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

Shahriar Asta, Ender Özcan

*ASAP Research Group*
*School of Computer Science*
*University of Nottingham*
*NG8 1BB, Nottingham, UK*

**Abstract**

Hyper-heuristics have emerged as automated high level search methodologies that manage a set of low level heuristics for solving computationally hard problems. A generic selection hyper-heuristic combines heuristic selection and move acceptance methods under an iterative single point-based search framework. At each step, the solution in hand is modified after applying a selected heuristic and a decision is made whether the new solution is accepted or not. In this study, we represent the trail of a hyper-heuristic as a third order tensor. Factorization of such a tensor reveals the latent relationships between the low level heuristics and the hyper-heuristic itself. The proposed learning approach partitions 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 is formed and while solving a given problem instance, heuristics are allowed to operate only in conjunction with the associated acceptance method at each stage. 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 hyper-heuristic in the prescribed manner. The empirical results across six different problem domains from a benchmark indeed indicate the success of the proposed approach.

*Email addresses:* sba@cs.nott.ac.uk (Shahriar Asta), Ender.Ozcan@nottingham.ac.uk (Ender Özcan)

*Keywords:* Hyper-heuristic, Data Science, Machine Learning, Move Acceptance, Tensor Analysis, Algorithm Selection

## 1. Introduction

Hyper-heuristics have emerged as effective and efficient methodologies for solving hard computational problems. They perform search over the space formed by a set of low level heuristics, rather than solutions directly [49]. 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. More on hyper-heuristics can be found in [12, 16, 49]. The focus of this paper is on selection hyper-heuristics which, often operate under a single point based search framework by improving an initially created solution iteratively, exploiting the strengths of multiple low level heuristics. At each step, a complete solution is updated forming a new solution using a selected heuristic and this new solution is considered for use in the next step via a move acceptance method.

There are some recent studies indicating the effectiveness and potential of mixing multiple move acceptance methods under a selection hyper-heuristic framework. Kheiri and Özcan [28] described a bi-stage hyper-heuristic which allows improving and equal moves only in the first stage while a naive move acceptance method allows worsening moves in the following stage. Özcan et al. [47] recently combined different move acceptance methods and tested different group decision making strategies as a part of selection hyper-heuristics. One of the rare theoretical studies reveal that mixing move acceptance within a selection hyper-heuristic framework could yield a better running time on some benchmark functions [36]. Machine learning techniques, such as reinforcement learning and learning classifier systems have been used as a component of selection hyper-heuristics since the early ideas have emerged [23]. In this study, we propose a multi-stage selection hyper-heuristic, hybridizing two simple move acceptance methods, which is significantly improved by the use of a machine learning technique, namely tensor analysis [40].

In the proposed approach, we represent the trail of a selection hyper-heuristic as a $3^{rd}$ order tensor. 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 [38] 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 a benchmark indicate the success of the proposed hyper-heuristic mixing different acceptance methods.

This paper is organized as follows. An overview of hyper-heuristics, together with the description of the benchmark framework used in this paper is given in Section 2. Section 3 includes a description of the data analysis method we have used in our study. Section 4 discusses a detailed account of our framework while experimental design issues as well as the results are discussed in Section

5. Finally, conclusion is provided in Section 6.

## 2. Selection Hyper-heuristics

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. The framework proposed in this paper is a single point based search algorithm which fits best in the online learning selection hyper-heuristic category.

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. A survey on hyper-heuristics including their components can be found in [12, 49].

### *2.1. Heuristic Selection and Move Acceptance Methodologies*

In this section, we describe some of the basic and well known heuristic selection approaches. [20, 21] 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) [20, 22, 51] 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 [43], 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 [14], 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. There are numerous other heuristic selection mechanisms. Interested reader can refer to [12] for further detail.

As for the move acceptance component, currently, there are two types of move acceptance methods: *deterministic* and *non-deterministic* [12]. 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, utilizing 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), Improvement and Equal (IE) and Naive Acceptance (NA) [20]. 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 $\alpha$. Although there are more elaborate move acceptance methods, for example, Monte-Carlo based move acceptance strategy [7], Simulated Annealing [10], Late Acceptance [11], 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. [46] shows that the performance of a selection hyper-heuristic could vary if the set of low level heuristics change, as expected. [29] 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 [28, 36, 47]. Hence, in this study, we fix the heuristic selection method as SR and propose an online learning method to partition the low level heuristics, predicting which ones would perform well with a naive move acceptance method and assigning the others to IE. Then we mix the move acceptance methods under a multi-stage selection hyper-heuristic framework invoking only the associated heuristics when chosen.

## *2.2. HyFlex and the First Cross-Domain Heuristic Search Challenge (CHeSC 2011)*

Hyper-heuristics Flexible Framework (HyFlex) [44] 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 [19] 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 1). On the other hand, pieces of problem dependent information, such as, 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.

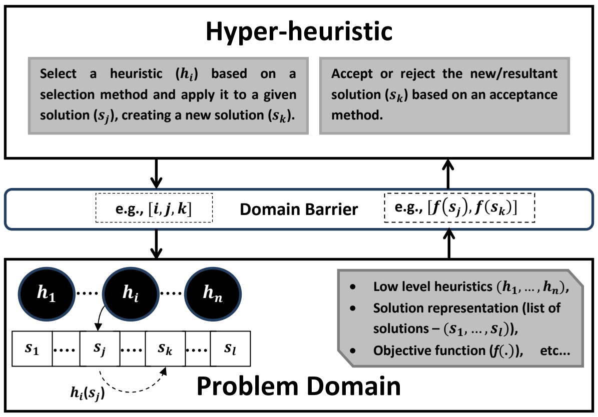

Hyper-heuristic
Select a heuristic ($h_i$) based on a selection method and apply it to a given solution ($s_j$), creating a new solution ($s_k$).
Accept or reject the new/resultant solution ($s_k$) based on an acceptance method.
e.g., $[i, j, k]$
Domain Barrier
e.g., $[f(s_j), f(s_k)]$
$h_1$
$h_i$
$h_n$
$s_1$
$s_j$
$s_k$
$s_l$
$h_i(s_j)$
Low level heuristics ($h_1, \dots, h_n$),
Solution representation (list of solutions – ($s_1, \dots, s_l$)),
Objective function ($f(.)$), etc...
Problem Domain

Figure 1: A selection hyper-heuristic framework [19].

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

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

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 1 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 pro-

Table 1: 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 |

vided in Table 2. The top three selection hyper-heuristics that generalize well across the CHeSC 2011 problem domains are multi-stage approaches of AdapHH [41], VNS-TW [27] and ML [33].

