheduling problem. Section 6.4 discusses the details of the experiments and results. Finally, Section 6.5 provides our conclusions.

6.2 Permutation flow shop scheduling problem (PFSP)

In this section, we provide a formal description of the permutation flow shop scheduling problem (PFSP) and an overview of some recent studies on PFSP. Given a set of n jobs, $J = \{1, ..., n\}$, available at a given time 0, and each to be processed on each of a set of m machines in the same order, $M = \{1, ..., m\}$. A job $j \in J$ requires a fixed but job-specific non-negative processing time $p_{j,i}$ on each machine $i \in M$. The objective of the PFSP is to minimise the *makespan*. That is, to minimise the completion time of the last job on the last machine C_{max} [266]. A *feasible schedule* is hence uniquely represented by a

permutation of the jobs. There are $n!$ possible permutations to search from for a given instance and PFSP is NP-hard [267].

A solution can hence be represented, uniquely, by a permutation $S = (\sigma_1, ..., \sigma_j, ...\sigma_n)$, where $\sigma_j \in J$ indicates the job in the j^{th} position. The completion time $C_{\sigma_j,i}$ of job σ_j on machine i can be calculated using the following formulae:

$$C_{\sigma_1,1} = p_{\sigma_1,1} \tag{6.1}$$

$$C_{\sigma_1,i} = C_{\sigma_1,i-1} + p_{\sigma_1,i}, \text{ where } i = 2, ..., m \tag{6.2}$$

$$C_{\sigma_j,i} = max(C_{\sigma_j,i-1}, C_{\sigma_{j-1},i}) + p_{\sigma_j,i},$$
$$\text{where } i = 2, ..., m, \text{ and } j = 2, ..., n \tag{6.3}$$

$$C_{max} = C_{\sigma_n,m} \tag{6.4}$$

PFSP has received considerable attention by researchers with numerous papers being published since the introduction of the problem in mid fifties [268]. As such, in this brief review, we will concentrate on recent work. The readers can refer to the survey papers of [269–271] for an overview of the developments in the area and more.

One of the well known algorithm in the field of PFSP is the deterministic constructive heuristic of Nawaz et al. [4] often known simply as "NEH". The algorithm comprises of two basic stages. In stage one an initial order of jobs is created with respect to an indicator value. While in stage two, a new solution is constructed iteratively by inserting jobs into a partial sequence according to the ordering of jobs in stage one until a new unique sequence of jobs is created. In more detail there are 3 steps to the algorithm:

1. Make a list of jobs in decreasing order based on the sums of their processing times on all the machines.
2. Take the first two jobs in the list and schedule in order to minimise the makespan of these two jobs as though this was the complete list.
3. For k jobs from $k = 3$ to n insert the kth job at the place in the schedule that minimises the partial makespan among the k possible ones.

Taillard [272] introduced a significant improvements to the basic NEH algorithm making it faster and more efficient. It is often referred to as "NEHT" and is the version most commonly used in subsequent studies since it was first proposed. NEHT improves the complexity of all insertions in NEH by using a series of matrix calculations. NEHT seems to be the best-known polynomial time heuristic for flow shop scheduling problems. This

is a simple but powerful deterministic heuristic that was until recently one of the best-performing algorithms for PFSP [269]. Indeed, interestingly, those algorithms that now outperform NEHT still make use of many important features of this heuristic. Most of these new improving algorithms either try to change the way the initial jobs list of stage one is calculated and ordered, and/or, in stage two, change the criterion for choose jobs from the list.

One of the most important studies of the past few years was proposed by Ruiz and Stützle [7].They proposed an iterated greedy (IG) search algorithm that generates a sequence of solutions by iterating over greedy constructive heuristics. They developed a two phased algorithm that iteratively removes jobs from an incumbent solution in the first destruction phase and in the second construction phase reinserts them into the solution using the famous NEH construction heuristic [272].

In the destruction phase $d < n$ jobs are removed at random from an incumbent solution. This creates two partial lists one with the jobs removed and the list of removed jobs. Both lists retain their order with respect to the way the jobs were removed. In the construction phase the NEHT construction heuristic is used to re-insert the removed jobs into the remaining jobs list to create a new potential solution. Once the new solution has been constructed a local search heuristic based on the insertion neighbourhood heuristic of Osman and Potts [273] is used to further improve the solution. The acceptance criterion uses a Simulated Annealing like diversification strategy to make sure the algorithm does not get stuck in a local minimum. When a new potential solution is found that improves on the previous incumbent the new solution replaces the old one and the search repeats until the stopping criterion is reached.

Dong et al. [5] introduced improvement to both the initial and the construction stages of the NEHT heuristic and their heuristic is referred to as NEHD. In the first stage, rather than building the tardy job list as described for NEH, NEHD finds the average and standard deviation of processing times of jobs on each machine. The list is constructed in decreasing order based on these measures. NEHD also modifies the second stage of NEHT by developing a strategy for when there is more than one improving solution obtained by the construction technique of NEHT. Such ties are resolved by finding the solution that is most likely to increase the utilisation of each machine.

Zobolas et. al [2] introduced a hybrid approach. A constructive initialisation method based on a greedy randomised NEH is used to produce the initial population. This is then improved using a Genetic (memetic) Algorithm (GA) employing a variable neighbourhood search algorithm for intensification. The proposed approach also uses a restart technique where old solutions in the population are replaced with the solutions produced by the greedy randomised NEH algorithm.

Fernandez and Framinan [274] described a method for resolving the high number of ties between jobs when iteratively constructing a new schedule. This approach greatly improves the overall performance of the proposed algorithm. Juan at al. [275, 276] also made use of NEHT, creating a randomised variant of NEHT which chooses jobs for the ordered job list according to a biased random function.

A number of different types of metaheuristics have been proposed to tackle PFSP. Single point based search methods include Simulated Annealing(SA) [277], Tabu Search(TS)[278] and various hybrid meta-heuristics (e.g., [279]). A number of population based meta-heuristics have also been investigated recently, including Particle Swarm Optimisation (PSO) [280], evolutionary algorithms [281, 282], and Ant Colony Optimisation (ACO) [283]. Chen et al. [284] recently proposed a new search technique which uses NEHT to provide a good first best solution. They then analyse the search trajectory contained in that first solution to identify potential good and bad areas for further search. A construction heuristic is deployed to generate a population based on these trajectories which is further improved by filtering. If an improving solution is found the local best solution is updated. If there is no improvement a jump diversification strategy is applied.

In this chapter, we will show that collections of cooperating agents each executing a modified version of NEHT coupled with an online tensor-based learning mechanism can produce solutions that are, at least good, and in some cases better, than the current state-of-the-art.

6.3 Proposed Approach

Martin et al. [263, 264] proposed a multi-agent system where each agent is autonomous and communicates asynchronously and directly with each other. The system also features a learning mechanism where the agents share improving partial solutions. Each agent then uses a construction heuristic to generate new incumbent solutions from these partial solutions. The authors claim that their system is generic in that it can solve problems from different domains such as nurse rostering, permutation flow shop scheduling and vehicle routing. They claim further that cooperating agents perform better on the same problem than non-cooperating agents and that search results will improve if more agents are used. In this chapter, we propose a tensor learning system that identifies good partial solutions instead of the previous frequency based method [263] and focus on a single domain namely permutation flow shop scheduling problem (PFSP).

6.3.1 Cooperative search

The multi-agent system used in this study was described in great detail in Martin et al. [263]. Consequently, we will provide a general description of how the system works and will only go into greater detail where the two systems diverge.

The platform itself is built using the FIPA compliant [285] Java Agent Development Environment(JADE) [286]. JADE is both a set of libraries for developing agents as well as providing an agent execution environment.

The platform contains two types of agents, a *launcher agent* and a *metaheuristic agent*. In any search, there is one launcher agent while there will be many metaheuristic agents operating cooperatively. The launcher's job is to read in a problem, parse it and send it to the metaheuristic agents for solving. When the metaheuristic agents have found a solution to a problem they will communicate the answer back to the launcher which then parses the result into a human readable format. The launcher also controls the number of times a problem instance will be solved.

The metaheuristic agents solve a given problem by cooperating with each other. This is achieved by using a selection of FIPA complaint interaction protocols [287]. These protocols represent many types of human communication behaviour such as asking someone for something or, telling someone about something. The agent-based system uses these communication protocols to allow the agents to cooperate with each other in order to solve complex optimisation problems.

Martin et al. [263] developed what amounts to a distributed metaheuristic with a pattern matching learning mechanism that all the metaheuristic agents participate in to solve a problem. One iteration of this distributed metaheuristic is called a "conversation". During a conversation each of the agents can take on one of two roles where one agent takes on the role of initiator while the others take on the role of responders. These roles will change with each new conversation. In their chapter, Martin et al. [263] describe how they used a pattern matching system based on frequency of recurring pairs of elements in good solutions as their learning mechanism. These pairs are used by the metaheuristics instantiated by each agent to construct new incumbent solutions. However, here we use tensor online learning to identify good patterns rather than frequency of pairs of elements. Each conversation executes the following steps:

1. Each agent executes a modified version of NEHT (see Section 6.3.2);
2. Each agent saves up to the 20 best potential solutions and their make-span values from the NEHT execution phase;

3. Each agent converts these potential solutions in upto 20 lists of edges (pairs of solution elements) based on the order of each solution;
4. The responders each send their 20 best solutions to the initiator;
5. The initiator creates a tensor of size $P \times Q \times R$. The first two dimensions of the tensor are of size n, the number of jobs. The length of the tensor R is then $20 \times a$, where a is the number of agents;
6. The initiator factorises the tensor with a call to Matlab;
7. The initiator takes the basic frame, the result of the factorisation, and converts into a list of good edges.
8. The initiator shares these good edges with the responders;
9. The initiator and responders use these edges to modify the job list used by NEHT;
10. The conversation repeats a given number of times (10 times in this study).

6.3.2 Metaheuristic agents

The (meta)heuristic executed by each agent is a modified version of randomised version of NEHT as proposed by Juan et al. [275]. Essentially they introduced a biased random function for choosing jobs from the tardy jobs list described in step one of the NEH algorithm.

In our version, we do not use the biased random function and as such we use the standard deterministic NEHT. However, we modify the tardy jobs list itself by taking good edges that have been identified by the agents in step 7 of a conversation. We take the list of good edges convert it into unique a ordered list of jobs maintaining the inherent order of the edges list. This list is compared with the tardy job list. The tardy list is then reordered so that the jobs in the good edge list are moved to the head of the tardy jobs list. This then influences the way NEHT constructs a new incumbent solution favouring our identified jobs. The reason for doing this is that good jobs as identified by the tensor learning will tend to be jobs that feature in good solutions. By putting them at the head of the list we favour them when a new incumbent solution is constructed.

6.3.3 Construction of tensors and tensor learning on PFSP

The tensor learning method is an reinforcement scheme. It is implemented in Matlab. Once the initiator has created the tensor for that conversation, it is written to file. The

Matlab executable is then called from the JAVA code and it reads in the tensor from file and executes the tensor factorisation algorithm. The resultant matrix called a basic frame is written to file, whereupon the JAVA code reads it in and executes the rest of the search algorithm. In the first conversation, the initiator receives all the best 20 solutions as lists of edges from each agent in the order they are received. It then creates a tensor of size $P \times Q \times R$, where $P, Q = n$ (number of jobs) and $R = 20 \times a$ (number of agents). This is achieved by creating an $n \times n$ adjacency matrix where 1 to n rows and columns represent the ID number of each job. The value 1 is then entered if for a given edge (x, y) x represents the row number while y is the column number. Zero is entered otherwise. In this way, a sparse tensor is created where each slice, called a frame, represents a single solution generated by the agents. Also each frame has a label associated with it which is the makespan of that solution. Some filtering takes place at this stage, the values of frame labels are averaged, those frames with labels higher than average are discarded. This is because in PFSP we are minimising the makespan. This procedure results in a tensor $\mathcal{T}$. A copy of this tensor is stored after the first conversation which we will denote as $\mathcal{R}$.

In each subsequent conversation, once a tensor has been generated by the initiator, the tensor $\mathcal{R}$ is appended to the fully generated tensor. The worst half of this final tensor is then discarded, resulting a new tensor $\mathcal{T}$. Once again, the better half of the tensor $\mathcal{T}$ is stored as $\mathcal{R}$. This cycle repeats until the maximum number of conversations is reached. By this procedure, the current tensor $\mathcal{T}$ is reinforced by the contents of the tensor $\mathcal{R}$ and represents the best improvement so far. In this way, good solutions are rewarded and preserved for later conversations.

