A Tensor Analysis Improved Genetic Algorithm for Online Bin Packing

Shahriar Asta, Ender Özcan
ASAP Research Group
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, U.K.
{sba,exo}@cs.nott.ac.uk

ABSTRACT

Mutation in a Genetic Algorithm is the key variation operator adjusting the genetic diversity in a population throughout the evolutionary process. Often, a fixed mutation probability is used to perturb the value of a gene. In this study, we describe a novel data science approach to adaptively generate the mutation probability for each locus. The trail of high quality candidate solutions obtained during the search process is represented as a 3^{rd} order tensor. Factorizing that tensor captures the common pattern between those solutions, identifying the degree of mutation which is likely to yield improvement at each locus. An online bin packing problem is used as an initial case study to investigate the proposed approach for generating locus dependent mutation probabilities. The empirical results show that the tensor approach improves the performance of a standard Genetic Algorithm on almost all classes of instances, significantly.

Keywords

packing, data science, tensor, mutation

1. INTRODUCTION

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) [26] 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 [11] 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. [23] 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 [9], a mutation operator was designed based on knowledge capturing the distribution of candidate solutions in Extremal Optimization context. This method was successfully applied to PID tuning. The approach proposed in [35] 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 study, we use tensor analysis to generate mutation probabilities for each locus of a chromosome, also referred to as individual, representing a candidate solution. Tensorial techniques are powerful analysis methods for high dimensional data and are nowadays widely used in data mining and machine learning applications. In our approach, within a GA, the trail of high quality solutions, where each solution has a matrix form, is represented as a 3^{rd} order tensor. Factorizing 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. Our experiments show that tensor factorization achieves this objective and identifies genes which should have similar mutation likelihoods due to their close relationship.

An online bin-packing problem is used as a case study to analyse the performance of the proposed approach. The approach in [24] represents candidate solutions to a given packing problem as index policies (e.g., [14]). These policies have a matrix form in which an entry scores each potential packing option separately and the highest value option is selected. Due to this implicit two dimensional encoding of policies as candidate solutions, their trail during the evolutionary search process takes the form of a 3^{rd} order tensor. Hence, online bin packing forms a good playground for testing the proposed approach. The first results (reported in [24] and later in [4] and [36]) indicates that the GA approach finds high quality policies for the specific packing problems that perform significantly better than the generic 'human designed' heuristics, such as, first fit and best fit [27, 10]. In

GECCO '15, July 11 - 16, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754787

this paper, we show that by modifying mutation probabilities using tensor analysis, the performance of this framework can be improved significantly (when compared to the results achieved for our implementation of the GA approach in [24] as well as standard heuristics) on almost all instances.

The structure of the paper is as follows: Section 2 gives basic definitions of the bin-packing problem, the instances that we use, and the existing standard heuristics. In Section 3, index policies (policy matrices) are introduced and the original GA framework which evolves these policies is presented. Subsequently, Section 4 gives an introduction to tensorial representation and the factorization method used in this study. Section 5 describes the proposed approach which integrates tensor analysis into the GA approach. The setting used during the experiments as well as the experimental results are given in Section 6. Finally, Section 7 concludes the study and discusses future research directions.

2. ONLINE BIN PACKING PROBLEM

An online bin packing problem deals with the packing of items having various sizes which arrive one at a time. Each item has to be immediately packed into one of the open bins before the next item arrives and its size is revealed. This decision process is extremely difficult, since the future impact of the decisions made is extremely difficult to predict. Each bin has a fixed capacity of $C > 1$. The size of each item is chosen from a given range $[s_{min},\ s_{max}]$ where $s_{min} > 0$ and $s_{max} < C$. If an item is placed into an empty bin, a new bin is opened immediately, guaranteeing that there always exists a new empty bin. A bin is closed whenever its remaining capacity is too small to take in any new item.

A parameterized stochastic generator is used to produce uniform bin packing instances. Such a generator is represented by the formalism UBP (C, s_{min}, s_{max}, N) (adopted from [24]). In this formalism C is the bin capacity, s_{min} and s_{max} are minimum and maximum item sizes and N is the number of total items. As an example, $UBP(30, 4, 25, 10^5)$ is a random instance generator, producing a class of problem instances where each one is a sequence of 10^5 integer values. Each value represents the size of items and is randomly drawn from the range $[4, 25]$. It should be remembered that UBP's are instances of distributions. The actual sequence representing an instance varies depending on the seed given to the random number generator. That is, the generator is seeded with different seed values to generate a different sequence of items at each time used. This is indeed the case when our approach is tested as it will be seen in the coming sections.