## 3. Tensor Analysis

*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 [31], a boldface Euler script letter, boldface capital letter and boldface lower-case letter denote a tensor (e.g., $\boldsymbol{\mathcal{T}}$), 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 $\boldsymbol{\mathcal{T}}$ is denoted as $t_{pqr}$.

### *3.1. Tensor Decomposition (Factorization)*

Tensor decomposition (a.k.a tensor factorization) is used in many research fields to identify the correlations and relationships among different modes of high dimensional data. The tensor decomposition methods are mainly generalizations of the Singular Value Decomposition (SVD) to higher dimensions. Higher Order SVD (HOSVD) [35], Tucker decomposition [55], Parallel Factor

Table 2: 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 | 181 | 11 | ACO-HH | 39 |
| 2 | VNS-TW | 134 | 12 | GenHive | 36.5 |
| 3 | ML | 131.5 | 13 | DynILS | 27 |
| 4 | PHUNTER | 93.25 | 14 | SA-ILS | 24.25 |
| 5 | EPH | 89.75 | 15 | XCJ | 22.5 |
| 6 | HAHA | 75.75 | 16 | AVEG-Nep | 21 |
| 7 | NAHH | 75 | 17 | GISS | 16.75 |
| 8 | ISEA | 71 | 18 | SelfSearch | 7 |
| 9 | KSATS-HH | 66.5 | 19 | MCHH-S | 4.75 |
| 10 | HAEA | 53.5 | 20 | Ant-Q | 0 |

(a.k.a PARAFAC or CANDECOMP or CP) [26] and Non-negative Tensor Factorization (NTF) [50] are among numerous factorization methods proposed by researchers. Tucker decomposition has applications in data compression [57], dimensionality reduction [56] and noise removal [42] among others. Also CP decomposition has been used for noise removal and data compression [24]. In addition, it has a wide range of applications in many scientific areas such as Data Mining [8] and telecommunications [34]. There are few other factorization 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 [31]. Also, the studies in [2] and [1] provide very detailed comparison between factorization methods.

In this study, we have used the CP decomposition method. Furthermore, we focus on $3^{rd}$-order tensors. Assume that we have such tensor denoted by $\mathcal{T}$ with a size $P \times Q \times R$. CP decomposition utilizes the Alternating Least Square

(ALS) algorithm [15, 25], in which tensor $\mathcal{T}$ is approximated by another tensor $\hat{\mathcal{T}}$ as in Equation 1.

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

$\lambda_k \in \mathbb{R}_+$, $\mathbf{a}_k \in \mathbb{R}^P$, $\mathbf{b}_k \in \mathbb{R}^Q$ 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. $\lambda_k$ is the weight of the $k$th component. 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 1 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 $a_p b_q c_r$.

The outer product $\mathbf{a}_k \circ \mathbf{b}_k$ in Equation 1 quantifies the relationship between the object pairs, i.e., score values indicating the "level of interaction" between object pairs in component $k$ [31]. The purpose of the ALS algorithm is to minimize the error difference between the original tensor and the estimated tensor, denoted as $\varepsilon$ as follows:

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

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

$$||\mathcal{T} - \hat{\mathcal{T}}||_F^2 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \left(t_{pqr} - \hat{t}_{pqr}\right)^2 \tag{3}$$

After applying the factorization 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 $\phi$, is computed using the following equation:

$$\phi = 1 - \frac{||\mathcal{T} - \hat{\mathcal{T}}||_F}{||\mathcal{T}||_F} \tag{4}$$

where the value $\phi$ is in the range $[0, 1]$. The $\phi$ 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 factorization (where $\mathcal{T} = \hat{\mathcal{T}}$) results in $\phi = 1$, while a poor factorization results in $\phi = 0$, hence as the $\phi$ value increases, so does the factorization accuracy. More on tensor decompositions and their applications can be found in [31].

As well as representing data in a concise and more generalizable manner which is immune to data anomalies (such as missing data), tensor factorization 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, [30] and [32] separately show that factorization 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.

## *3.2. Motivation: Inspirations From Applications of 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, [56] 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 [39]. 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 ([4], [5]). Similar claims have been registered in various research areas such as human action recognition in videos [32], hand written digit recognition [59], image compression [60], object recognition [58], gait recognition [54], Electroencephalogram (EEG) classification [37], Anomaly detection in streaming data [52], dimensionality reduction [39], tag recommendation systems [48] and Link Prediction on web data [3].

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 $3^{rd}$-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 hypothetical tensor representing the search history.

Some selection hyper-heuristics, such as reinforcement-based hyper-heuristics or hyper-heuristics embedding late acceptance or tabu search components do use some of the search history selectively. Their performance 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 study, we use a $3^{rd}$-order tensor to represent the search history of a hyper-heuristic. The first two modes 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 [32], except that, instead of examining the video of human body motion and looking for different body parts moving in harmony, we examine 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.

## 4. A Tensor-based Selection Hyper-Heuristic Approach Hybridizing Move Acceptance

The low level heuristics in HyFlex are divided into four groups as described in Section 2. 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 study 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 the paper. 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 factorization, (iv) tensor analysis, and (v) hybrid acceptance as illustrated in Figure 2. 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 factorization, 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 study are minimizing problems. A detailed description of each phase is given in the subsequent sections.

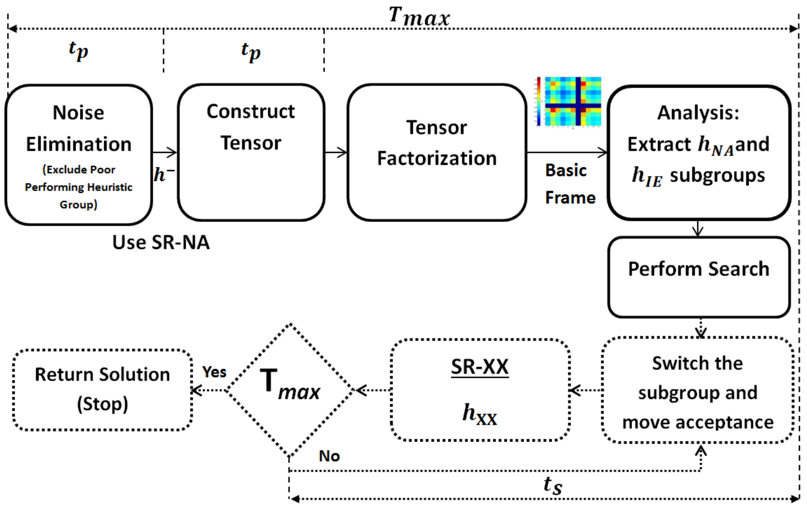


Figure 2: The schematic of our proposed framework.