Finally, the tensor is factorised by the Matlab code executing CP decomposition and the results are written into a matrix called the basic frame. This matrix is then read and treated as an adjacency matrix by the Java code. It is converted to a list of pairs or edges of a graph. This is achieved by taking the row and columns numbers and making a list of pairs according to the score values in the basic frame. These edges are put into order according to their basic frame score which represents their likelihood to be an element of a good solution. The initiator shares the best 10% of good edges with the other agents. Furthermore, it is this list that agents use to modify the NEHT tardy jobs list.

6.4 Computational Results

We will refer to the proposed multi-agent system embedding the tensor based online learning technique as TB-MACS, and MACS [263] for the previous version of the system

using a pattern matching mechanism instead. In this section, we provide the experimental results of applying our approach to the permutation flow shop scheduling problem instances from the benchmarks of Taillard [265] and Vallada et al. [3]. Throughout this section, we will follow the convention in permutation flow shop scheduling of describing benchmark instances by the number of jobs n followed by the number of machines m. For example 500×20 describes a dataset where each instance in the set is made up of 500 manufacturing jobs and to be executed on 20 flow shop machines.

In the first part of this section, we discuss the parameter tuning experiments for tensor learning and then provide a performance comparison of TB-MACS to MACS on the Taillard instances under different number agent settings. Next we compare the performance of TB-MACS to MACS based on 16 agents using the benchmark instances of Vallada et al. [3]. Finally, the results from the proposed approach is compared to the previously proposed approaches. The experiments are all run on identical computers (Intel i7 Windows 7 machine -3.6 GHz- with 16 GB RAM). We have used the same settings for the multi-agent system as in Martin et al. [263]. Each agent executes for twelve CPU seconds in parallel before each conversation which occurs ten times.

We have used the relative percentage deviation (RPD), also referred to as %-gap as a performance indicator in our comparisons:

$$RPD = \frac{makespan(Method_{sol}) - BKS}{BKS} \cdot 100,$$

where $makespan(Method_{sol})$ refers to the makespan of the solution produced by our multi-agent system or by the state-of-the-art metaheuristics used in comparison with our system. BKS refers to the makespan of the best known solution or upper bound published by Taillard [265] and Vallada et al. [3]. We have performed the Wilcoxon signed rank test to evaluate the average performance of pair of given algorithms based on the RPD values obtained from multiple runs.

6.4.1 Parameter configuration of tensor learning

The value for the variable K in Eq. 2.7 (number of desired components) should be provided to the factorisation procedure. In some studies, when tensor dimensions follow a specific structure, the value for K can be estimated [171]. Some other studies use informed guesses to decide on the number of desired components [153]. This is possible when the data under consideration provides means to infer the number of components. For instance, consider a video dataset containing recorded human actions. Assume that we are interested in recognising actions related to the head and the torso of the person

in the video. Furthermore, given that there are only 3 possible positions for the head and 2 for the torso, one can easily infer that $K = 5$ [171]. Some other applications [168], use arbitrarily large number of components and use a proportion of basic factors. That is, they discard all the basic frames with a weight (λ in Eq. 2.7) lower than a selected threshold and reconstruct the tensor to produce noise free data.

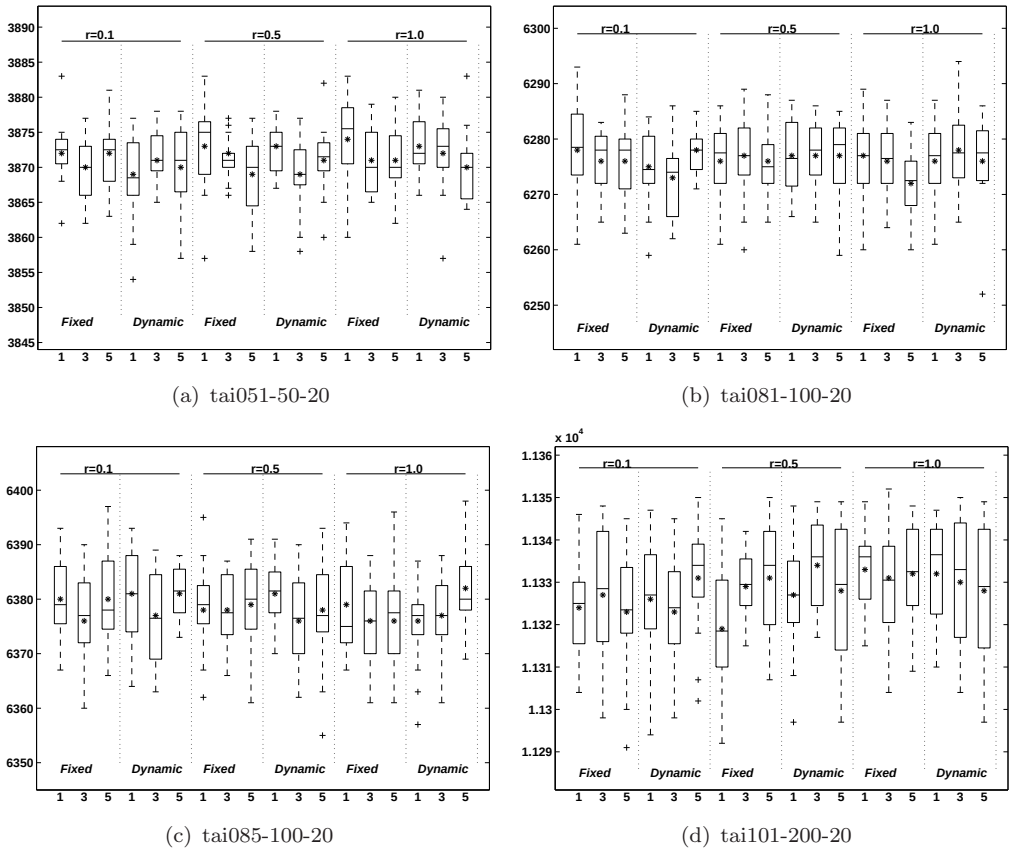
The data we are dealing with, in this chapter, matches none of the applications above. Therefore, we need to determine this value experimentally. Furthermore, in cases where $K > 1$, a strategy should be devised to decide which component to use and send to other agents. In our approach, after the tensor factorisation, the basic frames are generated in the descending order of their weights (λ in Eq. 2.7). That is, it is guaranteed that the first component has the maximum weight among all other components. The second component has the second largest weight and so on. Thus, a simple strategy would be to rely on the weight values and select the first component (which has the maximum weight). Another strategy would be to consider the trend of a basic frame. The trend of a given basic frame k is the vector $\mathbf{c}_k$ in Eq.2.7. Assuming that basic frames contain good patterns and we only have to choose the best one, it makes sense to choose the basic frame with a dominant trend. To clarify this further, assume that $K = 2$ in Eq. 2.7 leaving us with two components. Hence we will have two basic frames after factorisation where each one has a trend vector denoted by $\mathbf{c}_1$ and $\mathbf{c}_2$ respectively. The length of both vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ are equal. If, for the majority of points i, $c_{1i} > c_{2i}$, then we will consider $\mathbf{c}_1$ to be the dominant trend. Please note that, a dominant trend does not mean better makespan values in candidate solutions represented in the tensor. Rather, it simply tells us how much the basic frame has been present in the original tensor. Naturally, given our assumption that basic frames contain good patterns, the basic frame with dominant trend could be a good choice since it dominates the original tensor more than other frames and is thus a more reliable choice. The assumption that basic frames contain good patterns is also a reasonable assumption given that the original tensor contains the better half of the combination of the solutions obtained from all the agents and the reinforcement frames (described in Section 6.3.3).

In addition to the value for K and the selection strategy for basic frames, there is one more decision to be made (experimentally). We should decide how much of the basic frame should be used. The basic frame represents the outline of a good solution. Every edge in this solution outline is associated with a real valued score and we use these scores to generate a ranked list of all the edges. For small instance, ranking can be performed using all these scores because there are not many of them. In large instances however, given the small fraction of time provided to the agents to evaluate these scores, using all the scores may not be a wise decision. Top scores could be selected, based

on a threshold, and these scores could be utilised more effectively. We determine this threshold (denoted by r) experimentally.

We have set up an experiment to decide about the above mentioned strategies and parameter values. For the variable K three values have been considered: $\{1, 3, 5\}$. To decide on which component to choose (in case $K > 1$) two strategies are tested. The first one, referred to as the *fixed* strategy always chooses the first component. In other words, the fixed strategy always chooses the basic frame with maximum weight (λ). The second strategy referred to as the *dynamic* strategy, chooses the basic frame with the maximum increasing trend. As for the threshold r discussed in the previous paragraph we use three values: $\{0.1, 0.5, 1.0\}$. As an example, when $r = 0.1$, only the top 10% of the scores and their corresponding edges are passed to agents for usage. Four instances from the Talliard benchmark have been chosen for these initial experiments: tai051-50-20, tai081-100-20, tai085-100-20 and tai101-200-20. These instances are specifically chosen so as to ensure that every problem size in the spectrum of all the instances in this benchmark are presented. Furthermore, for each instance, 20 runs of experiments have been conducted for each of the 18 possible configurations. Also, four agents have been used to run these initial experiments as the parameters considered here are independent from the number of agents involved.

The results are illustrated in Figure 6.1. It is hard to draw a conclusion by judging the performance plots due to the slight variation in the average performance of different configurations. The average performance achieved by each configuration varies slightly in most of the cases. There is no single configuration which is better than the others with a statistically significant performance difference. The performance of each configuration across the four instances are ranked based on the tied rank method. The configuration which is the best among others on all the instances considering both mean and minimum makespan is the combination of $K = 1$ and $r = 0.1$. Since $K = 1$ we no longer concern ourselves with determining a strategy to select among basic frames (since there will only be one such frame after factorisation). Interestingly, this is in line with the experiments in Asta and Özcan[37]. The final tensor which is fed into the factorisation procedure consists of the frames (solutions) which have achieved better than mean makespan values, and so outcome of the configuration experiment makes sense. Unlike applications where data has desired and undesired content (which normally yields to $K > 1$), our data consists only of desired solutions. This means the choice of $K = 1$ is sensible.



(a) tai051-50-20

(b) tai081-100-20

(c) tai085-100-20

(d) tai101-200-20

FIGURE 6.1: The box plot of makespan values for each parameter configuration combining different values of r, K and strategy for choosing a component from K. The median and mean values are indicated by a horizontal line and *, respectively, within each box.

6.4.2 Performance comparison of TB-MACS to MACS on the Talliard instances

After achieving the best configuration for the TB-MACS algorithm, we tested it on 12 arbitrarily chosen instances of the Talliard benchmark and compared its performance to MACS, which does not use tensor analysis. The results are summarised in Table 6.1 and 6.2. The proposed tensor based approach outperforms MACS on almost all instances (in terms of average and minimum performance). Moreover, it finds the optimum for the tai-055-50-20 in a run.

In Martin et al [263], the setting with 16 agents is always the winner compared to settings with fewer agents in terms of mean performance based on the RPD values averaged over 20 runs. Unlike [263], here, although the 16 agent framework still wins in the majority of instances, on some instances fewer number of agents perform better. The Wilcoxon signed rank test is conducted to assess whether or not the average results are significantly different from those of the original framework ([263]). The results of this

TABLE 6.1: Mean RPD values achieved by different number of agents (4,8 and 16) by the TB-MACS and MACS approaches on the Taillard benchmark instances over 20 runs and their performance comparison to NEH and NEGAVNS (Zobolas et. al[2]). The best result is marked in bold style. The 'vs' columns highlights the results of the Wilcoxon signed rank test where $>$ ($<$) means that TB-MACS is significantly better (worse) than MACS within a confidence interval of 95% for any given number of agents. Similarly, $\geq$ ($\leq$) shows that TB-MACS performs slightly better (worse) than MACS (with no statistical significance) for any given number of agents.