Note that there are various instances available in the literature ([30] and [13]), however, these instances are devised for offline bin packing algorithms and usually consist of a small number of instances. That is the reason why we generate our own random instances. In this study, we always deal with instances for which the number of items is 10^5. Therefore, throughout the remainder of the paper, we use a shortened notation for UBP's, omitting the last term. For example the UBP in the above example is simply denoted by $UBP(30, 4, 25)$

There are well known 'human-designed' heuristics for online bin packing such as First Fit (FF), Best Fit (BF) and Worst Fit (WF) [17, 27, 10]. The FF heuristic assigns the item to the first open bin where an assignment is feasible. The BF heuristic places the current item into the bin with the least remaining space among all open bins whereas the WF heuristic assigns the item to the bin with the largest remaining space. Another well known heuristic for online bin packing is the harmonic based approach [22, 28]. Compared to other heuristics, harmonic based algorithms provide a better worst case performance ratio. However, the assumption is the items sizes are chosen from the range $(0, 1]$. The harmonic algorithm partitions the range into sub-intervals and each arriving item is assigned to its category depending on its size.

The performance of a bin packing solution can be evaluated based on various criteria. For example, one could use the number of bins used B for this purpose. B increases as larger number of items (N) are considered. Another performance evaluation criteria is the average generic fullness F_{gf} which provides an insight into the variation of resulting fullness between bins. Consider that the fullness of bin t is equal to f_t, $t \in \{1, \ldots, B\}$ then the average generic bin fullness is calculated as: $F_{gf} = 1/B \sum_t f_t^2$. In our study however, average fullness is considered as the fitness value of function evaluations. Average fullness, denoted by F_{af} is the value of the occupied space, averaged over the number of used bins and is calculated according to Eq.1 below.

$$F_{af} = 1/B \sum_t f_t \tag{1}$$

3. POLICY MATRIX REPRESENTATION

A packing policy/heuristic can be represented as a matrix, referred to as 'policy matrix'. In a policy matrix, each entry at the r^{th} row and s^{th} column, denoted as $W_{r,s}$ is a score. $W_{r,s}$ determines the priority of placing an item of size s in a bin with remaining capacity r. The values for scores $W_{r,s}$ are integer values and chosen from a range $[w_{min}, w_{max}]$. Given a policy matrix, one could simply scan the remaining bin capacity of existing (and feasible) open bins, assign each bin a score from the policy matrix, and then choose the bin with the highest score. The placement of an item in an open bin is *feasible* when the bin has sufficient free space to accommodate that item ($r \geq s$). During the packing process, an open empty bin is always considered as an option.

Some elements of the policy matrix do not require a placement strategy. For instance, cases were $s > r$ can never occur, i.e , a policy matrix always has a lower triangular structure. Thus, not all the elements in the policy matrix correspond to real cases and therefore no policy is required to handle such cases. The elements for which a handling policy is required are referred to as active entries whereas the rest of elements are called inactive entries. In other words, inactive entries correspond to pairs of item size and bin remaining capacity which either can never occur or are irrelevant. The active entries along columns of the policy matrix contain score values and represent a policy for a specific item size. Since the policy for a certain item size can be very different than that of other item size, the scores in each column are considered to be independent from each other.

In order to further clarify how a policy matrix functions, an example is given here. The policy matrix in Figure 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 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.

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

There is a growing interest on automating the design of heuristics (e.g. for some recent work see [29, 7]). In [24], 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 [25], 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 [4] 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 [5] an Apprenticeship Learning approach was proposed where an agent observes and learns the actions of high quality policy matrices during packing and generalizes its knowledge to unseen instances of various sizes exhibiting high performance particularly for larger instances. In [36], 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 study, we use the same GA with the same settings as described in [24], however, we present a tensor-based approach for improving its performance via an adaptive locus based mutation operator, instead of using a generic one.