## 4.1. 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 structures. Thus, reducing the noise is a necessary step in constructing datasets. Under the selection hyper-heuristic framework in which a low level heuristic is chosen randomly, if a heuristic perturbs a solution and consistently generates highly worsening solutions, then such a heuristic is considered as a *poor* heuristic, causing partial re-starts which is often not a desirable behaviour. Hence, the tensor dataset produced while such a heuristic is used can be considered *noisy* (noise is generally defined as undesired data) and that heuristic can be treated as source of the noise. Identifying such heuristics at the start and eliminating them results in a less noisy dataset.

The type of noise happens to be very important in many data mining techniques and tensor factorization is not an exception. CP factorization method which is one of the most widely used factorization algorithms, assumes a Gaussian type noise in the data. It has been shown that CP is very sensitive to non-Gaussian noise types [18]. 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 factorization 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.

## *4.2. Tensor Construction and Factorization*

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. The tensor $\mathcal{T}$ 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.

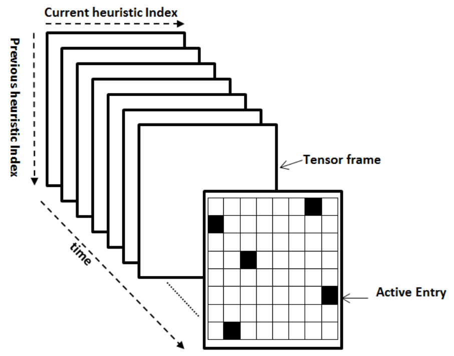


Figure 3: The tensor structure in TeBHA-HH. The black squares (also referred to as active entries) within a tensor frame highlight heuristic pairs invoked subsequently by the underlying hyper-heuristic.

The tensor frame is filled with binary data as demonstrated in Algorithm 1. 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 ($\delta_f$). 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).

While appending a frame to the tensor, each tensor frame is labelled by the $\Delta_f$ it yields, where $\Delta_f$ is the overall change in objective value caused during the construction of the frame. $\Delta_f$ is different from $\delta_f$ 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.

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 1). The following section illustrates how the basic frame is used in our approach.

**Algorithm 1:** The tensor construction phase

```
In: $\boldsymbol{h} = \boldsymbol{h}^-$;
Initialize tensor frame $\mathbf{M}$ to 0;
$counter = 0$;
while $t < t_p$ do
    if $counter = \lfloor |\boldsymbol{h}|/2 \rfloor$ then
        append $\mathbf{M}$ to $\mathcal{T}$;
        set frame label to $\Delta_f$;
        Initialize tensor frame $\mathbf{M}$ to 0;
        $counter = 0$;
    end
    $h_{previous} = h_{current}$;
    $h_{current} =$ selectHeuristic($\boldsymbol{h}$);
    $f_{current} = f_{new}$;
    $f_{new}$ =applyHeuristic($h_{current}$);
    $\delta_f = f_{current} - f_{new}$;
    if $\delta_f > 0$ then
        $m_{h_{previous}, h_{current}} = 1$;
        $counter++$;
    else
        if $probability > 0.5$ then
            $m_{h_{previous}, h_{current}} = 1$;
            if $\delta_f = 0$ then
                $m_{h_{current}, h_{previous}} = 1$;
            end
            $counter++$;
        end
    end
end
```

## *4.3. Tensor Analysis: Interpreting The Basic Frame*

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

$$\hat{\mathcal{T}} = \lambda\ \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \tag{5}$$

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, 5 of them being mutation and 2 of them being local search heuristics. Moreover, the factorization of the tensor produces a basic frame as illustrated in Figure 4. 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 corresponding to this maximum pair are then sorted to determine a ranking between the heuristics. Since heuristics $LS0$ and $LS1$ are already the maximizing pair, they are considered to be the top two elements of this list. In the basic frame of Figure 4 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 ($\boldsymbol{h}_{IE}$).

## *4.4. 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 [27] and ML [33] assume

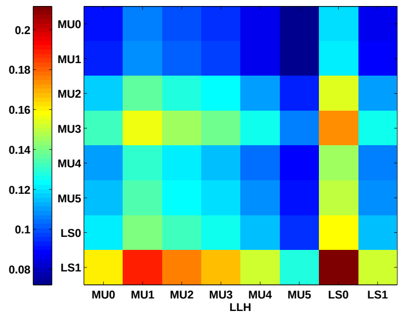


Figure 4: A sample basic frame. Each axis of the frame represents heuristic indexes. Higher scoring pairs of heuristics are darker in color.

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 [41] 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 this paper, 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. 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 2) 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) [38] and memetic algorithms (MAs) [17, 45]. 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.

## 5. 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 benchmarking tool provided by the CHeSC 2011 organizers. This is to ensure a fair comparison between various algorithms. We used the Matlab Tensor Toolbox [9] [2] for tensor operations. The HyFlex, as well as the implementation of our framework is in Java. Hence, subsequent to

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

**Algorithm 2:** Tensor Based Hyper-heuristic with Hybrid Acceptance Strategy

```
1  let $\boldsymbol{h}$ be the set of all low level heuristics;
2  let $t$ be the time elapsed so far;
3  $\boldsymbol{h}_x^- $ = exclude XO heuristics;
4  $\boldsymbol{h}^- = $ preProcessing($t_p, \boldsymbol{h}_x^-$);
5  $\mathcal{T} = $ constructTensor($t_p, \boldsymbol{h}^-$);
6  $\mathbf{a}, \mathbf{b}, \mathbf{c} = \mathrm{CP}(\mathcal{T}, K = 1)$ , $\mathbf{B} = \mathbf{a} \circ \mathbf{b}$;
7  $x, y = max(\mathbf{B})$;
8  $\boldsymbol{h}_s = sort(\mathbf{B}_{i=1:|\boldsymbol{h}^-|,y})$;
9  $\boldsymbol{h}_{NA} = (\boldsymbol{h}_s)_{i=1:\lfloor |\boldsymbol{h}_s|/2 \rfloor}$ , $\boldsymbol{h}_{IE} = \boldsymbol{h}_x^- - \boldsymbol{h}_{NA}$;
10 while $t < T_{max}$ do
11     if $acceptance = NA$ then
12         h = selectRandomHeuristic($\boldsymbol{h}_{NA}$);
13     else
14         h = selectRandomHeuristic($\boldsymbol{h}_{IE}$);
15     end
16     $s_{new}, f_{new} = $ applyHeuristic($h, s_{current}$);
17     $\delta = f_{old} - f_{new}$;
18     updateBest($\delta$,$f_{new}$);
19     if $acceptanceTimer \geq t_s$ then
20         toggle acceptance mechanism;
21     end
22     if $switch = true$ then
23         $s_{new}, s_{old} = $NA($s_{new}, f_{new}, s_{current}, f_{current}$) ;
24     else
25         $s_{new}, s_{old} = $IE($s_{new}, f_{new}, s_{current}, f_{current}$);
26     end
27 end
```

tensor construction phase, Matlab is called from within Java to perform the factorization task. Throughout the paper, 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.