Number of Agents:				**4**			**8**			**16**		
Instance	**BKS**	**NEH**	**NEGAVNS**	**TB-MACS**	**vs**	**MACS**	**TB-MACS**	**vs**	**MACS**	**TB-MACS**	**vs**	**MACS**
tai051-50-20	3850	6.03%	0.77%	0.53%	$>$	0.84%	0.44%	$>$	0.76%	**0.44**%	$>$	0.63%
tai055-50-20	3610	6.23%	1.03%	0.45%	$>$	0.67%	0.37%	$>$	0.62%	**0.30**%	$>$	0.50%
tai081-100-20	6202	5.47%	1.63%	1.17%	$>$	1.52%	1.17%	$>$	1.41%	**1.14**%	$>$	1.30%
tai085-100-20	6314	6.03%	1.57%	1.08%	$>$	1.34%	**0.94**%	$>$	1.22%	0.95%	$>$	1.11%
tai091-200-10	10862	0.74%	0.24%	0.09%	$\geq$	0.09%	0.09%	$\geq$	0.09%	0.09%	$\geq$	0.09%
tai095-200-10	10524	1.15%	**0.03**%	0.04%	$>$	0.09%	**0.03**%	$\geq$	0.05%	**0.0**%	$\geq$	**0.03**%
tai101-200-20	11195	3.56%	1.34%	1.16%	$>$	1.38%	1.13%	$>$	1.25%	**1.11**%	$>$	1.19%
tai105-200-20	11259	3.78%	1.04%	0.86%	$>$	1.02%	0.86%	$>$	0.94%	**0.83**%	$\geq$	0.88%
tai106-200-20	11176	4.05%	**1.11**%	1.41%	$>$	1.55%	1.37%	$>$	1.44%	1.31%	$>$	1.42%
tai111-500-20	26059	2.34%	**0.73**%	1.13%	$<$	1.01%	1.09%	$<$	0.95%	1.05%	$<$	0.88%
tai115-500-20	26334	1.49%	0.82%	0.77%	$\geq$	1.01%	0.72%	$\geq$	0.95%	**0.70**%	$\geq$	0.88%
tai116-500-20	26477	1.95%	**0.49**%	0.85%	$<$	0.69%	0.79%	$<$	0.67%	0.75%	$<$	0.61%

TABLE 6.2: Best of run RPD values achieved for different number of agents on the Taillard benchmark instances over 20 runs. The lowest value for each instance is marked in bold.

No. of Agents:	**4**		**8**		**16**	
Instance	**TB-MACS**	**MACS**	**TB-MACS**	**MACS**	**TB-MACS**	**MACS**
tai051-50-20	0.23%	0.55%	**0.18**%	0.47%	0.31%	0.44%
tai055-50-20	0.25%	0.50%	0.19%	0.28%	**0.00**%	0.30%
tai081-100-20	**0.87**%	1.26%	0.94%	1.06%	1.02%	1.02%
tai085-100-20	0.89%	1.11%	0.76%	0.97%	**0.68**%	0.89%
tai091-200-10	0.09%	0.09%	0.09%	0.09%	0.09%	0.09%
tai095-200-10	0.03%	0.03%	0.03%	0.03%	0.03%	0.03%
tai101-200-20	0.96%	1.09%	**0.84**%	1.01%	0.98%	0.93%
tai105-200-20	**0.69**%	0.89%	**0.69**%	0.78%	0.73%	0.71%
tai106-200-20	1.26%	1.35%	1.22%	1.27%	**1.16**%	1.33%
tai111-500-20	0.98%	0.88%	0.96%	0.86%	0.92%	**0.69**%
tai115-500-20	**0.60**%	0.88%	0.62%	0.86%	0.64%	0.69%
tai116-500-20	0.65%	0.56%	0.66%	0.60%	0.66%	**0.54**%

test as provided in Table 6.1 show that TB-MACS performs significantly better than the original framework on at least six (out of twelve) Talliard instances, regardless of the number of agents. The performance of TB-MACS is also compared to NEH and the hybrid approach of Zobolas et. al [2], denoted as NEGAVNS. TB-MACS outperforms NEH on all instances, while it delivers a better performance than at least on seven instances using any chosen number of agents in the overall. show that TB-MACS outperforms NEH on all instances.

Considering the instance tai095-200-10, the tensor based approach outperforms MACS, significantly when 4 agents are involved and slightly when 8 or 16 agents are involved. On the instance tai105-200-20, the tensor based approach outperforms the original framework significantly when 4 or 8 agents are used and slightly when 16 agents are used. Apart from these instances, on 3 (out of 12) instances the tensor based approach performs slightly better than the original framework for any given number of agents. In total, the TB-MACS approach outperforms the original framework (either significantly or slightly) on 10 out of 12 instances regardless of the number of agents.

In Figure 6.2, the temporal behaviour of the two algorithms have been compared against each other on the tai-051-50-20 instance. The TB-MACS algorithm using 16 agents improves the solution quality right until the end of the search. While for the same search the MACS algorithm gets stuck early on. A similar behaviour is observed on majority of the problem instances for which TB-MACS performs better.

TB-MACS is outperformed by MACS on two of the larger instances (tai-111-500-20 and tai-116-500-20). This is potentially because the proposed approach, gathers tensor frames consisting of good local optima achieved between the two conversations and looks for useful patterns in these local optima. Similar to the other data mining methods, the performance of the tensor analysis approach depends on the quality and the quantity of the data. In larger instances, achieving high quality data at the beginning of the search

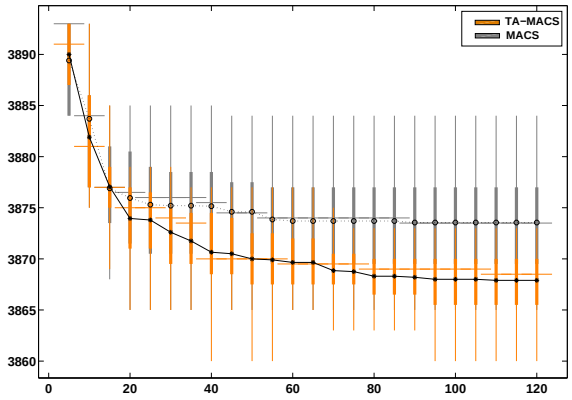
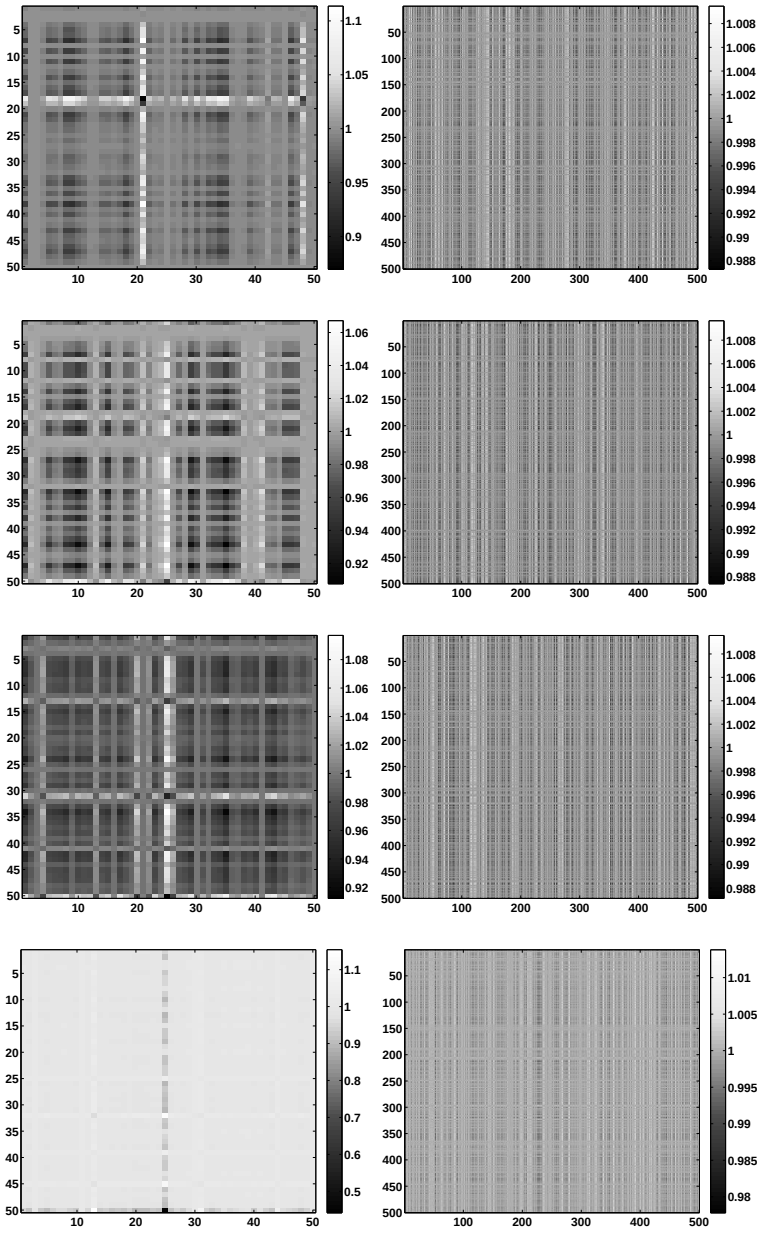


FIGURE 6.2: The progress plot for TB-MACS and MACS using 16 agents while solving tai-051-50-20 from 20 runs. The horizontal axis corresponds to the time (in seconds) spent by an algorithm and the vertical axis shows the makespan.

may not be possible. Therefore, the factorisation method may experience difficulty in producing a good ranking. Moreover, as the size of the instances increases, so should the number of frames in the tensor. However, here, we have fixed the number of frames in the tensor. In other words, the size of the problem increases but the amount of data available to the factorisation method does not change. Thus, in larger instances (tai-111-500-20 and tai-116-500-20), the tensor analysis approach suffers from lack of data in terms of quantity and quality.

Figure 6.3 provides an illustration indicating how the aforementioned reasons affect the pattern extraction procedure. The images in the first column, correspond to basic frames constructed for the instance tai-051-50-20 after conversations 3, 5, 7 and 9 respectively (from left to right) using four agents. The images on the second column are basic frames achieved for the same conversations for the instance tai-116-500-20, again, using four agents. The two instances are respectively the smallest and the largest Talliard instances used in this chapter. They have been chosen deliberately to show the reasons behind the difference in the performance of the TB-MACS on small and large instances. Figure 6.3 shows that basic frames achieved for the small instance tai-051-50-20 throughout the conversations vary quite a lot. This means that for this instance, TB-MACS extracts new and different patterns at each conversation. This is indeed the way it should be, because as the search proceeds, one would expect that new local optima are found and they are likely to have a different solution structure. The existence and discovery of new optima makes each basic frame (pattern) different from the others. However, this is not the case for basic frames constructed for the larger instance (tai-116-500-20). The basic frames remain almost the same throughout the conversations, indicating that little or no new local optima has been detected. More data or a better underlying heuristic



(d) tai-051-50-20 (e) tai-116-500-20

FIGURE 6.3: The illustration of the gradual change in the basic frame (patterns) collected from the agent conversations 3, 5, 7 and 9 on (a) tai-051-50-20 (smallest instance), and (b) tai116-500-20 (largest instance) of the Taillard benchmark.

would change the basic frames leading to gradually changing patterns similar to those achieved for the smaller instances.

6.4.3 Performance comparison of TB-MACS to MACS on the VRF Instances

Following the promising results achieved by the TB-MACS algorithm on Talliard instances, TB-MACS is also tested on the new *hard large VRF* benchmark instances from Vallada et al. [3]. We have performed the experiments using 16 agents in this part. The experiments are performed using each algorithm for once (one replicate) on each instance and RPD is computed for each dataset containing ten instance and then averaged. The results from the experiments are provided in Table 6.3. The performance comparison between TB-MACS and MACS shows that the the TB-MACS method performs better than the MACS algorithm on the majority of instances with an overall mean RPD of 0.78% when compared to 0.95% across all instances. However, similar to the experiments in the previous section, as the size of the instances increases, the performance of the TB-MACS method deteriorates slightly. It appears that it is absolutely crucial to increase the size of the tensor data when the size of the instances increases. Nevertheless, the performance of the TB-MACS algorithm is impressive as it performs better than the MACS algorithm for 16 out of 24 datasets based on the mean RPD values.

They produce a similar result on 600×20 and 800×60. We can conclude from the overall results that the proposed tensor online learning approach is promising and it is indeed capable of improving the overall performance of the multi-agent optimisation system in the overall.