4. TENSOR ANALYSIS

There is a plethora of applications and research areas benefiting from tensorial representation and tensor analytic approaches. Tensor tools have a great record of contribution to many research fields, such as, computer vision [33], video processing [20], data compression [34], web mining [1] and etc. Generally speaking, many problems produce data which are high dimensional in nature. For instance, video streams constitute a data which is three dimensional (pixel coordinates and time) or higher (when information such as audio, text, change of environment and etc are also considered). The traditional approach in data mining has been to collapse this data into a two dimensional dataset in order to apply machine learning techniques and perform tasks such as classification and prediction. However, recent research ([33],[2] and [1]) shows that preserving the natural dimensionality of the data is useful in that it keeps the information regarding latent relationship between variables along different dimensions. Tensor representation fulfils this requirement as tensors are multidimensional arrays and can represent data with high dimensions. The *order* of a tensor indicates its dimensionality where each dimension of a tensor is referred to as a *mode*. The interested readers can refer to a comprehensive survey on tensors and their applications in [19] for further information. The method widely used to extract the latent relationship between various modes of data involves in 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 generalizations of the Singular Value Decomposition (SVD) to higher dimensions. Higher Order SVD (HOSVD) [21], Tucker decomposition [32], Parallel Factor (a.k.a PARAFAC or CANDECOMP or CP) [16] and Non-negative Tensor Factorization (NTF) [31] are among numerous factorization methods proposed by researchers. In this paper, we use CP factorization ([3]) as described in the following subsection with the goal of enabling an adaptive locus based mutation operator within a GA for online bin packing.

4.1 CP Factorization

We use, boldface Euler script letters, boldface capital letters and boldface lower-case letters to denote tensors (e.g., $\mathcal{T}$), matrices (e.g., $\mathbf{M}$) and vectors (e.g., $\mathbf{v}$), respectively. Scalar values (e.g., tensor, matrix and vector entries) are indexed by lower-case letters. For instance, t_{pqr} is the (p, q, r) entry of a $3^{rd}-$order (three dimensional) tensor $\mathcal{T}$.

CP decomposition uses the Alternating Least Square (ALS) algorithm ([8],[15]) to factorize (decompose) a tensor. Fac-

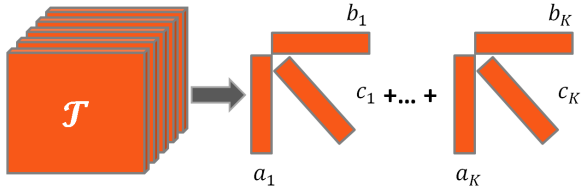


Figure 2: Factorizing a tensor to K components.

torizing a 3^{rd} order tensor[1] $\mathcal{T}$ of size $P \times Q \times R$ using the CP factorization results in another tensor $\hat{\mathcal{T}}$ which approximates $\mathcal{T}$ as in Equation 2. This process is also illustrated in Figure 2.

$$\hat{\mathcal{T}} = \sum_{k=1}^{K} \lambda_k \ \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k \tag{2}$$

In Eq.2, $\lambda_k \in \mathbb{R}_+$, $\mathbf{a}_k \in \mathbb{R}^P$, $\mathbf{b}_k \in \mathbb{R}^Q$ and $\mathbf{c}_k \in \mathbb{R}^R$ for $k = 1 \cdots K$, where K is the number of desired components. A component is expressed as $(\lambda_k \ \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k)$ and the resulting vectors (e.g., $\mathbf{a}_k$, $\mathbf{b}_k$ and $\mathbf{c}_k$) are called basic factors. λ_k is the weight of the kth component. Note that "$\circ$" is the outer product operator. The outer product of three vectors produces a 3^{rd}-order tensor. For instance, the product $\lambda \ \mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_1$ results in a 3^{rd}-order tensor where each tensor entry, indexed as pqr is computed through a simple multiplication like $a_p b_q c_r$. In a special case, the outer product of two vectors $\mathbf{a}_k$ and $\mathbf{b}_k$ produces a matrix $\mathbf{B}_k$ of size $P \times Q$ for each component k

$$\mathbf{B}_k = \mathbf{a}_k \circ \mathbf{b}_k \tag{3}$$

The matrix $\mathbf{B}_k$ will be referred to as the *Basic Frame* throughout the paper. The basic frame $\mathbf{B}_k$ quantifies the relationship between the object pairs across the first two dimensions in each component. In other words, this quantity indicates the "level of interaction" between pairs of object, each from a separate tensor dimension, in component k [19].