## *5.1. 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 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 paper.

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 factorization 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.

## *5.2. Pre-processing Time*

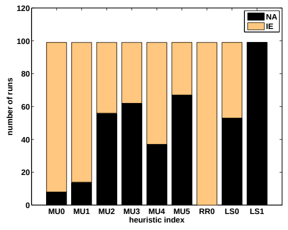
The experiments in this section concentrates on the impact of the time ($t_p$) given to the first two phases (noise elimination and tensor construction) on the performance of the overall approach. During the first phase in all of the runs, the $RR$ group of heuristics have been identified as source of 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 5.1, 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 5 compare the histograms of three different $t_p$ values for the first instances of the

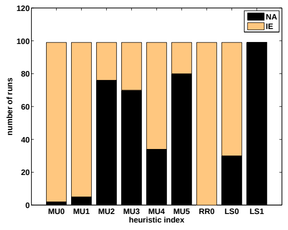
6 problem domains in HyFlex.

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 5(a), 5(b) and 5(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 factorization method, having several good options (heuristic subsets), chooses various heuristic subsets in various runs.

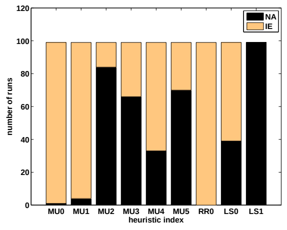
Interestingly, $RR0$ and $LS1$ are diversifying and intensifying heuristics respectively. Assigning $RR0$ to $\boldsymbol{h}_{IE}$ means that the algorithm usually chooses diversifying operations that actually improves the solution. A tendency to such assignments is observable for other problem instances, 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 heuristics, as we will see later in this section, it often results in a *good* balance between intensification and diversification during the search.