6.4.4 Performance comparison of TB-MACS and MACS to previously proposed methods

The focus of this chapter is to test the viability of using tensors as a machine learning technique directly classifying problem data. However, it is natural to want to know how the PFSP test results compare with the state-of-art algorithms in the field. In this section, we provide an *indirect* performance comparison between TB-MACS, MACS and other algorithms tested by [3] using the hard VRF benchmarks. In Tables 2 and 7 of Vallada et al. [3], they tested their hard VRF benchmarks with the best deterministic and stochastic algorithms in the literature. The deterministic algorithms were: NEH [4] and an improved NEH algorithm by Dong et al. [5], referred to as NEHD. The stochastic algorithms were a hybrid genetic algorithm (HGA) [6] and the iterated greedy algorithm (IG) [7]. The experiments are performed using each algorithm run once on each replicate of each instance (there are 10 replicates per instance, hence 10 runs per instance is performed) and RPD is computed for each dataset and then averaged. We did the same using MACS and TB-MACS and compare the mean RPD values from

TABLE 6.3: Mean RPD achieved for 16 agents on large instances provided in Vallada et al. [3] where only one replicate for each algorithm is run (VRF Hard Large benchmarks). The best average result in each row can be distinguished by the bold font. The 'vs' columns highlight the results of the Wilcoxon signed rank test where $>$ ($<$) means that TB-MACS is significantly better (worse) than MACS within a confidence interval of 95%. Similarly, $\geq$ ($\leq$) shows that TB-MACS performs slightly better (worse) than MACS (with no statistical significance). The performance of TB-MACS and MACS is also compared to the NEH [4], NEHD [5], HGA [6] and IG [7] algorithms.

Dataset	TB-MACS	vs	MACS	NEH	NEHD	HGA	IG
100x20	**0.24**%	$>$	0.51%	5.63%	5.25%	1.09%	0.94%
100x40	**0.33**%	$>$	0.69%	5.44%	5.00%	1.14%	0.85%
100x60	**0.33**%	$>$	0.67%	4.80%	4.51%	0.98%	0.89%
200x20	**0.66**%	$\geq$	0.75%	4.24%	3.66%	1.07%	0.78%
200x40	**0.70**%	$>$	1.07%	4.54%	4.34%	1.03%	1.06%
200x60	**0.74**%	$>$	1.17%	4.61%	4.17%	1.07%	0.91%
300x20	**0.46**%	$>$	0.66%	2.91%	2.38%	0.67%	0.49%
300x40	1.00%	$>$	1.59%	4.06%	3.60%	1.05%	**0.86**%
300x60	1.15%	$>$	1.75%	3.92%	3.84%	1.13%	**0.82**%
400x20	0.43%	$>$	0.84%	2.42%	1.87%	0.54%	**0.39**%
400x40	1.17%	$>$	1.39%	3.55%	3.08%	1.10%	**0.87**%
400x60	1.51%	$\leq$	1.42%	3.70%	3.16%	1.13%	**0.81**%
500x20	0.47%	$>$	0.69%	1.98%	1.62%	0.51%	**0.36**%
500x40	1.05%	$>$	1.17%	3.24%	2.56%	1.05%	**0.77**%
500x60	1.35%	$\leq$	1.32%	3.47%	3.01%	1.12%	**0.95**%
600x20	0.50%	$\leq$	0.50%	1.78%	1.27%	0.40%	**0.28**%
600x40	1.08%	$\leq$	1.07%	3.17%	2.48%	1.11%	**0.85**%
600x60	1.17%	$\leq$	1.09%	2.99%	2.53%	1.09%	**0.77**%
700x20	0.43%	$\geq$	0.49%	1.40%	0.94%	0.40%	**0.30**%
700x40	0.86%	$\geq$	0.91%	2.85%	2.25%	1.02%	**0.77**%
700x60	0.98%	$<$	0.93%	2.89%	2.35%	1.08%	**0.87**%
800x20	0.39%	$\geq$	0.41%	1.32%	0.89%	0.33%	**0.25**%
800x40	0.83%	$\leq$	0.79%	2.61%	2.02%	1.05%	**0.78**%
800x60	0.98%	$\geq$	0.98%	2.74%	2.36%	1.24%	**0.88**%
Average	0.78%		0.95%	3.34%	2.88%	0.93%	**0.73**%

all the algorithms in Table 6.3. We emphasise this is an *indirect* comparison between deterministic and stochastic algorithms, where the stopping conditions for TB-MACS and the other stochastic algorithms differ. There is currently no public Java based software library for tensor factorisation which can be included by TB-MACS. Hence, Matlab is used for tensor factorisation involving file reads and writes during the search process, immensely slowing down the execution of TB-MACS.

Given these caveats, TB-MACS performs well on the smaller instances 100×20 to 300×20 when compared to IG whereupon for the rest of the instances the IG algorithm performs better based on mean RPD as shown in Table 6.3. We explained the potential reason for this in Section 6.4.3 namely, that as the problem size increases so should the size of the tensor. TB-MACS outperforms the NEH and NEHD algorithms producing better results for all VRF instances. TB-MACS also performs better than HGA on sixteen (out of twenty four) datasets. They have a similar performance for 500×40. In the overall, TB-MACS achieves an average RPD value of 0.78% and delivers a competitive

performance when compared to IG which yields an average RPD value of 0.73% over 240 VRF instances.

6.5 Summary

In this chapter, we introduced a multi-agent system which embeds an online learning technique based on tensor analysis, applied to permutation flow shop scheduling. Each agent in the distributed system uses the NEHT heuristic to perform the search. The search trace of each agent is presented as a $3^{rd}-$order tensor. Periodically, the tensors constructed by each agent are appended to each other and factorised. Useful patterns are extracted in form of graphs which outline the approximated structure of a good solution. This pattern is then shared between agents where it is used by each agent to guide the search towards better local optima.

The use of tensor analysis in heuristic optimisation was first discussed in Chapter 3 where it was applied to data collected from the trace of a hyper-heuristic. In subsequent chapters, the proposed tensor-based learning was embedded in various hyper-heuristics. These hyper-heuristics did not have access to solution representation and their trace data is considered as highly abstract. This chapter however, focuses on the opposite end of the spectrum of data abstraction levels. Unlike hyper-heuristics where search is performed in the space of heuristics, the heuristics in this chapter search in the space of candidate solutions and thus leave a trace which is richer in details and hence is less abstract. In this chapter, we have shown that the tensor analysis approach can use such data (with low abstraction level) to improve the underlying heuristic significantly on a variety of benchmark problem instances. Together with the results achieved in Chapters 3, 4 and 5, a clear conclusion is that, the tensor analysis approach can handle data with various levels of abstraction and is thus applicable to a wide range of search algorithms like metaheuristics and hyper-heuristics. Furthermore, the results in this chapter are testimony that tensor analysis approach can be embedded in distributed, multi-agent search algorithm and extract useful patterns. Finally, we have observed that curse of dimensionality is an issues as, having a fixed length for the tensor, the performance of the tensor-based learning mechanism deteriorates as the size of the problem instances increases.

Combining the impressive results achieved in this chapter as well as in previous chapters, it is now possible to confirm that the tensor analysis approach is capable of analysing trace data with various levels of abstraction. It is also possible to confirm that the

tensor analysis can be embedded in search algorithms with very different heuristic design philosophies. In the next chapter, more detail is given regarding these concluding remarks. Also, an outline of the future work will be provided.

Chapter 7

Conclusion

7.1 Summary of Work

Search methodologies (i.e., heuristics) are at the core of almost all decision support systems, particularly while dealing with combinatorial optimisation problems. The state-of-the-art systems are often tailored for a particular problem by the experts in the area. Such systems are generally very costly to build and maintain. Since they are custom-made, it is almost impossible to apply/reuse them to/in another problem domain. Even a slight change in the problem definition could require an expert intervention. Whenever exact methods fail, researchers and practitioners resort to heuristics which are 'rule of thumb' methods for solving a given problem. Designing intelligent heuristics with higher levels of generality has been the most important goal in heuristic optimisation community. Several design philosophies (such as hyper-heuristics and automated tuning of meta-heuristics) has been pursued over the years to achieve this goal.

In recent years, there has been a growing interest in employing machine learning methods to improve the performance of heuristic optimisation algorithms. Machine learning can be used to mine the data provided by a given heuristic to discover patterns. These patterns can be used as additional knowledge during the decision process/search. The data provided by heuristics differs when the underlying algorithm changes. That is, depending on the design philosophy, each heuristic leaves a trace which consists of different descriptive features and can have different levels of abstraction. A hyper-heuristic which operates using the domain barrier often produces data with high levels of abstraction. The features produced in such data are often simple and patterns easy to extract. However, the predictive power of such patterns can sometimes be low. On the other hand, a (meta)heuristic which directly searches in the space of candidate solutions leaves a trace

which is rich in details and patterns are very hard to extract. However, such patterns are usually very descriptive and can have great predictive powers.

Tensor analysis is a very powerful data mining technique which has been widely employed in challenging problems such as natural language processing and face recognition. Instead of collapsing data to two dimensional datasets (as is usually the case in machine learning), in tensor analysis, the natural dimensionality of the data is preserved and represented as a tensor. Doing this protects the correlations between data variables which would otherwise be lost if the data were to be collapsed. Tensor factorisation is used to decompose these tensors to their basic factors. The pattern which describes the correlation between various modes of data can then be extracted from these factors.

In this study, the trace of a given (hyper-/meta) heuristic is represented as a $3^{rd}-$order tensor. After applying tensor factorisation to the collected data, basic frames are extracted. The basic frames are then used to guide the underlying heuristic optimisation approach throughout the search process with the goal of improving its overall performance.

The proposed approach has been applied on a wide range of problems using a variety of underlying (hyper-/meta) heuristics. Since learning can be done in single or multiple episodes, both learning approaches have been considered in this study. Furthermore, we have tested the flexibility of our method on data with various levels of abstraction.

7.2 Discussion and Remarks

Our experimental results show that, regardless of the design philosophy used for a given heuristic, the proposed approach can improve its performance significantly. We have tested our approach to mine the tensorial trace of a very simple hyper-heuristic in a single episode fashion (Chapter 3). We have used the extracted patterns and created a multi-stage hyper-heuristic which also uses very primitive components. Our experimental results show that the tensor-based hyper-heuristic improves the performance of the underlying algorithm significantly and achieves extraordinary cross-domain performance on the HyFlex benchmark. Furthermore, to analyse the multi-episode behaviour of the learning phase and to assess whether or not the proposed approach is capable of extracting useful patterns in longer run times, we have applied an extended version of the tensor-based approach to real-world instances of the Nurse Rostering Problem (Chapter 4). Our experiments show that the proposed approach performs very well and is capable of producing new best known solutions for several instances of the benchmark.

In other words, the tensor-based approach performs well in multi-episode training and is capable of extracting useful patterns continuously.

We also have applied the tensor approach to a GA-based hyper-heuristic (referred to as the CHAMP framework) which generated policies to solve one dimensional online bin packing problems (Chapter 5). Periodically, the trace of the GA algorithm is represented as a $3^{rd}-$order tensor and factorized to a basic factor. The contents of the basic factor is then used as an adaptation strategy which operates at the gene level during the evolutionary process and yields locus-based mutation probabilities. These probabilities are used as mutation probabilities for different loci of each individual. The tensor-based approach has been employed as a multiple episode approach where the learning of mutation probabilities occurs periodically. The experiments show that the tensor-based approach improves the performance of the GA framework significantly on almost all of the instances. The tensorial data provided by the GA is less abstract compared to the data provided by hyper-heuristics. In the latter case, the hyper-heuristic does not have access to the design of low level heuristics. As opposed to hyper-heuristics in Chapters 3 and 4, the CHAMP framework evolves and generates heuristics. Thus, the tensor analysis component has direct access to heuristic designs which are somewhat less abstract (compared to tensorial data in Chapters 3 and 4). The success of this algorithm shows that the tensor analysis approach can handle data with lower levels of abstraction. However, the data abstraction level can even be lower than the case with the CHAMP framework when moving from hyper-heuristics to metaheuristics where the search heuristics directly deal with candidate solutions.

To evaluate the tensor-based approach on the lowest possible level of data abstraction, it is integrated into a multi-agent system to solve Flow Shop Scheduling Problem. Several agents run in parallel each applying it's own metaheuristic to a given problem instance. Search traces of all the agents are again represented as a $3^{rd}-$order tensor. Unlike hyper-heuristics, the contents of the tensor here are the candidate solutions. That is, the data is rich in details and low in abstraction levels. After factorisation, the emerging pattern is shared with all the agents which use the pattern to construct better solutions. Again the experimental results show that the proposed approach improves the underlying heuristic(s) significantly. Also, in contrast to hyper-heuristics in Chapters 3 and 4 where the tensor design was hard coded, in the case of metaheuristics, there is no need to design the structure of the tensor. A metaheuristic deals directly with candidate solution and each such solution is regarded as a tensor frame. In addition to simplifying the implementation, employing tensor analysis in metaheuristics has thus the advantage of high automation levels.