The ALS algorithm approximates the original tensor $\mathcal{T}$ by minimizing the error difference between $\mathcal{T}$ and the estimation ($\hat{\mathcal{T}}$). This error is denoted as ε and calculated as in Eq.4 (the subscript F refers to the Frobenius norm as in Eq.5).

$$\varepsilon = \frac{1}{2} ||\mathcal{T} - \hat{\mathcal{T}}||_F^2 \tag{4}$$

$$||\mathcal{T} - \hat{\mathcal{T}}||_F^2 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \left(t_{pqr} - \hat{t}_{pqr}\right)^2 \tag{5}$$

The approximated tensor is, to some extent, a generalized representation of the original tensor and immune to data anomalies, such as, missing data. Basic factors produced by the factorization method can be used to represent the high dimensional data in a compressed manner. The tensor factorization exhibit other interesting characteristics. It enables the detection of more comprehensible sub-space from the original data. This can be very useful in many applications. In computer vision, for instance, [18] and [20] separately showed that factorization of a 3^{rd}-order tensor of a video sequence results in basic frames which reveal the location and functionality of synchronously moving human body parts. Evolutionary algorithms are among many approaches which produce high dimensional data. This possibility was first acknowledged in [3] where the trail of a hyper-heuristic search algorithm was represented as a tensor. The space of heuristics where the hyper-heuristic conducts the search was then partitioned using the basic factors obtained from factorizing the tensor. The proposed tensor based approach achieves impressive results despite its simplicity. 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 [3]. 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.

[1]In this study we only focus on 3^{rd} order tensors.

5. PROPOSED APPROACH

In the original framework [24], 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 (Eq.1) as feedback. The cycle of evolution continues until the stopping criterion is met. Our method modifies the training procedure as illustrated in Figure 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 is 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.

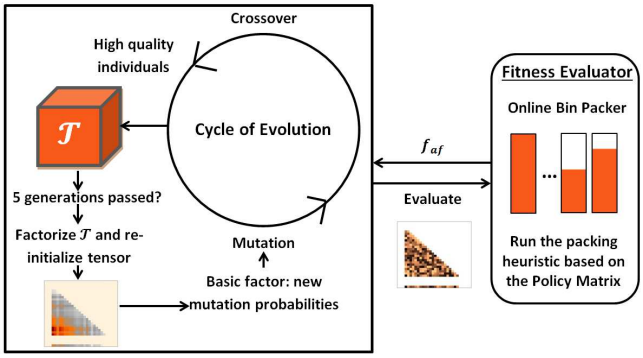


Figure 3: The GA+TA framework

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 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 is set to 1. That is, the original tensor $\mathcal{T}$ is approximated by $\hat{\mathcal{T}}$ as the following

$$\hat{\mathcal{T}} = \lambda \, \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \tag{6}$$

where the length of the vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are C, C and R respectively. As depicted in Figure 4, the outer product of vectors $\mathbf{a}$ and $\mathbf{b}$ results in a basic frame $\mathbf{B}$ which is exactly the shape of a policy matrix with the size $C \times C$ (Eq.3 is used with $k = 1$ to produce $\mathbf{B}$). The difference between $\mathbf{B}$ 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.

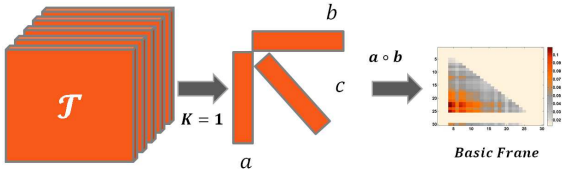


Figure 4: Extracting the basic frame for $K = 1$ in Eq.2.

Thus, the values in $\mathbf{B}$ 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 $\mathbf{B}$ has been applied.

6. EXPERIMENTS

In this section, after providing the details of our experimental setting, we discuss the results. In order to to gain a better understanding of our approach, the basic frames achieved from our experiments are investigated first. Subsequently, we compare our approach to the original framework by conducting a comparative study between the performance of the two algorithms on ten different classes of instances.

6.1 Experimental Setting

The setting used for the GA framework is adopted from [24]. That is, for each training phase, the number of generations is set to 200 while the population size is $\lceil C/2 \rceil$. Tournament selection is the strategy of choice for parent selection and the tour size is 2. Crossover and mutation operators are uniform and traditional respectively and the crossover probability is equal to 1.0. As discussed in Section 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 1, $w_{max} = 7$). For tensor operations, Matlab Tensor Toolbox [6] 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 paper is compared to the original GA framework. Throughout the experiments, our approach is referred to by the GA+TA whereas the original work in [24] is referred to by GA. 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.