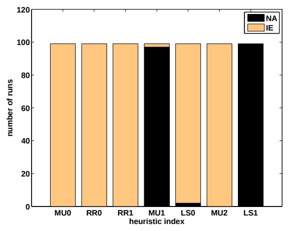
(a) Max-SAT, $t_p = 15sec$



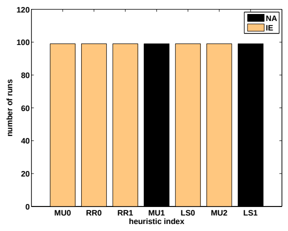
(b) Max-SAT, $t_p = 30sec$



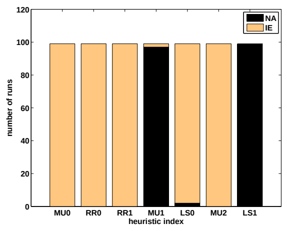
(c) Max-SAT, $t_p = 60sec$



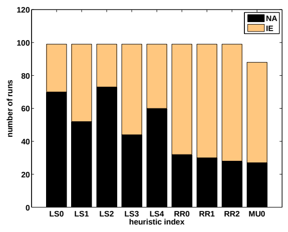
(d) BP, $t_p = 15sec$



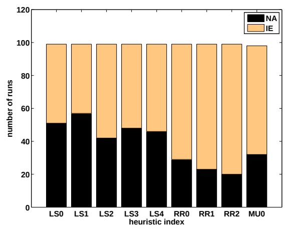
(e) BP, $t_p = 30sec$



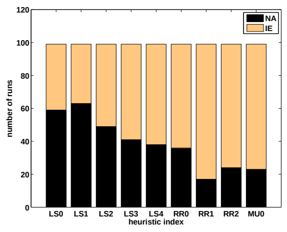
(f) BP, $t_p = 60sec$



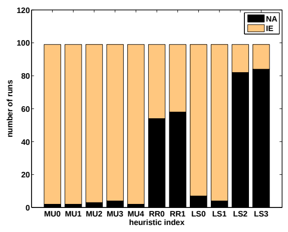
(g) PS, $t_p = 15sec$



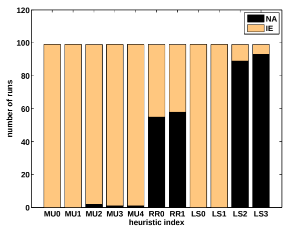
(h) PS, $t_p = 30sec$



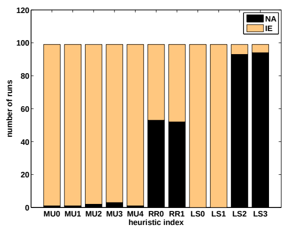
(i) PS, $t_p = 60sec$



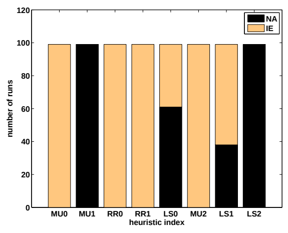
(j) FS, $t_p = 15sec$



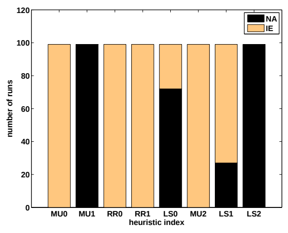
(k) FS, $t_p = 30sec$



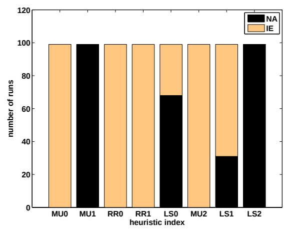
(l) FS, $t_p = 60sec$



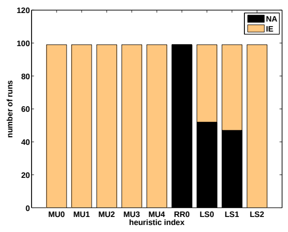
(m) VRP, $t_p = 15sec$



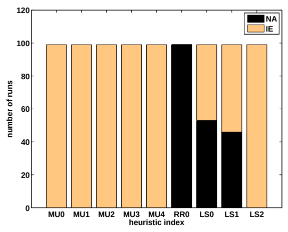
(n) VRP, $t_p = 30sec$



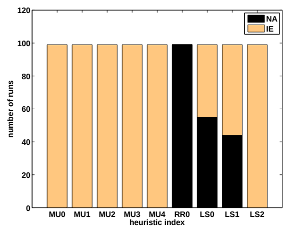
(o) VRP, $t_p = 60sec$



(p) TSP, $t_p = 15sec$



(q) TSP, $t_p = 30sec$

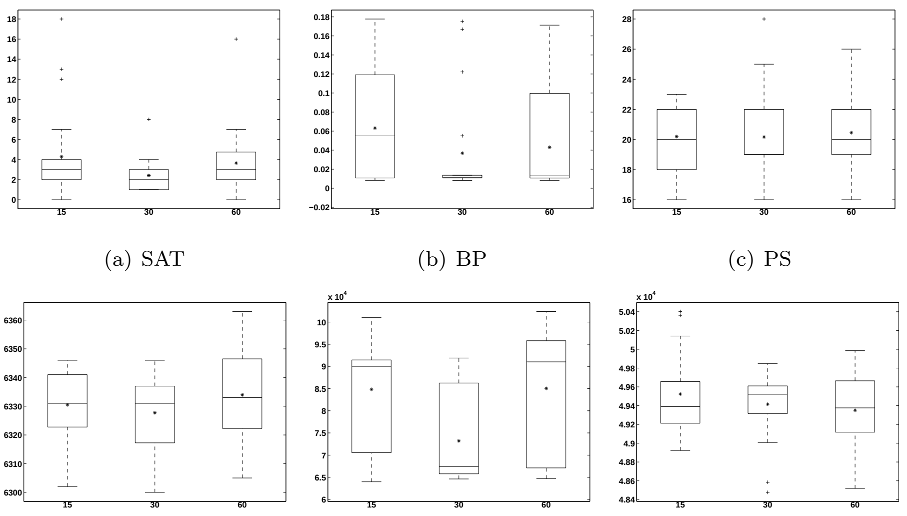


(r) TSP, $t_p = 60sec$

Figure 5: Histograms of heuristics selected as $\boldsymbol{h}_{NA}$ and $\boldsymbol{h}_{IE}$ for various $t_p$ values across all CHeSC 2011 problem domains.

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 5. 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 6 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 4) ii) No Noise Elimination (NONE) in which the first phase of our algorithm is entirely ignored iii) RR-LS

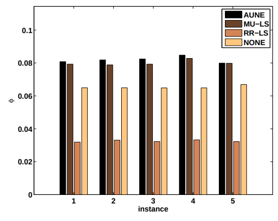


(a) SAT (b) BP (c) PS

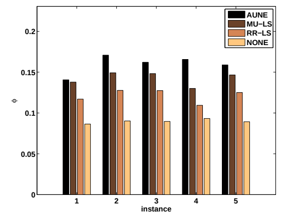
(d) FS (e) VRP (f) TSP

Figure 6: 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.

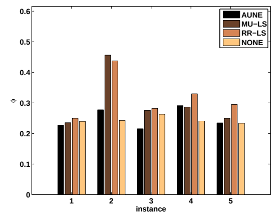
in which ruin and recreate and Local Search heuristics only are participated in tensor construction and iv) MU-LS where mutation and local search heuristics are considered in tensor construction. Each scenario is tested on all CHeSC 2011 instances and during those experiments $t_p$ is fixed as 30 seconds. After performing the factorization, the $\phi$ value (Equation 4) is calculated for each instance at each run. Figure 7 provides the performance comparison of different noise elimination strategies based on the $\phi$ values averaged over 31 runs for each instance. It is desirable that the $\phi$ 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 $\phi$ 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 5(g), 5(h) and 5(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 factorization 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 5 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.



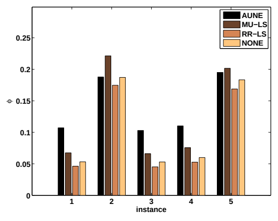
(a) SAT



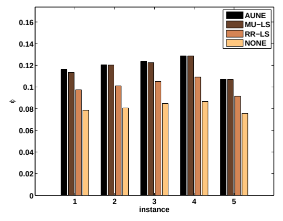
(b) BP



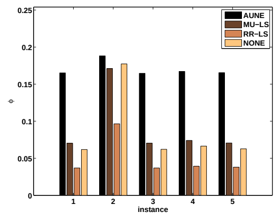
(c) PS



(d) FS



(e) VRP

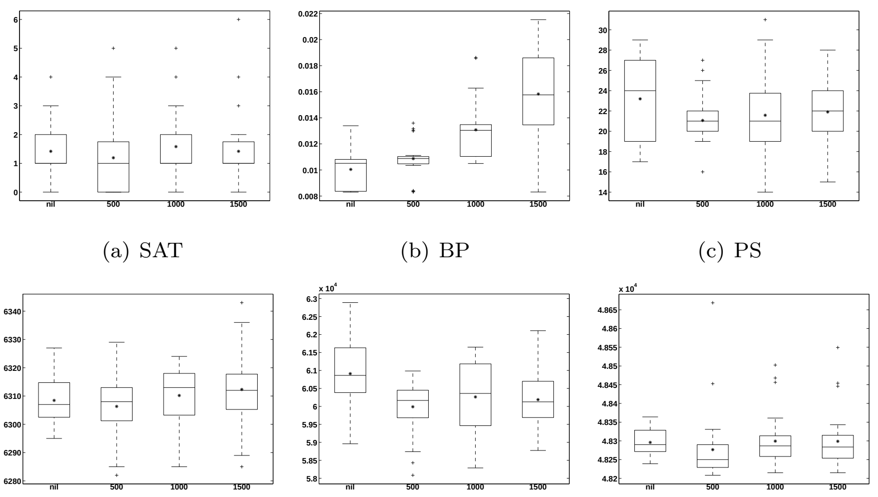


(f) TSP

Figure 7: Comparing the model fitness in factorization, $\phi$ (y axis of each plot), for various noise elimination strategies. Higher $\phi$ values are desirable. The x-axis is the ID of each instance from the given CHeSC 2011 domain.

## 5.3. 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

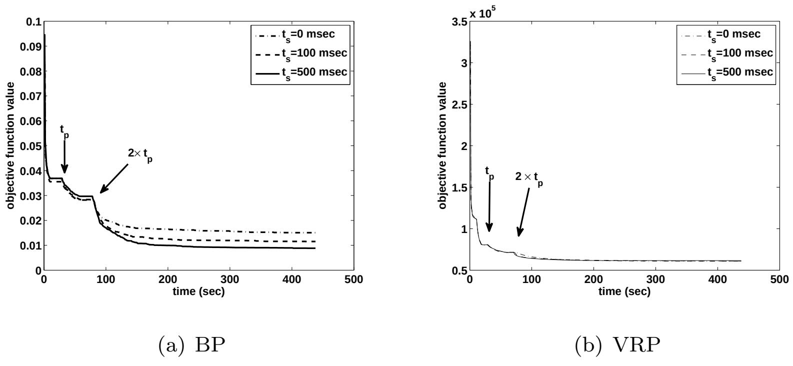
(d) FS (e) VRP (f) TSP

Figure 8: 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.

A comparison between various values considered for $t_s$ is given in Figure 8. Judging by the average performance (shown by an asterisk on each box), $t_s =$ 500 msec performs slightly better than other values. Figure 9 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.