Of course, there are different machine learning techniques which can be (potentially) used instead of tensor analysis. Nevertheless, compared to those methods, tensor analysis has the advantage of being capable of operating at several levels of data abstraction with minor modifications to the algorithm design. Since many optimisation algorithms leave a high dimensional trace (third order or higher), it further makes sense to use tensor analysis to mine the trace data rather than representing this trace in two dimensions, as required by other existing machine learning techniques. However, tensor analysis is not the holy grail of machine learning techniques and it has its own disadvantages. As observed in Chapter 6, when the number of tensor frames is considerably small (relative to the problem size), the performance of the tensor learning approach could decrease. This is not specific to the problem and generally, tensor analytic approaches are very sensitive to the amount and the sparsity of the data provided to them. Moreover, tensor analysis in heuristic optimisation applications could introduce certain parameters which require tuning. For a good pattern recognition performance (e.g., discovery of latent structures in the data), the number of components (K in Eq.2.7) should be specified manually. Regardless of the tensor structure (e.g. number of dimensions, contents and labels), the best number of components has to be determined prior to the application of tensor analysis to the problem (as in experiments of Chapter 6), which is a time-consuming process. Also, in cases where multiple learning episodes are necessary (as simulated in Chapter 4) the frequency and duration of each episode need to be specified.

The experiments in this study, show that the performance of the tensor-based approach proposed here is not very sensitive to the values of these parameters. This is partly due to the smart design of the tensors. That is, throughout this study, we have assigned labels to data frames and have only added a frame to the tensor if the frame label satisfies some quality-related criteria. For example, in Chapter 3, each data frame has a real-valued label. A frame is added to the tensor if its label and that of the frame next to it are both positive. This, somehow increases the likelihood that the content of the frame yields good changes during the search and renders the frame a promising one. That is why the value of K is consistently 1 (determined either experimentally or manually) in all our work. However, developing a proper labelling strategy may not always be possible. In the absence of a proper labelling strategy or a criteria which determines the quality of the frames, one should experimentally determine the best value for K. There is no specific rule or guideline for the range of potential values for the parameter K and it differs from a dataset to another. The same argument is valid for the frequency and duration of each learning episode in multiple episode version of the proposed approach. That is, there is no general rule using which the size of the data (e.g. number of tensor frames) can be determined. Therefore, the value for these parameters should be decided experimentally (for example, similar to experiments in Sections 3.3.1 and 4.4.1).

7.3 Summary of Contribution

As a conclusion, we have shown that tensor analysis can be coupled to a wide range of heuristic optimisation algorithms and contribute to their performance significantly. Furthermore, the experiments in this study shows that the tensor-based approach can handle data with various levels of abstraction and can thus be a reliable and flexible component of (hyper-/meta) heuristics. At this point, perhaps it would be best to have another look at quality measures mentioned in Section 2.4 (Chapter 2) to assess how much the approach proposed here satisfies these measures.

- **Generality**: As we have seen in Chapter 3, tensor analysis improves the generality level of the (very simple and primitive) underlying hyper-heuristic. The SR-NA hyper-heuristic performs worse than many CHeSC 2011 participants and is considered to be one of the simplest hyper-heuristics. Nevertheless, after embedding the tensor learning into the SR-NA hyper-heuristic, the resulting algorithm performed way better than the SR-NA and delivered high quality solutions across various domains. This indicates that the tensor-based hyper-heuristic has a good (or rather impressive) level of generality.

- **Domain Independent Data**: The proposed approach was tested on several domains, using both short and long run times as well as single and multiple episodes of data collection and analysis. Despite this rigorous testing, the algorithm exhibited a good performance compared to other methods (including the state-of-the-art) and improved the performance of the underlying (hyper-/meta) heuristic significantly in almost all the test scenarios. This is strong evidence that the proposed approach has a good capability in handling domain independent data.

- **Performance**: As mentioned before, many well-known machine learning algorithms have been applied to heuristic optimisation and performed poorly. Examples are Markov decision processes [136] and Reinforcement Learning [130]. Unlike these methods, the tensor analysis approach was applied, not only on the CHeSC 2011 instances but to a wider range of problems, and performed very well. Therefore, we can confirm that the proposed approach satisfies the performance criteria.

- **Agility**: The tensor analysis approach has two major bottlenecks: data collection and factorisation. The CP factorisation method (used throughout this study) is very quick. In the experiments conducted in this study, the time spent for factorisation is negligible (less than a couple of seconds) in all test scenarios. Furthermore, during our experiments we have not felt the need for long training/data collection times. Regardless of the problem domain under investigation, the duration of the

data collection for each learning episode throughout this study ranges between 30 seconds to four or five minutes at most. Moreover, the data collection occurs during the search and is hence online. Thus, the training time is fairly short and the proposed approach can be said to be an agile one. Of course more training (more data) may improve the performance. However, it is important to note that even with such short training times as in our study, the tensor analysis approach performs well.

- **Originality**: In our study, regarding the tensor analysis mechanism and tensorial representation, we did not make any over-simplifying assumptions. We did not alter the original method of factorisation and followed the same train/test standard as in computer vision, web mining, signal processing and etc. Following the standard formulation makes it easier to re-implement the proposed approach.

- **Data Abstraction Levels**: We started this study by applying the proposed approach on highly abstract data and we ended the study by applying it on the data with lowest possible level of abstraction. Along the way, the proposed approach was applied on data produced by various underlying (hyper-/meta) heuristics with very different design philosophies. The nature of the data to which the tensor analysis was exposed varied quite a lot throughout this study. In chapters 3 and 4, the data consisted of tensor frames where each frame contained the binary information regarding pairs of low level heuristics which have been applied consecutively. This data contains little information regarding the problem instances and is highly abstract. In Chapter 5, each tensor frame was a heuristic represented as a policy matrix and contained integer score values. Compared to the first two chapters, the tensorial data here was less abstract as it contained more detail about the heuristic design. This is while, the tensor frames in Chapter 6 were adjacency matrices which represented candidate solutions in form of graphs. That is, the tensorial data in this chapter had minimum level of abstraction as the underlying metaheuristic operated directly on the solution space. However, regardless of the level of abstraction in data, the tensor analysis managed to find useful patterns and improve the performance of the underlying (hyper-/meta) heuristic. This is strong evidence that the proposed approach is capable of handling data with various levels of abstraction.

- **Novelty**: This is the first time that tensor analysis has been considered for heuristic optimisation. However, there are a lot of various novel (machine learning) techniques in the literature which could be used in the field of heuristic optimisation. Part of the plan for our future work will focus on these methods to investigate

how much the field of heuristic optimisation can benefit from these highly advanced learning techniques.

All in all, our experiments show that heuristic optimisation algorithms can greatly benefit from tensor analysis.

7.4 Future Research Directions

This study has introduced the tensor learning approach in the field of heuristic optimisation. We have shown that tensor analysis can be very powerful in detecting useful patterns and be flexible at the same time. The promising results achieved here as well as in previous chapters shows that the tensor analysis approach is indeed an effective learning technique and that it warrants further research. In line with the work of Samorani and Laguna [30], there is mounting evidence that machine learning techniques can greatly contribute to heuristic optimisation and increase the generality level of metaheuristics, potentially making them applicable to multiple problem domains. Thus, there is a wide range of possible directions that should be explored to further benefit from the strengths of this data mining technique.

The field of tensor learning can be considered as a relatively new field of study. The literature related to tensor analysis approaches is growing every day and new methodologies are emerging frequently. Heuristic optimisation can benefit from these new techniques to achieve higher levels of performance and generality. That is, although factorisation does a good job in detecting hidden patterns in the data of this study, better techniques can be applied to tensorial data to detect these patterns. One such example is tensor clustering [288]. Moreover, there are numerous topics related to heuristic optimisation which are left untouched in this study. Machine learning in general and tensor analysis in particular can be considered to address a plethora of challenges in the field of heuristic optimisation. For instance, the role of this learning technique in improving the performance of various components in population-based approaches can be investigated. It is particularly interesting to conduct a study similar to what has been done in Chapter 5, to identify promising alleles for crossover. As discussed in Chapter 4, one of the goals of investigating the longer runs with multiple episodes of learning was to see whether the proposed approach is suitable for the life-long learning context. The good results achieved in the experimental study in that chapter shows that indeed this could be a possible research area. Another interesting application would be to hybridise tensor analysis with the apprenticeship learning approach (please refer to Section 2.3.2) to increase the generality level of a given (hyper-/meta) heuristic. Also, throughout this

study, decisions have been made regarding the design of the tensor such as number of dimensions and the content of the tensor. Although these decisions ultimately lead to good outcomes, other interesting tensor structures should be explored. For example, instead of the adjacency matrices considered for tensor frames in Chapter 6 other interesting graph representations could be tested. Moreover, some methods such as memetic algorithms may leave a trace which has both a tensorial trace (genetic material) and a non-tensorial trace (memetic material). This framework could be extended to accommodate such trace data by using the Coupled Tensor Matrix Factorization [289] approach.

Bibliography

[1] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg, 2001.

[2] G.I. Zobolas, C.D. Tarantilis, and G. Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4):1249 – 1267, 2009. ISSN 0305-0548.

[3] E. Vallada, R. Ruiz, and J.M. Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677, 2015.

[4] M. Nawaz, E. E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[5] X. Dong, H. Huang, and P. Chen. An improved neh-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, 35(12):3962–3968, 2008.

[6] R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5):461–476, 2006.

[7] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[8] K. Sörensen and F. W. Glover. Metaheuristics. In S. I. Gass and M. C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer US, 2013.

[9] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.

[10] W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, (117), 1963.

[11] E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[12] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.

[13] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J.R. Woodward. A classification of hyper-heuristics approaches. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 15, pages 449–468. Springer, 2nd edition, 2010.

[14] E. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In J. Kacprzyk, L. C. Jain, C. L. Mumford, and L. C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01799-5.

[15] P. Ross. Hyper-heuristics. In E. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, 2005.

[16] E. Burke, G. Kendall, M. Mısır, and E. Özcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1):73–90, 2012.

[17] E. Özcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[18] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X, 9780262012430.

[19] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[20] L. Sirovich and M. Kirby. Low-Dimensional Procedure for the Characterization of Human Faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987.

[21] R. Socher. *Recursive Deep Learning for Natural Language Processing and Computer Vision*. PhD thesis, Stanford University, 2014.

[22] M.S. van der Knaap, J. Valk, N. de Neeling, and J.J.P. Nauta. Pattern recognition in magnetic resonance imaging of white matter disorders inchildren and young adults. *Neuroradiology*, 33(6):478–493, 1991. ISSN 0028-3940.

[23] F. Neri and C. Cotta. Memetic algorithms and memeting computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[24] F. Neri, G. Iacca, and E. Mininno. Disturbed exploitation compact differential evolution for limited memory optimization problems. *Information Sciences*, 181 (12):2469 – 2487, 2011. ISSN 0020-0255.

[25] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[26] G. Iacca, F. Neri, E. Mininno, Y.S. Ong, and M.H. Lim. Ockhams razor in memetic computing: Three stage optimal memetic exploration. *Information Sciences*, 188 (0):17 – 43, 2012. ISSN 0020-0255.

[27] F. Caraffini, F. Neri, M. Gongora, and B.N. Passow. Re-sampling search: A seriously simple memetic approach with a high performance. In *2013 IEEE Workshop on Memetic Computing (MC)*, pages 52–59, April 2013.

[28] M. Narita, M. Haraguchi, and Y. Okubo. Data abstractions for numerical attributes in data mining. In H. Yin, N. Allinson, R. Freeman, J. Keane, and S. Hubbard, editors, *Intelligent Data Engineering and Automated Learning IDEAL 2002*, volume 2412 of *Lecture Notes in Computer Science*, pages 35–42. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44025-3.

[29] E. Özcan and A. J. Parkes. Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 2011–2018, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0.

[30] M. Samorani and M. Laguna. Data-mining-driven neighborhood search. *INFORMS J. on Computing*, 24(2):210–227, April 2012. ISSN 1526-5528.

[31] F. Thabtah and P. Cowling. Mining the data from a hyperheuristic approach using associative classification. *Expert Systems with Applications*, 34(2):1093 – 1101, 2008. ISSN 0957-4174.

[32] T.P. Runarsson. Learning heuristic policies a reinforcement learning problem. In C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 423–432. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-25565-6.