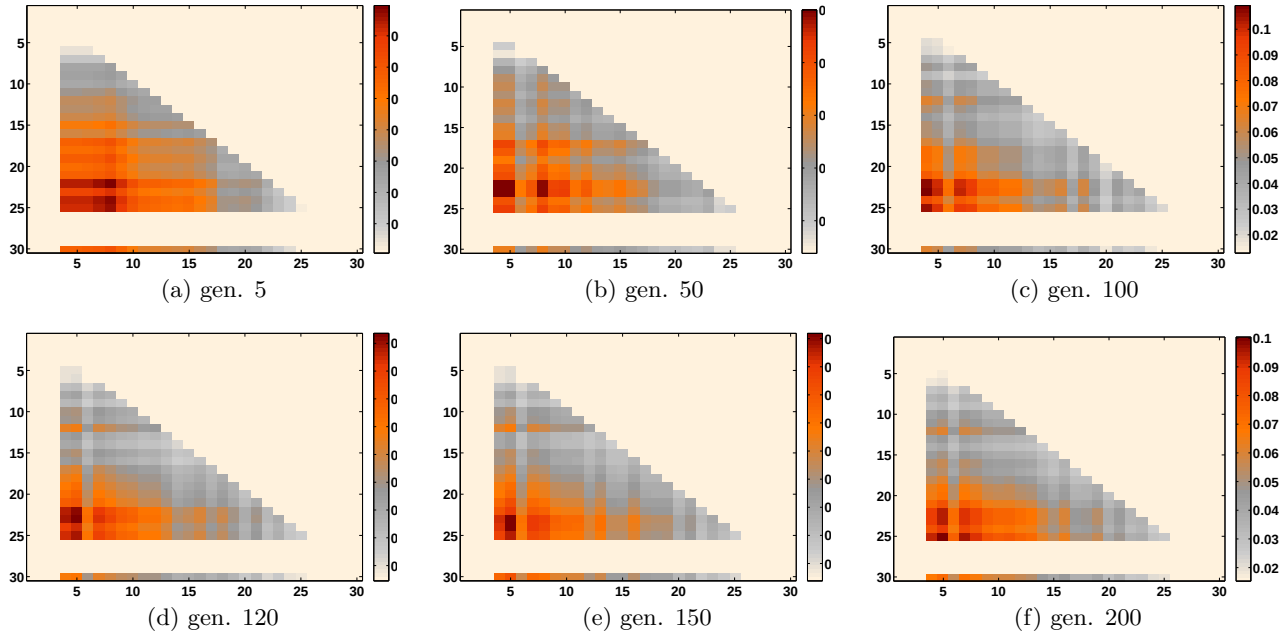
6.2 Basic Frames: An Analysis

As discussed in Section 5, the GA+TA algorithm frequently constructs and factorizes a tensor of high quality candidate solutions. The factorization 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 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 is $UBP(30, 4, 25)$. The basic frame generated after the first factorization (Figure 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 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 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(d).

A closer look at Figure 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.6, the factor $\mathbf{a}$ captures the gene patterns corresponding to bin remaining capacity while $\mathbf{b}$ does the same for gene patterns concerning the item size. The factor $\mathbf{c}$ 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 $\mathbf{a}$ and $\mathbf{b}$, combining the gene patterns related to each dimension of the tensor. It has been observed in many studies (such as [12] and [20]) 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



(a) gen. 5 (b) gen. 50 (c) gen. 100

(d) gen. 120 (e) gen. 150 (f) gen. 200

Figure 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(d) is the probability matrix using which the best policy is achieved (gen 121)

regions with similar score values [2] in the basic frame (as it is visible in both figures 5 and 6), the genes are considered to be *related*.

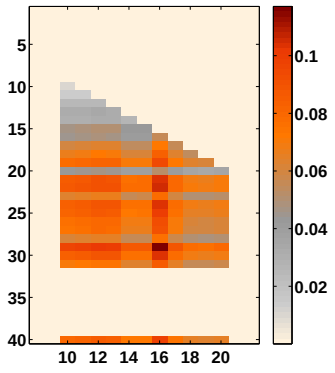


Figure 6: The basic frame for $UBP(40, 10, 20)$ using which the best policy matrix was found.

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 7 is generated using the probabilities in Figure 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

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

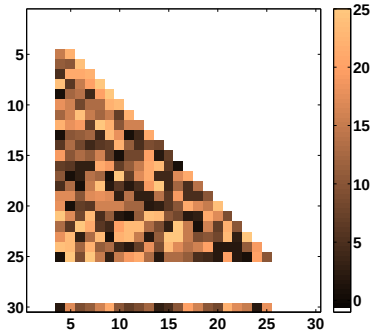


Figure 7: The best policy matrix obtained by GA+TA for $UBP(30, 4, 25)$ using the basic frame entries (see Figure 5(d)) as mutation probabilities.