## 5.4. 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 elimina-



(a) BP (b) VRP

Figure 9: 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.

tion, we run another round of experiments testing the algorithm on all CHeSC 2011 instances. Table 3 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 [41] is included in the comparisons. Table 4 provides the details of the average performance 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. Despite this, the TeBHA-HH manages to perform significantly better than its building blocks of SR-IE and SR-NA as illustrated in Figure 10 on all almost all domains. The same behaviour is

Table 3: The performance of the TeBHA-HH framework on each CHeSC 2011 instance over 31 runs, where $\mu$ and $\sigma$ 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 | $\mu$ : | 1.9 | 4.0 | 1.3 | 3.8 | 7.4 |
| | min : | 0 | 1 | 0 | 1 | 7 |
| | $\sigma$ : | 2.3 | 3.5 | 1.0 | 3.0 | 0.8 |
| BP | $\mu$ : | 0.0116 | 0.0086 | 0.0107 | 0.1087 | 0.0173 |
| | min : | **0.0083** | 0.0035 | 0.0058 | 0.1083 | 0.0114 |
| | $\sigma$ : | 0.0013 | 0.0032 | 0.0027 | 0.0007 | 0.0069 |
| PS | $\mu$ : | 20.8 | 9618.1 | 3241.8 | 1611.2 | 349.1 |
| | min : | 13 | 9355 | 3141 | 1458 | 310 |
| | $\sigma$ : | 4.3 | 129.5 | 59.7 | 101.2 | 23.9 |
| FS | $\mu$ : | 6310.9 | 26922.2 | 6372.5 | 11491.3 | 26716.2 |
| | min : | 6289 | 26875 | 6364 | 11468 | 26606 |
| | $\sigma$ : | 11.3 | 27.1 | 5.8 | 16.1 | 37.5 |
| VRP | $\mu$ : | 59960.6 | 13367.9 | 148318.6 | 20862.3 | 147540.2 |
| | min : | **57715.2** | 13297.9 | 143904.5 | 20656.2 | 145672.5 |
| | $\sigma$ : | 30.7 | 5.3 | 44.8 | 20.0 | 31.1 |
| TSP | $\mu$ : | 48274.2 | 20799913.7 | 6874.7 | 66812.4 | 53392.0 |
| | min : | 48194.9 | **20657952.5** | 6851.1 | 66074.7 | 52661.2 |
| | $\sigma$ : | 6.9 | 442.1 | 3.2 | 17.5 | 25.7 |

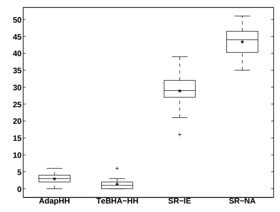
Table 4: Average performance comparison of TeBHA-HH to AdapHH, the winning hyper-heuristic of CHeSC 2011 for each instance. Wilcoxon signed rank test is performed as a statistical test on the objective values obtained over 31 runs from TeBHA-HH and AdapHH. $\leq$ ($<$) denotes that TeBHA-HH performs 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 |

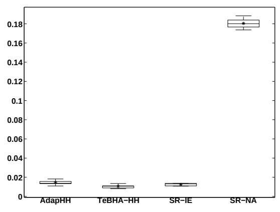
observed across the rest of the CHeSC 2011 instances.

### *5.4.1. Performance comparison to the competing algorithms of CHeSC 2011*

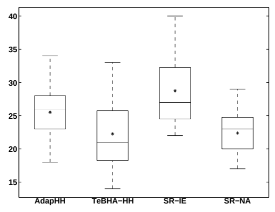
The results obtained from the experiments as described in the previous section, are then compared to the results achieved by all CHeSC 2011 contestants. We used the Formula 1 scoring system provided by the organizers to determine the rank of our hyper-heuristic among all other competitors. Table 5 provides 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 11. 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 evident in Table 3, we produce the best results (compared to the results announced after the CHeSC 2011 competition) for the first instances of the BP



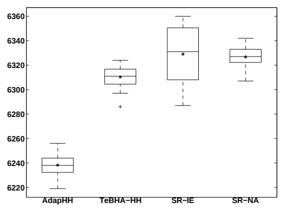
(a) SAT



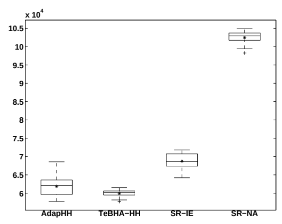
(b) BP



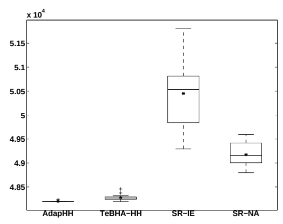
(c) PS



(d) FS



(e) VRP



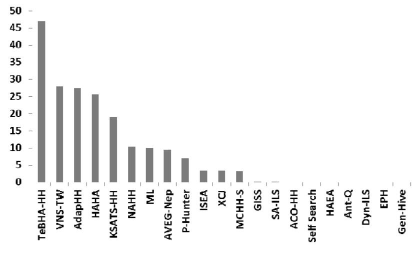
(f) TSP

Figure 10: 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.

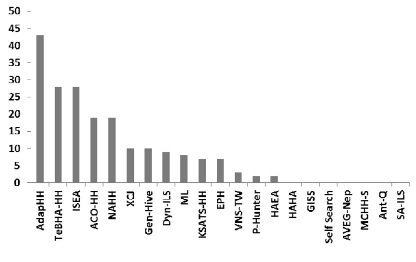
and VRP domains as well as the best result for the second instances of the TSP domain (the bold entries in Table 3).

### *5.4.2. An Analysis of TeBHA-HH*

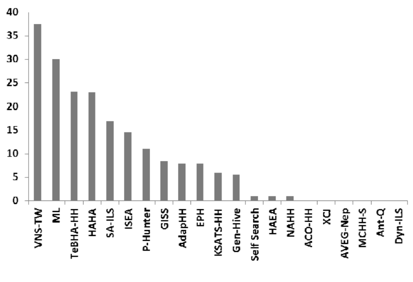
In this section, an analysis of the TeBHA-HH algorithm is performed to gain some insight into its behaviour during the search process. The objective value progress plots in almost all cases look the same, hence we have chosen an instance of the BP problem from CHeSC 2011 for which the TeBHA-HH demonstrates a good performance, while it consistently produces the same heuristic space partitioning in the tensor analysis stage. It is clear from Figure 5(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 12(a) shows how the average objective value changes (decreases) in time during the search process. We have divided