[33] K. Krawiec and U. OReilly. Behavioral search drivers for genetic programing. In M. Nicolau, K. Krawiec, M.I. Heywood, M. Castelli, P. García-Sánchez, J.J. Merelo, V.M. Rivas Santos, and K. Sim, editors, *Genetic Programming*, volume 8599 of *Lecture Notes in Computer Science*, pages 210–221. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44302-6.

[34] J.C. Ortiz-Bayliss, H. Terashima-Marín, and S.E. Conant-Pablos. A supervised learning approach to construct hyper-heuristics for constraint satisfaction. In J.A. Carrasco-Ochoa, J.F. Martínez-Trinidad, J.S. Rodríguez, and G.S. di Baja, editors, *Pattern Recognition*, volume 7914 of *Lecture Notes in Computer Science*, pages 284–293. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38988-7.

[35] J.C. Ortiz-Bayliss, H. Terashima-Marin, and S.E. Conant-Pablos. Using learning classifier systems to design selective hyper-heuristics for constraint satisfaction problems. In *2013 IEEE Congress on Evolutionary Computation (CEC)*, pages 2618–2625, June 2013.

[36] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, and E.K. Burke. Hyflex: A benchmark framework for cross-domain heuristic search. In J.K. Hao and M. Middendorf, editors, *European Conference on Evolutionary Computation in Combinatorial Optimisation, EvoCOP '12.*, volume 7245 of *LNCS*, pages 136–147, Heidelberg, 2012. Springer.

[37] S. Asta and E. Özcan. A tensor-based selection hyper-heuristic for cross-domain heuristic search. *Information Sciences*, 299(0):412 – 432, 2015. ISSN 0020-0255.

[38] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363–397. Springer US, 2010.

[39] J. Baxter. Local optima avoidance in depot location. *Journal of the Operation Research Society*, 32:815–819, 1981.

[40] O. Martin, S.W. Otto, and E.W. Felten. Large-step markov chains for the {TSP} incorporating local search heuristics. *Operations Research Letters*, 11(4):219 – 224, 1992. ISSN 0167-6377.

[41] S. Asta and E. Özcan. A tensor analysis improved genetic algorithm for online bin packing. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, New York, NY, USA, 2015. ACM.

[42] S. Asta and E. Özcan. An apprenticeship learning hyper-heuristic for vehicle routing in hyflex. In *Proceedings of the 2014 IEEE Symposium Series on Computational Intelligence*, SSCI 2014, 2014.

[43] S. Asta and E. Ozcan. A tensor-based approach to nurse rostering. *10th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2014)*, 10:442–445, 2014.

[44] S. Asta, E. Özcan, and A.J. Parkes. Batched mode hyper-heuristics. In *LION*, pages 404–409, 2013.

[45] S. Asta, E. Özcan, A. J. Parkes, and A. Etaner-Uyar. Generalizing hyper-heuristics via apprenticeship learning. In *Proceedings of the 13th European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP'13, pages 169–178, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-37197-4.

[46] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[47] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[48] K. Sastry, D. E. Goldberg, and G. Kendall. Genetic algorithms. In E. Burke and G. Kendall, editors, *Search Methodologies*, pages 93–117. Springer US, 2014.

[49] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[50] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 975–980, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[51] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36 (1):267–306, 2009.

[52] I. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, 156(11):2050 – 2069, 2008. ISSN 0166-218X. In Memory of Leonid Khachiyan (1952 - 2005).

[53] A. Subramanian, M. Battarra, and C.N. Potts. An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 52(9):2729–2742, 2014.

[54] N.R. Sabar and G. Kendall. An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. *Omega*, 56(0):88 – 98, 2015. ISSN 0305-0483.

[55] T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519 – 1539, 2006. ISSN 0377-2217.

[56] P. Vansteenwegen, W. Souffriau, G. Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281 – 3290, 2009. ISSN 0305-0548. New developments on hub location.

[57] E. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J.A. Vázquez-Rodríguez, and M. Gendreau. Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, July 2010.

[58] J.D. Walker, G. Ochoa, M. Gendreau, and E. Burke. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, pages 265–276, 2012.

[59] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, editors, *Parallel Problem Solving from Nature PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-87699-1.

[60] J. Michallet, C. Prins, L. Amodeo, F. Yalaoui, and G. Vitry. Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. *Computers & Operations Research*, 41(0):196 – 207, 2014. ISSN 0305-0548.

[61] R. M'Hallah. An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *International Journal of Production Research*, 52(13):3802–3819, 2014.

[62] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA, 1989. ISBN 0201157675.

[63] A. Alkan and E. Özcan. Memetic algorithms for timetabling. In *Congress on Evolutionary Computation, CEC '03.*, volume 3, pages 1796–1802, 2003.

[64] E. Özcan, A.J. Parkes, and A. Alkan. The interleaved constructive memetic algorithm and its application to timetabling. *Computers & Operations Research*, 39 (10):2310–2322, 2012. ISSN 0305-0548.

[65] E. Özcan and A. Alkan. A memetic algorithm for solving a timetabling problem: An incremental strategy. In *Proceedings of the third Multidisciplinary International Conference On Scheduling: Theory and Applications*, pages 394–401, 2007.

[66] E. Burke and J.D. Landa Silva. The design of memetic algorithms for scheduling and timetabling problems. In W. E. Hart, J.E. Smith, and N. Krasnogor, editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 289–311. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-22904-9.

[67] D. Qaurooni. A memetic algorithm for course timetabling. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 435–442, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0.

[68] J. Knowles and D. Corne. Memetic algorithms for multiobjective optimization: Issues, methods and prospects. In W. E. Hart, J.E. Smith, and N. Krasnogor, editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 313–352. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-22904-9.

[69] C.K. Goh, Y.S. Ong, and K.C. Tan. *Multi-Objective Memetic Algorithms*. Springer Berlin Heidelberg, 2009.

[70] H. Ishibuchi, T. Yoshida, and T. Murata. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223, 2003. ISSN 1089-778X.

[71] N. Krasnogor, B.P. Blackburne, E.K. Burke, and J.D. Hirst. Multimeme algorithms for protein structure prediction. In J.J.M. Guervós, P. Adamidis, H.G.

Beyer, H.P.l Schwefel, and J.L. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 769–778. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44139-7.

[72] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000. ISSN 1089-778X.

[73] F. Neri, V. Tirronen, T. Karkkainen, and T. Rossi. Fitness diversity based adaptation in multimeme algorithms:a comparative study. In *IEEE Congress on Evolutionary Computation, 2007. CEC 2007.*, pages 2374–2381, Sept 2007.

[74] P. Moscato and M. G. Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel Computing and Transputer Applications*, 1:177–186, 1992.

[75] L. Buriol, P. M. França, and P. Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506, September 2004. ISSN 1381-1231.

[76] N. Krasnogor and S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.

[77] E. Özcan and C. Başaran. A case study of memetic algorithms for constraint optimization. *Soft Computing*, 13(8-9):871–882, 2009. ISSN 1432-7643.

[78] Y.S. Ong, M.H. Lim, N. Zhu, and K.W. Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):141–152, 2006. ISSN 1083-4419.

[79] N. Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms.* PhD thesis, University of the West of England, Bristol, UK, 2002.

[80] E. Özcan, S. Asta, and C. Altıntaş. Memetic algorithms for cross-domain heuristic search. In Y. Jin and S. Thomas, editors, *Proceedings of the 13th Annual Workshop on Computational Intelligence (UKCI 2013)*, pages 175–182, Surrey, UK, 2013. IEEE Press.

[81] Y.S. Ong, M.H. Lim, and X. Chen. Memetic computation - past, present & future [research frontier]. *IEEE Computational Intelligence Magazine*, 5(2):24–31, May 2010. ISSN 1556-603X.

[82] X. Chen, Y. S. Ong, M. Lim, and K. C. Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011. ISSN 1089-778X.

[83] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: Recent developments. In C. Cotta, M. Sevaux, and K. Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer Berlin Heidelberg, 2008.

[84] G. Kendall and M. Mohamad. Channel assignment optimisation using a hyper-heuristic. In *Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004)*, pages 790–795, Singapore, 2004.

[85] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *PLACEMENT MACHINE, INTECH03 THAILAND*, pages 132–141, 2003.

[86] E. Özcan, Y. Bykov, M. Birben, and E. K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *IEEE Congress on Evolutionary Computation (CEC '09)*, pages 997–1004, 2009.

[87] E. K. Burke, J. D. Landa-Silva, and E. Soubeiga. Multi-objective hyper-heuristic approaches for space allocation and timetabling. In Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces Series*, pages 129–158. Springer US, 2005.

[88] E. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[89] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. pages 523–544. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1-4020-7653-3.

[90] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435, 2007.

[91] K.A. Dowsland, E. Soubeiga, and E. Burke. A simulated annealing based hyper-heuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759 – 774, 2007. ISSN 0377-2217.

[92] D. Ouelhadj and S. Petrovic. A cooperative distributed hyper-heuristic framework for scheduling. In *Proceedings of the IEEE International Conference on Man, Cybernetics, and Systems*, pages 1232–1238, 2008.

[93] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference, MIC 2001*, pages 127–131, 2001.

[94] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G.R. Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43432-0.

[95] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, and F. Paredes. Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization. *Expert Systems with Applications*, 40(5):1690 – 1695, 2013. ISSN 0957-4174.

[96] R. Soto, B. Crawford, S. Misra, W. Palma, E. Monfroy, C. Castro, and F. Paredes. Choice functions for Autonomous Search in Constraint Programming: GA vs PSO. *Technical Gazette*, 20(4):621–629, 2013.

[97] R. Aron, I. Chana, and A. Abraham. A hyper-heuristic approach for resource provisioning-based scheduling in grid environment. *The Journal of Supercomputing*, 71(4):1427–1450, 2015. ISSN 0920-8542.

[98] D. Li, M. Li, X. Meng, and Y. Tian. A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(2):315–325, Feb 2015. ISSN 2168-2216.

[99] B. Bilgin, E. Özcan, and E.E. Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In E. Burke and H. Rudová, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 394–412. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-77344-3.

[100] R. Bai and G. Kendall. An investigation of automated planograms using a simulated annealing based hyper-heuristic. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces Series*, pages 87–108. Springer US, 2005. ISBN 978-0-387-25382-4.

[101] E. Burke and Y. Bykov. A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems. In *PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[102] E. Özcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In T. P. Runarsson, H. G. Beyer, E. Burke, J. J. Merelo-Guerv´s, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin Heidelberg, 2006.

[103] A. Kheiri, E. Özcan, and A. J. Parkes. A stochastic local search algorithm with adaptive acceptance for high-school timetabling. *Annals of Operations Research*, 2014. doi: 10.1007/s10479-014-1660-0.

[104] A. Kheiri and E. Özcan. A hyper-heuristic with a round robin neighbourhood selection. In M. Middendorf and C. Blum, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7832 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013.

[105] E. Özcan, M. Mısır, and A. Kheiri. Group decision making hyper-heuristics for function optimisation. In *Proceedings of the 13th UK Workshop on Computational Intelligence, UKCI 2013, Guildford, United Kingdom, September 9-11, 2013*, pages 327–333. IEEE, 2013.

[106] P. K. Lehre and E. Özcan. A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms XII*, FOGA XII '13, pages 97–104, New York, NY, USA, 2013. ACM.

[107] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In C. L. Mumford and L. C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009.

[108] N.R. Sabar, M. Ayob, G. Kendall, and R. Qu. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE T. Cybernetics*, 45(2):217–228, 2015.

[109] N.R. Sabar and G. Kendall. Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems. *Inf. Sci.*, 314:225–239, 2015.

[110] C.W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics*, 145(1):67 – 77, 2013. ISSN 0925-5273.

[111] C. D. Geiger, R. Uzsoy, and H. Aytug. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *J. of Scheduling*, 9(1):7–34, February 2006.

[112] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett. Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 1227–1234, New York, NY, USA, 2008. ACM.

[113] N. Pillay and W. Banzhaf. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In J. Neves, M.F. Santos, and J.M. Machado, editors, *Progress in Artificial Intelligence*, volume 4874 of *Lecture Notes in Computer Science*, pages 223–234. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-77000-8.

[114] M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In N. Monmarché, E. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79304-5.

[115] E. Burke, M. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *Trans. Evol. Comp*, 14(6):942–958, December 2010.

[116] J. H. Drake, N. Kililis, and E. Özcan. Generation of vns components with grammatical evolution for vehicle routing. In *Proceedings of the 16th European Conference on Genetic Programming*, EuroGP'13, pages 25–36, Berlin, Heidelberg, 2013. Springer-Verlag.