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.

6.3 Experimental Results

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 1 summarizes the results. The only instance generator on 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 per-

formance 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 factorization algorithms are designed for high dimensional data where it is expected that various dimensions of data are correlated. Perhaps the study in [20] 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 factorization algorithm. This is something which couldn't be achieved if the policies were vectorized (as in [4]). 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 [3] introduced the use of these approaches in heuristic research for the first time. The impressive results achieved in this study confirms this and is encouragement for further research in transferring tensor analytic approaches to the field of (meta)heuristic optimization.

7. CONCLUSION

In this study, a powerful data mining tool, namely, tensor analysis is integrated into a GA framework [24] 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. The empirical 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. The 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.

8. REFERENCES

[1] E. Acar, D. Dunlavy, and T. 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.

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

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

[4] 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.

[5] S. Asta, E. Özcan, A. J. Parkes, and A. S. Etaner-Uyar. Generalizing hyper-heuristics via apprenticeship learning. In M. Middendorf and C. Blum, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7832 of *Lecture Notes in Computer Science*, pages 169–178. Springer Berlin Heidelberg, 2013.

[6] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.5. Available online, January 2012.

[7] E. K. 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.

[8] J. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of âĂIJeckart-youngâĂI decomposition. *Psychometrika*, 35(3):283–319, 1970.

[9] 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.

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

[11] 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.

[12] 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, Feb. 2011.

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

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

[15] 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.

[16] 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.

[17] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds

Table 1: Performance comparison of the GA+TA, GA and BF algorithms for each UBP over 100 trials. The values *avg*, *min* and *max* indicate the mean, minimum and maximum bin fullness in persentages, respectively. The 'vs' column in middle highlights the results of the Wilcoxon sign rank test where $>$ ($<$) means that GA+TA is significantly better (worse) than GA within a confidence interval of 95%. Similarly, $\geq$ shows that GA+TA performs slightly better than GA (with no statistical significance). The sign = refers to equal performance.

Instance	GA+TA				GA			BF		
	avg	min	max	vs	avg	min	max	avg	min	max
UBP(6, 2, 3)	**99.99**	99.99	100	=	99.99	99.99	100	92.29	92.26	92.31
UBP(15, 5, 10)	**99.62**	99.28	99.88	$\geq$	99.58	99.16	99.86	99.62	99.15	99.89
UBP(20, 5, 10)	**98.28**	98.19	98.34	$>$	97.89	97.68	98	91.55	91.45	91.62
UBP(30, 4, 20)	**99.53**	99.41	99.61	$>$	99.09	98.95	99.23	96.84	96.80	96.90
UBP(30, 4, 25)	**99.53**	99.23	99.7	$>$	98.39	98.06	98.57	98.38	98.31	98.46
UBP(40, 10, 20)	**96.27**	95.99	96.39	$>$	96.08	95.86	96.35	90.23	90.12	90.31
UBP(60, 15, 25)	99.47	99.14	99.77	$<$	**99.68**	99.1	99.91	92.55	92.43	92.66
UBP(75, 10, 50)	**98.53**	98.4	98.6	$>$	98.27	98.12	98.36	96.08	96.04	96.13
UBP(80, 10, 50)	**98.66**	98.58	98.72	$>$	98.07	97.98	98.13	96.39	96.32	96.44
UBP(150, 20, 100)	**98.22**	98.12	98.32	$>$	97.78	97.68	97.86	95.82	95.77	95.87

for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[18] 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, Aug. 2009.

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

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

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

[22] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[23] 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.

[24] 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.

[25] 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.

[26] A. Prakash, F. T. Chan, and S. Deshmukh. Fms scheduling with knowledge based genetic algorithm approach. *Expert Systems with Applications*, 38(4):3161 – 3171, 2011.

[27] 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.

[28] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203 – 227, 1991.

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

[30] 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.

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

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

[33] M. A. O. Vasilescu and D. 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.

[34] D. 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.

[35] G. Yan, G. Xie, Z. Chen, and K. Xie. Knowledge-based genetic algorithms. In G. Wang, T. Li, J. 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.

[36] A. Yarimcam, S. Asta, E. Özcan, and A. J. Parkes. Heuristic generation via parameter tuning for online bin packing. In *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on*, pages 102–108, Dec 2014.