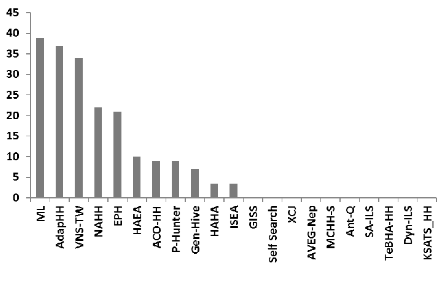
(a) Max-SAT



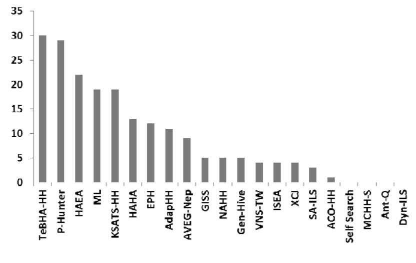
(b) BP



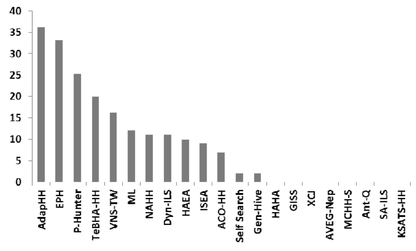
(c) PS



(d) FS



(e) VRP



(f) TSP

Figure 11: Ranking of the TeBHA-HH and hyper-heuristics which competed at CHeSC 2011 for each domain.

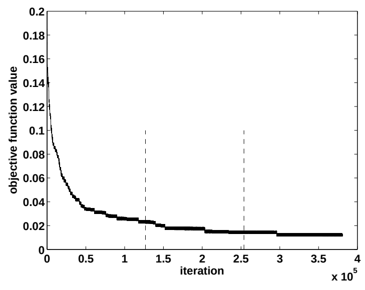
Table 5: Ranking of the TeBHA-HH among the selection hyper-heuristics that were competed in CHeSC 2011 with respect to their Formula 1 scores.

| 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 |

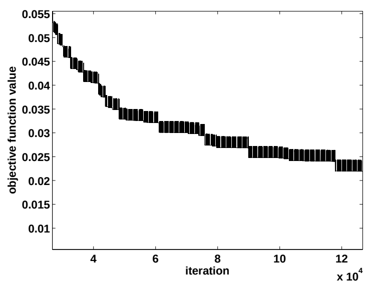
the progress of the algorithm into 3 distinct stages which represent the early, medium and late stages of the search, respectively (not to be confused with algorithm phases/stages, this is simply dividing the run-time into 3 periods). Figure 12(b) shows a close-up of the same plot achieved for a single run within the early stage. The dark rectangle shapes correspond to the times when naive 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 [53] where the knowledge encapsulates the heuristic indexes assigned to each acceptance mechanism. We have seen in Section 5.3 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.

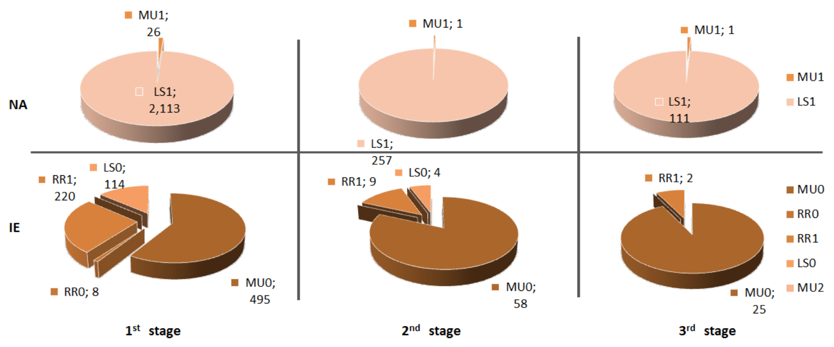
Furthermore, regardless of the type of acceptance mechanism, the algorithm manages to update the best values as is shown in Figure 12(c). The share of each heuristic in updating the best-so-far solution is also demonstrated in Figure 12(c). Interestingly, while the local search heuristic $LS1$ is responsible 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.



(a)



(b)



(c)

Figure 12: 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.

## 6. Conclusions

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 [12]. In this study, 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 [56], 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 hybridizing 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.

There is further space for future work. Similar to the work in [6], apprenticeship learning approach could be used in combination with tensor analysis to shed light on the mechanisms with which a given (hyper-)heuristic discovers the interaction between algorithm components. In addition to this, we will investigate into different ways of employing and exploiting tensors in heuristic search and optimization as well as improving the performance of the proposed framework even further.

## References

[1] E. Acar, S. Çamtepe, B. Yener, Collective sampling and analysis of high order tensors for chatroom communications, in: S. Mehrotra, D. Zeng, H. Chen, B. Thuraisingham, F.-Y. Wang (eds.), Intelligence and Security Informatics, vol. 3975 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 213–224.

[2] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, B. Yener, Modeling and multiway analysis of chatroom tensors, in: Proceedings of the 2005 IEEE International Conference on Intelligence and Security Informatics, ISI'05, Springer-Verlag, Berlin, Heidelberg, 2005.

[3] E. Acar, D. Dunlavy, T. Kolda, Link prediction on evolving data using matrix and tensor factorizations, in: IEEE International Conference on Data Mining Workshops, 2009. ICDMW '09., 2009.

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

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

[6] S. Asta, E. Özcan, A. J. Parkes, A. c. Etaner-Uyar, Generalizing hyper-heuristics via apprenticeship learning, in: Proceedings of the 13th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP'13, Springer-Verlag, Berlin, Heidelberg, 2013.

[7] M. Ayob, G. Kendall, A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine, in: PLACEMENT MACHINE, INTECH03 THAILAND, 2003.

[8] B. Bader, M. Berry, M. Browne, Discussion tracking in enron email using parafac, in: M. Berry, M. Castellanos (eds.), Survey of Text Mining II, Springer London, 2008, pp. 147–163.

[9] 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/`

[10] R. Bai, G. Kendall, An investigation of automated planograms using a simulated annealing based hyper-heuristic, in: T. Ibaraki, K. Nonobe, M. Yagiura (eds.), Metaheuristics: Progress as Real Problem Solvers, vol. 32 of Operations Research/Computer Science Interfaces Series, Springer US, 2005, pp. 87–108.

[11] E. K. Burke, 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.

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

[13] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. R. Woodward, A classification of hyper-heuristics approaches, in: M. Gendreau, J.-Y. Potvin (eds.), Handbook of Metaheuristics, vol. 57 of International Series in Operations Research & Management Science, 2nd ed., chap. 15, Springer, 2010, pp. 449–468.

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

[15] J. Carroll, J.-J. Chang, Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition, Psychometrika 35 (3) (1970) 283–319.

[16] K. Chakhlevitch, P. Cowling, Hyperheuristics: Recent developments, in: C. Cotta, M. Sevaux, K. Sörensen (eds.), Adaptive and Multilevel Metaheuristics, vol. 136 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2008, pp. 3–29.