[117] N. Sabar, M. Ayob, G. Kendall, and R. Qu. The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, PP(99), 2014. ISSN 1089-778X.

[118] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 1–8, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5.

[119] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artif. Intell.*, 40(1-3):235–282, September 1989. ISSN 0004-3702.

[120] J.C. Ortiz-Bayliss, H. Terashima-Marin, P. Ross, and S.E. Conant-Pablos. Evolution of neural networks topologies and learning parameters to produce hyper-heuristics for constraint satisfaction problems. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Companion Material Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 261–262, 2011.

[121] M. Mısır, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *Learning and Intelligent Optimization*, pages 461–466. Springer, 2012.

[122] P.C. Hsiao, T.C. Chiang, and L.C. Fu. A vns-based hyper-heuristic with adaptive computational budget of local search. In *2012 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2012.

[123] M. Larose. A hyper-heuristic for the chesc 2011. In *LION6*, 2011.

[124] C. Núñez, J and A. Ceballos. A general purpose hyper heuristic based on ant colony optimization. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/nunez-chesc.pdf`.

[125] T. Cichowicz, M. Drozdowski, M. Frankiewicz, G. Pawlak, F. Rytwiński, and J. Wasilewski. Five phase and genetic hive hyper-heuristics for the cross-domain search. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 354–359. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34412-1.

[126] M. Johnston, T. Liddle, J. Miller, and M. Zhang. A hyperheuristic based on dynamic iterated local search. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/johnston-chesc.pdf`.

[127] C.Y. Chan, F. Xue, W.H. Ip, and C.F. Cheung. A hyper-heuristic inspired by pearl hunting. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 349–353. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34412-1.

[128] D. Meignan. An evolutionary programming hyper-heuristic with co-evolution for chesc'11. 2012. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/meignan-chesc.pdf`.

[129] A. Lehrbaum and N. Musliu. A new hyperheuristic algorithm for cross-domain search problems. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 437–442. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34412-1.

[130] L. Di Gaspero and T. Urli. A reinforcement learning approach for the cross-domain heuristic search challenge. In *The IX Metaheuristics International Conference*, 2011.

[131] F. Mascia and T. Stützle. A non-adaptive stochastic local search algorithm for the chesc 2011 competition. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 101–114. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34412-1.

[132] A. Acuña, V. Parada, and G. Gatica. Cross-domain heuristic search challenge: Giss algorithm presentation. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/acuna-chesc.pdf`.

[133] J. Kubalk. Hyper-heuristic based on iterated local search driven by evolutionary algorithm. In J.K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29123-4.

[134] J. Elomari. Self-search (extended abstract). URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/elomari-chesc.pdf`.

[135] K. Sim. Ksats-hh: A simulated annealing hyper-heuristic with reinforcement learning and tabu-search. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/sim-chesc.pdf`.

[136] K. McClymont and E. C. Keedwell. Markov chain hyper-heuristic (mchh): An online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 2003–2010, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0.

[137] J. Gómez. hybrid adaptive evolutionary algorithm hyper heuristic. URL `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/gomez-chesc.pdf`.

[138] I. Khamassi, M. Hammami, and K. Ghedira. Ant-q hyper-heuristic approach for solving 2-dimensional cutting stock problem. In *2011 IEEE Symposium on Swarm Intelligence (SIS),*, pages 1–7, April 2011.

[139] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[140] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[141] J.H. Drake, E. Özcan, and E. Burke. An improved choice function heuristic selection for cross domain heuristic search. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving From Nature (PPSN XII)*, volume 7492 of *Lecture Notes in Computer Science*, pages 307–316. Springer Berlin Heidelberg, 2012.

[142] E. K. Burke, M. Gendreau, G. Ochoa, and J. D. Walker. Adaptive iterated local search for cross-domain optimisation. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 1987–1994, New York, NY, USA, 2011. ACM.

[143] G. Ochoa, J. Walker, M. Hyde, and T. Curtois. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature (PPSN XII)*, volume 7492 of *Lecture Notes in Computer Science*, pages 418–427. Springer Berlin Heidelberg, 2012.

[144] S. Adriaensen, T. Brys, and A. Nowé. Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic. In Dirk V. Arnold, editor, *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14)*, pages 1303–1310, New York, NY, USA, 2014. ACM.

[145] W. G. Jackson, E. Özcan, and J. H. Drake. Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In *13th UK Workshop on Computational Intelligence (UKCI2013)*, pages 228–235. IEEE, 2013.

[146] S. Chernova and M. Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 233:1–233:8, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5.

[147] S. Chernova and M. Veloso. Multi thresholded approach to demonstration selection for interactive robot learning. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, HRI '08, pages 225–232, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-017-3.

[148] D.H. Grollman and O.C. Jenkins. Dogged learning for robots. In *in 2007 IEEE International Conference on Robotics and Automation (ICRA*, 2007.

[149] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *ECCV (1)*, volume 2350 of *Lecture Notes in Computer Science*, pages 447–460. Springer, 2002. ISBN 3-540-43745-2.

[150] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. Mpca: Multilinear principal component analysis of tensor objects. *IEEE Transactions on Neural Networks*, 19 (1):18–39, 2008.

[151] E. Acar, T. G. Kolda, and D. M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *CoRR*, abs/1105.3422, 2011.

[152] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *CoRR*, abs/1210.7559, 2012.

[153] B. Krausz and C. Bauckhage. Action recognition in videos using nonnegative tensor factorization. In *ICPR*, pages 1763–1766. IEEE, 2010.

[154] D. Xu, S. Yan, Z. Lei, S. Lin, H.J. Zhang, and T. S. Huang. Reconstruction and recognition of tensor-based objects with concurrent subspaces analysis. *IEEE Trans. Circuits Syst. Video Techn.*, 18(1):36–47, 2008.

[155] J. Ye, R. Janardan, and Q. Li. Gpca: an efficient dimension reduction scheme for image compression and retrieval. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD*, pages 354–363. ACM, 2004. ISBN 1-58113-888-1.

[156] Y. Wang and S. Gong. Tensor discriminant analysis for view-based object recognition. In *ICPR (3)*, pages 33–36. IEEE Computer Society, 2006. ISBN 0-7695-2521-0.

[157] D. Tao, X. Li, X. Wu, and S. J. Maybank. General tensor discriminant analysis and gabor features for gait recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(10):1700–1715, 2007.

[158] J. Li, L. Zhang, D. Tao, H. Sun, and Q. Zhao. A prior neurophysiologic knowledge free tensor-based scheme for single trial eeg classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 17(2):107–115, 2009. ISSN 1534-4320.

[159] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Incremental tensor analysis: Theory and applications. *ACM Trans. Knowl. Discov. Data*, 2(3):11:1–11:37, October 2008. ISSN 1556-4681.

[160] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 727–736, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9.

[161] E. Acar, D.M. Dunlavy, and T.G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *IEEE International Conference on Data Mining Workshops, 2009. ICDMW '09.*, pages 262–269, Dec 2009.

[162] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, March 2000. ISSN 0895-4798.

[163] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966c.

[164] R. A. Harshman. *PARAFAC: Methods of three-way factor analysis and multidimensional scaling according to the principle of proportional profiles.* PhD thesis, University of California, Los Angeles, CA, 1976.

[165] A. Shashua and H. Tamir. Non-negative tensor factorization with applications to statistics and computer vision. In *ICML*, pages 792–799, 2005.

[166] W. Wang, J. Zhou, K. He, C. Liu, and J. Xia. Using tucker decomposition to compress color images. In *2nd International Congress on Image and Signal Processing, 2009. CISP '09.*, pages 1–5, Oct 2009.

[167] D. Muti and S. Bourennane. Multidimensional filtering based on a tensor approach. *Signal Processing*, 85(12):2338 – 2353, 2005. ISSN 0165-1684.

[168] X. Guo, X. Huang, L. Zhang, and L. Zhang. Hyperspectral image noise reduction based on rank-1 tensor decomposition. *ISPRS Journal of Photogrammetry and Remote Sensing*, 83(0):50 – 63, 2013. ISSN 0924-2716.

[169] B.W. Bader, M.W. Berry, and M. Browne. Discussion tracking in enron email using parafac. In M.W. Berry and M. Castellanos, editors, *Survey of Text Mining II*, pages 147–163. Springer London, 2008. ISBN 978-1-84800-045-2.

[170] L. De Lathauwer and J. Castaing. Tensor-based techniques for the blind separation of dscdma signals. *Signal Processing*, 87(2):322 – 336, 2007. ISSN 0165-1684. Tensor Signal Processing.

[171] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, August 2009. ISSN 0036-1445.

[172] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener. Modeling and multiway analysis of chatroom tensors. In *Proceedings of the 2005 IEEE International Conference on Intelligence and Security Informatics*, ISI'05, pages 256–268, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-25999-6, 978-3-540-25999-2.

[173] E. Acar, S.A. Çamtepe, and B. Yener. Collective sampling and analysis of high order tensors for chatroom communications. In S. Mehrotra, D.D. Zeng, H. Chen, B. Thuraisingham, and F.Y. Wang, editors, *Intelligence and Security Informatics*, volume 3975 of *Lecture Notes in Computer Science*, pages 213–224. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-34478-0.

[174] A. K. Smilde, R. Bro, and P. Geladi. *Multi-way Analysis with Applications in the Chemical Sciences*. John Wiley & Sons, Ltd., 2004.

[175] B. W. Bader and T. G. Kolda. Algorithm 862: Matlab tensor classes for fast algorithm prototyping. *ACM Trans. Math. Softw.*, 32(4):635–653, December 2006. ISSN 0098-3500.

[176] J.D. Carroll and J.J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970. ISSN 0033-3123.

[177] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16(1):84, 1970.

[178] T.K. Kim and R. Cipolla. Canonical correlation analysis of video volume tensors for action categorization and detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(8):1415–1428, August 2009. ISSN 0162-8828.

[179] L. R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. In C. W. Harris, editor, *Problems in measuring change.*, pages 122–137. University of Wisconsin Press, Madison WI, 1963.

[180] L. R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen, editors, *Contributions to mathematical psychology.*, pages 110–127. Holt, Rinehart and Winston, New York, 1964.

[181] Y.-W. Leung and Y. Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 5(1):41–53, Feb 2001. ISSN 1089-778X.

[182] S. Rahnamayan, H.R. Tizhoosh, and M.M.A. Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79, Feb 2008. ISSN 1089-778X.

[183] J. Hunger and G. Huttner. Optimization and analysis of force field parameters by combination of genetic algorithms and neural networks. *Journal of Computational Chemistry*, 20(4):455–471, 1999. ISSN 1096-987X.

[184] A. Zhou and Q. Zhang. A surrogate-assisted evolutionary algorithm for minimax optimization. In *2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, July 2010.

[185] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *The 2005 IEEE Congress on Evolutionary Computation, 2005.*, volume 2, pages 1777–1784 Vol. 2, Sept 2005.

[186] J. M. Peña, J. A. Lozano, and P. Larrañaga. Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of bayesian networks. *Evol. Comput.*, 13(1):43–66, January 2005. ISSN 1063-6560.

[187] J. Zhang, H. Chung, and W.L. Lo. Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 11(3):326–335, 2007.

[188] H. Zhang and J. Lu. Adaptive evolutionary programming based on reinforcement learning. *Information Sciences*, 178(4):971 – 984, 2008. ISSN 0020-0255.

[189] J. Zhang, Z.H. Zhan, Y. Lin, N. Chen, Y.J. Gong, J.H. Zhong, H.S.H. Chung, Y. Li, and Y.H. Shi. Evolutionary computation meets machine learning: A survey. *IEEE Computational Intelligence Magazine*, 6(4):68–75, Nov 2011. ISSN 1556-603X.

[190] L. Jourdan, C. Dhaenens, and E. Talbi. Using datamining techniques to help metaheuristics: A short survey. In F. Almeida, M.J. Blesa Aguilera, C. Blum, J.M. Moreno Vega, M. Pérez PÉrez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 57–69. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-46384-9.

[191] E. Talbi. A unified taxonomy of hybrid metaheuristics with mathematical programming, constraint programming and machine learning. In E. Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 3–76. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-30670-9.

[192] R. Hamilton-Bryce, P. McMullan, and B. McCollum. Directed selection using reinforcement learning for the examination timetabling problem. In *PATAT '14 Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling*, 2014.

[193] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages

282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

[194] G. E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667.

[195] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M.A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[196] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 1214–1221 Vol.2, Dec 2003.