[17] X. Chen, Y.-S. Ong, M.-H. Lim, K. C. Tan, A multi-facet survey on memetic computation, IEEE Transactions on Evolutionary Computation 15 (5) (2011) 591–607.

[18] E. C. Chi, T. G. Kolda, Making tensor factorizations robust to non-Gaussian noise, Tech. Rep. SAND2011-1877, Sandia National Laboratories, Albuquerque, NM and Livermore, CA (March 2011).

[19] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: E. Burke, W. Erben (eds.), Practice and Theory of Automated Timetabling III, vol. 2079 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, pp. 176–190.

[20] P. Cowling, G. Kendall, E. Soubeiga, A parameter-free hyperheuristic for scheduling a sales summit, in: Proceedings of the 4th Metaheuristic International Conference, MIC 2001, 2001.

[21] P. Cowling, G. Kendall, E. Soubeiga, Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation, in: S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, G. Raidl (eds.), Applications of Evolutionary Computing, vol. 2279 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 1–10.

[22] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, F. Paredes, Parameter tuning of a choice-function based hyperheuristic using parti-

cle swarm optimization, Expert Systems with Applications 40 (5) (2013) 1690 – 1695.

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

[24] X. Guo, X. Huang, L. Zhang, L. Zhang, Hyperspectral image noise reduction based on rank-1 tensor decomposition, {ISPRS} Journal of Photogrammetry and Remote Sensing 83 (0) (2013) 50 – 63.

[25] 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) (1970) 84.

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

[27] P.-C. Hsiao, T.-C. Chiang, L.-C. Fu, A vns-based hyper-heuristic with adaptive computational budget of local search, in: Evolutionary Computation (CEC), 2012 IEEE Congress on, 2012.

[28] A. Kheiri, E. Özcan, A hyper-heuristic with a round robin neighbourhood selection, in: M. Middendorf, C. Blum (eds.), Evolutionary Computation in Combinatorial Optimization, vol. 7832 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 1–12.

[29] A. Kheiri, E. Özcan, A. J. Parkes, A stochastic local search algorithm with adaptive acceptance for high-school timetabling, Annals of Operations Research.

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

[31] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.

[32] B. Krausz, C. Bauckhage, Action recognition in videos using nonnegative tensor factorization., in: ICPR, IEEE, 2010.

[33] M. Larose, A hyper-heuristic for the chesc 2011, in: LION6, 2011.

[34] L. D. Lathauwer, J. Castaing, Tensor-based techniques for the blind separation of dscdma signals, Signal Processing 87 (2) (2007) 322 – 336, tensor Signal Processing.

[35] L. D. Lathauwer, B. D. Moor, J. Vandewalle, A multilinear singular value decomposition, SIAM J. Matrix Anal. Appl. 21 (4) (2000) 1253–1278.

[36] P. K. Lehre, 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, ACM, New York, NY, USA, 2013.

[37] J. Li, L. Zhang, D. Tao, H. Sun, 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) (2009) 107–115.

[38] H. Loureno, O. Martin, T. Sttzle, in: M. Gendreau, J.-Y. Potvin (eds.), Handbook of Metaheuristics, vol. 146 of International Series in Operations Research & Management Science, Springer US, 2010, pp. 363–397.

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

[40] H. Lu, K. N. Plataniotis, A. N. Venetsanopoulos, A survey of multilinear subspace learning for tensor data, Pattern Recognition 44 (7) (2011) 1540 – 1551.

[41] M. Mısır, K. Verbeeck, P. De Causmaecker, G. V. Berghe, An intelligent hyper-heuristic framework for chesc 2011, in: Learning and Intelligent Optimization, Springer, 2012, pp. 461–466.

[42] D. Muti, S. Bourennane, Multidimensional filtering based on a tensor approach, Signal Processing 85 (12) (2005) 2338 – 2353.

[43] A. Nareyek, Choosing search heuristics by non-stationary reinforcement learning, Kluwer Academic Publishers, Norwell, MA, USA, 2004, pp. 523–544.

[44] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, E. Burke, Hyflex: A benchmark framework for cross-domain heuristic search, in: J.-K. Hao, M. Middendorf (eds.), European Conference on Evolutionary Computation in Combinatorial Optimisation, EvoCOP '12., vol. 7245 of LNCS, Springer, Heidelberg, 2012.

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

[46] E. Özcan, B. Bilgin, 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, X. Yao (eds.), Parallel Problem Solving from Nature - PPSN IX, vol. 4193 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 202–211.

[47] E. Özcan, M. Mısır, 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, IEEE, 2013.

[48] S. Rendle, L. Balby Marinho, A. Nanopoulos, 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, ACM, New York, NY, USA, 2009.

[49] P. Ross, Hyper-heuristics, in: E. K. Burke, G. Kendall (eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, chap. 17, Springer, 2005, pp. 529–556.

[50] A. Shashua, T. Hazan, Non-negative tensor factorization with applications to statistics and computer vision, in: ICML, 2005.

[51] R. Soto, B. Crawford, S. Misra, W. Palma, E. Monfroy, C. Castro, F. Paredes, Choice functions for Autonomous Search in Constraint Programming: GA vs PSO, Technical Gazette 20 (4) (2013) 621–629.
URL `http://hal.inria.fr/hal-00875551`

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

[53] J. Swan, J. Woodward, E. Özcan, G. Kendall, E. Burke, Searching the hyper-heuristic design space, Cognitive Computation (2013) 1–8.

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

[55] L. R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (1966c) 279–311.

[56] M. A. O. Vasilescu, D. Terzopoulos, Multilinear analysis of image ensembles: Tensorfaces., in: A. Heyden, G. Sparr, M. Nielsen, P. Johansen (eds.), ECCV (1), vol. 2350 of Lecture Notes in Computer Science, Springer, 2002.

[57] D. Wang, J. Zhou, K. He, C. Liu, J. Xia, Using tucker decomposition to compress color images, in: 2nd International Congress on Image and Signal Processing, 2009. CISP '09., 2009.

[58] Y. Wang, S. Gong, Tensor discriminant analysis for view-based object recognition., in: ICPR (3), IEEE Computer Society, 2006.

[59] D. Xu, S. Yan, L. Z. 0001, S. Lin, H.-J. Zhang, T. S. Huang, Reconstruction and recognition of tensor-based objects with concurrent subspaces analysis., IEEE Trans. Circuits Syst. Video Techn. 18 (1) (2008) 36–47.

[60] J. Ye, R. Janardan, Q. Li, Gpca: an efficient dimension reduction scheme for image compression and retrieval., in: W. Kim, R. Kohavi, J. Gehrke, W. DuMouchel (eds.), KDD, ACM, 2004.