[197] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003. ISSN 0360-0300.

[198] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, CEC '02, pages 1185–1190, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7803-7282-4.

[199] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8.

[200] William W. C. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[201] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Fourth International Conference on Knowledge Discovery and Data Mining, KDD '98.*, pages 80–86, 1998.

[202] F. Thabtah, P. Cowling, and Y. Peng. Mcar: multi-class classification based on association rule. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications, Cayro, Egypt.*, pages 1–7. IEEE, 2005.

[203] F.A. Thabtah, P. Cowling, and Yonghong P. Mmac: a new multi-class, multi-label associative classification approach. In *Fourth IEEE International Conference on Data Mining, 2004. ICDM '04.*, pages 217–224, Nov 2004.

[204] H.R. Lourenço, O.C. Martin, and T. Stützle. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363–397. Springer US, 2010. ISBN 978-1-4419-1663-1.

[205] E.C. Chi and T.G. Kolda. Making tensor factorizations robust to non-Gaussian noise. Technical Report SAND2011-1877, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, March 2011.

[206] B.W. Bader, T.G. Kolda, et al. Matlab tensor toolbox version 2.5. Available online, January 2012. URL `http://www.sandia.gov/~tgkolda/TensorToolbox/`.

[207] J. Swan, J. Woodward, E. Özcan, G. Kendall, and E. Burke. Searching the hyper-heuristic design space. *Cognitive Computation*, pages 1–8, 2013. ISSN 1866-9956.

[208] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller, J.W. Thatcher, and J.D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972. ISBN 978-1-4684-2003-6.

[209] E. Burke, P. De Causmaecker, G. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *J. of Scheduling*, 7(6):441–499, November 2004. ISSN 1094-6136.

[210] D. Silver, Q. Yang, and L. Li. Lifelong machine learning systems: Beyond learning algorithms. 2013. URL `https://www.aaai.org/ocs/index.php/SSS/SSS13/paper/view/5802/5977`.

[211] E. Hart and K. Sim. On the life-long learning capabilities of a nelli*: A hyper-heuristic optimisation system. In T. Bartz-Beielstein, J. Branke, B. FilipiC, and J. Smith, editors, *Parallel Problem Solving from Nature PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 282–291. Springer International Publishing, 2014. ISBN 978-3-319-10761-5.

[212] K. Sim and E. Hart. An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 121–128, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2662-9.

[213] T. Curtois. Published results on employee scheduling instances. `http://www.cs.nott.ac.uk/~tec/NRP/`.

[214] E. Burke and T. Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014. ISSN 0377-2217.

[215] F. Xue, CY. Chan, WH. Ip, and CF. Cheung. Towards a learning-based heuristic searching reform scheme. In *24th European Conference on Operational Research (EURO XXIV).*, pages 262–269, July 2010.

[216] I. P. Solos, I. X. Tassopoulos, and G. N. Beligiannis. A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem. *Algorithms*, 6(2):278–308, 2013.

[217] S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stolevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1): 221–236, 2014. ISSN 0254-5330.

[218] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425 – 433, 2012. ISSN 0377-2217.

[219] J. Lü and J.K. Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865 – 876, 2012. ISSN 0377-2217.

[220] B. Bilgin, P. Demeester, M. Mısır, W. Vancroonenburg, G. Berghe, and T. Wauters. A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. 2010. URL `Online.https://www.kuleuven-kulak.be/~u0041139/nrpcompetition/abstracts/l3.pdf`.

[221] S. Ceschia, N. Thi Thanh, S. Haspeslagh, and A. Schaerf. The second international nurse rostering competition. In *10th International Conference of the Practice and Theory of Automated Timetabling*, 2014.

[222] C.A. Glass and R.A. Knight. The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research*, 202(2):379 – 389, 2010. ISSN 0377-2217.

[223] J.P. Métivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. In IanP. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 73–87. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04243-0.

[224] S. Loudni and P. Boizumault. Combining vns with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191(3):705 – 735, 2008. ISSN 0377-2217.

[225] E. Burke, J. Li, and R. Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484 – 493, 2010. ISSN 0377-2217.

[226] E. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008. ISSN 0377-2217.

[227] E. Burke, T. Curtois, R. Qu, and G. Berghe. A scatter search approach to the nurse rostering problem, 2010.

[228] P. Brucker, E. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, 16(4):559–573, 2010. ISSN 1381-1231.

[229] M. N. Azaiez and S. S. Al Sharif. A 0-1 goal programming model for nurse scheduling. *Comput. Oper. Res.*, 32(3):491–507, March 2005. ISSN 0305-0548.

[230] T. Curtois, G. Ochoa, M. Hyde, and J.A. Vázquez-Rodríguez. A hyflex module for the personnel scheduling problem. 2009.

[231] S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 399–406, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2958-5.

[232] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):pp. 148–177, 1979. ISSN 00359246.

[233] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms for bin packing: a survey*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997. ISBN 0-534-94968-1.

[234] J. Csirik and G. Woeginger. On-line packing and covering problems. In A. Fiat and G. Woeginger, editors, *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 147–177. Springer Berlin / Heidelberg, 1998.

[235] A. Scholl, R. Klein, and C. Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627 – 645, 1997. ISSN 0305-0548.

[236] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996. ISSN 1381-1231.

[237] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R.L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[238] W. T. Rhee and M. Talagrand. On line bin packing with items of random size. *Mathematics of Operations Research*, 18(2):pp. 438–445, 1993. ISSN 0364765X.

[239] E. G. Coffman Jr, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1 of *Intelligent Systems Reference Library*, pages 151–207. Kluwer Academic Publishers, 1999.

[240] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[241] M.B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203 – 227, 1991.

[242] A. J. Parkes, E. Özcan, and M. R. Hyde. Matrix analysis of genetic programming mutation. In *Proceedings of the 15th European Conference on Genetic Programming*, EuroGP'12, pages 158–169, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29138-8.

[243] S. Asta, E. Özcan, and A. J. Parkes. Dimension reduction in the search for online bin packing policies. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 65–66, New York, NY, USA, 2013. ACM.

[244] A. Yarimcam, S. Asta, E. Özcan, and A. J. Parkes. Heuristic generation via parameter tuning for online bin packing. In *2014 IEEE Symposium on Evolving and Autonomous Learning Systems (EALS)*, pages 102–108, Dec 2014.

[245] A. Prakash, F.T.S. Chan, and S.G. Deshmukh. Fms scheduling with knowledge based genetic algorithm approach. *Expert Systems with Applications*, 38(4):3161 – 3171, 2011. ISSN 0957-4174.

[246] F. Divina and E. Marchiori. Knowledge-based evolutionary search for inductive concept learning. In Y. Jin, editor, *Knowledge Incorporation in Evolutionary Computation*, volume 167 of *Studies in Fuzziness and Soft Computing*, pages 237–253. Springer Berlin Heidelberg, 2005. ISBN 978-3-642-06174-5.

[247] H. Maini, K. Mehrotra, C. K. Mohan, and S. Ranka. Knowledge-based nonuniform crossover. In *IEEE World Congress on Computational Intelligence*, volume 8-4, pages 22–27, Jun 1994.

[248] J. Chen, Y. Xie, and H. Chen. A population-based extremal optimization algorithm with knowledge-based mutation. In Y. Tan, Y. Shi, and C. Coello, editors, *Advances in Swarm Intelligence*, volume 8794 of *Lecture Notes in Computer*

Science, pages 95–102. Springer International Publishing, 2014. ISBN 978-3-319-11856-7.

[249] G. Yan, G. Xie, Z. Chen, and K. Xie. Knowledge-based genetic algorithms. In G. Wang, T. Li, J.W. Grzymala-Busse, D. Miao, A. Skowron, and Y. Yao, editors, *Rough Sets and Knowledge Technology*, volume 5009 of *Lecture Notes in Computer Science*, pages 148–155. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79720-3.

[250] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5(2):10:1–10:27, February 2011. ISSN 1556-4681.

[251] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[252] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing, 2004. Proceedings.*, pages 4–10. IEEE, 2004.

[253] S. H. Clearwater, T. Hogg, and B. A. Huberman. Cooperative problem solving. *Computation: The Micro and the Macro View*, pages 33–70, 1992.

[254] T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 231–231, 1993.

[255] E. G. Talbi and V. Bachelet. Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006. ISSN 1570-1166.

[256] T. G. Crainic and M. Toulouse. *Explicit and emergent cooperation schemes for search algorithms*. Springer, 2008.

[257] M. Milano and A. Roli. Magma: a multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34 (2):925–941, 2004.

[258] D. Meignan, A. Koukam, and J.C. Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[259] D. Ouelhadj and S. Petrovic. A cooperative hyper-heuristic search framework. *Journal of Heuristics*, 16(6):835–857, 2010.

[260] N. El Hachemi, T. G. Crainic, N. Lahrichi, W. Rei, and T. Vidal. Solution integration in combinatorial optimization with applications to cooperative search and rich vehicle routing. 2014.

[261] M.E. Aydin. Coordinating metaheuristic agents with swarm intelligence. *Journal of Intelligent Manufacturing*, 23(4):991–999, 2012.

[262] M. R. Khouadjia, E.G. Talbi, L. Jourdan, B. Sarasola, and E. Alba. Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. *The Journal of Supercomputing*, 63(3):836–853, 2013.

[263] S. Martin, D. Ouelhadj, P. Beullens, E. Ozcan, A. A. Juan, and E. Burke. A multi-agent based cooperative approach to scheduling and routing. *Submitted to Euroean JOurnal of Operations Research*, 2015.

[264] S. Martin, D. Ouelhadj, P. Smet, G. Berghe, and E. Özcan. Cooperative search for fair nurse rosters. *Expert Systems with Applications*, 40(16):6674–6683, 2013.

[265] E. Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

[266] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, New Jersey, 2002.

[267] C. L. Chen and R. L. Bulfin. Complexity of single machine, multi-criteria scheduling problems. *European Journal of Operational Research*, 70(1):115–125, 1993.

[268] S.M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[269] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

[270] B. Naderi and R. Ruiz. The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768, 2010.

[271] J.N.D. Gupta and E. F. Stafford. Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711, 2006.

[272] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1):65–74, 1990.

[273] I.H. Osman and C.N. Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[274] V. Fernandez-Viagas and J. M. Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, 2014.

[275] A. A. Juan, J. Faulin, R. Ruiz, B. Barrios, and S. Caballé. The sr-gcws hybrid algorithm for solving the capacitated vehicle routing problem. *Applied Soft Computing*, 10(1):215–224, 2010.

[276] A. A. Juan, H. R. Lourenço, M. Mateo, R. Luo, and Q. Castella. Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues. *International Transactions in Operational Research*, 21 (1):103–126, 2014.

[277] S.W. Lin and K.C. Ying. Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm. *Computers & Operations Research*, 40(6):1625–1647, 2013.

[278] J. Grabowski and M. Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 31(11):1891–1909, 2004.

[279] Y.R. Tzeng, C.L. Chen, and C.L. Chen. A hybrid eda with acs for solving permutation flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1139–1147, 2012.

[280] C.L. Chen, S.Y. Huang, Y.R. Tzeng, and C.L. Chen. A revised discrete particle swarm optimization algorithm for permutation flow-shop scheduling problem. *Soft Computing*, 18(11):2271–2282, 2014.

[281] C.Y. Hsu, P.C. Chang, and M.H. Chen. A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 83:159–171, 2015.

[282] S.H. Chen, P.C. Chang, T.C.E. Cheng, and Q. Zhang. A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers & Operations Research*, 39(7):1450–1457, 2012.

[283] F. Ahmadizar. A new ant colony algorithm for makespan minimization in permutation flow shops. *Computers & industrial engineering*, 63(2):355–361, 2012.

[284] C.L. Chen, Y.R. Tzeng, and C.L. Chen. A new heuristic based on local best solution for permutation flow shop scheduling. *Applied Soft Computing*, 29:75–81, 2015.

[285] D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. The ieee fipa approach to integrating software agents and web services. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 276. ACM, 2007.

[286] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[287] Foundation for Intelligent Physical Agents. Fipa communicative act library specification, 2008. URL URLhttp://www.fipa.org/specs/fipa00037/SC00037J/index.html.

[288] A.R. Benson, D.F. Gleich, and J. Leskovec. Tensor spectral clustering for partitioning higher-order network structures. *CoRR*, abs/1502.05058, 2015.

[289] Evrim Acar, Tamara G. Kolda, and Daniel M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. In *MLG'11: Proceedings of Mining and Learning with Graphs*, August 2011.