Random Keys on $\mathbb{ICE}$: Marginal Product Factorized Probability Distributions in Permutation Optimization

Peter A.N. Bosman
Peter.Bosman@cs.uu.nl

Dirk Thierens
Dirk.Thierens@cs.uu.nl

Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

December 2002

Abstract

In this paper, we discuss multivariately factorized probability distributions for permutation random variables and a greedy approach to estimating these probability distributions from data. We use the representation known as random keys for permutations. The major benefit of using random keys is that no infeasible solution can be generated if crossover is applied in an evolutionary algorithm (EA). The estimated multivariately factorized probability distribution can be used to construct a linkage friendly crossover operator with which new offspring can be generated. We call the EA that uses this technique to construct a crossover operator, $\mathbb{ICE}$. This technical report specifically presents the details of estimating multivariately factorized probability distributions for permutations using the random keys representation. As such, this paper is a extension of an earlier publication in which experiments with an EA that follows this approach have been reported as well [7].

1 Outline

This paper is organized as follows. In section 2 we discuss permutations and the random keys encoding of permutations. In section 3 we then define multivariately factorized probability distributions for permutation random variables and discuss a greedy approach to estimating such a probability distribution from data. Although this paper is not intended to be a fully self–contained EA paper in which the described algorithmics are brought into practice using a test–suite and new EAs (see [7] for a presentation of such results), we finish this paper with a brief presentation of the type of EAs in which the estimation of probability distributions plays a key role and specifically the $\mathbb{ICE}$ algorithm in which multivariately factorized probability distributions are used to construct a linkage friendly crossover operator.

2 Permutations and random keys

A permutation of length l is a vector of l unique objects. For simplicity, we assume that each of these objects is an integer. Moreover, we say that a permutation of length l is *compact* if it is a permutation of $(0, 1, \ldots, l-1)$.

For solving permutation optimization problems with EAs by using compact permutations as a genotype, the application of classical crossover operators, such as one–point crossover, is useless. The reason for this is that classical crossover operators can construct integer sequences that do not represent a permutation. For instance, if the parent genotypes are $(0, 1, 2, 3)$ and $(3, 2, 1, 0)$, an application of one–point crossover with a crossover point in the middle may result in two offspring genotypes $(0, 1, 1, 0)$ and $(3, 2, 2, 3)$, both of which are not permutations.

To ensure feasibility of the offspring, specialized recombination and mutation operators have been designed [8, 10, 15, 21] that ensure that the offspring are always permutations. Alternatively, a different encoding of permutations can be used such that classical crossover operators

can straightforwardly be applied to this encoding. The most prominent and successful example of such an encoding is the *random keys* encoding, which was proposed by Bean [2].

The random keys encoding of a permutation of length l is a string of l real values. The decoding function that is associated with mapping a random keys genotype $\boldsymbol{r}$ to an integer permutation $\boldsymbol{p}$ is such that integer i occurs before integer j in $\boldsymbol{p}$ if and only if $\boldsymbol{r}_i < \boldsymbol{r}_j$.

In practice, each real value in the random keys genotype is usually defined to be in $[0, 1]$ instead of in the space of all possible real values. Since there are no additional restrictions on the random keys genotype, the main advantage of the random keys genotype is that by crossing over the the real values, no genotype can be constructed that doesn't represent a permutation.

A gene at locus i in a random keys genotype $\boldsymbol{r}$ is a random key $\boldsymbol{r}_i$. From the definition of the associated decoding function it follows that the value of a random key $\boldsymbol{r}_i$ determines the *position* of the integer i in the decoded permutation $\boldsymbol{p}$. To actually decode a random keys genotype $\boldsymbol{r}$ into an integer permutation $\boldsymbol{p}$, the random keys can be sorted in ascending order in $\mathcal{O}(|\boldsymbol{r}|\log(|\boldsymbol{r}|))$ time. We denote this by by $\boldsymbol{p} = \sigma(\boldsymbol{r})$, where σ denotes the ascending sorting function such that $\boldsymbol{r}_{\boldsymbol{p}_0} < \boldsymbol{r}_{\boldsymbol{p}_1} < \ldots < \boldsymbol{r}_{\boldsymbol{p}_{|\boldsymbol{r}|-1}}$. As an example, we have that $\sigma((0.61, 0.51, 0.62, 0.31)) = (3, 1, 0, 2)$. From this example we can for instance see that random key $\boldsymbol{r}_2$, which is assigned the value 0.62 in this example, determines that the integer 2 in the integer permutation is placed at the end of the permutation because 0.62 is the largest value in the random keys genotype.

3 Multivariate factorized probability distributions for permutations based on random keys

Consider l random variables $\boldsymbol{\mathcal{Z}} = (Z_0, Z_1, \ldots Z_{l-1})$. A multivariately factorized probability distribution (also called multivariate factorization, marginal factorization or marginal product factorization) for random variables $\boldsymbol{\mathcal{Z}}$ is a product of multivariate joint *generalized probability density functions* (gpdfs), with the gpdfs defined over mutually exclusive vectors of variables. Each random variable thus occurs in a single gpdf. The vector of variables of a gpdf is called a *node vector*, denoted by $\boldsymbol{\nu}_i$. We call the vector of all node vectors the *node partition vector* and denote it by $\boldsymbol{\nu}$. A multivariately factorized probability distribution is defined by:

$$P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Z}})(\boldsymbol{z}) = \frac{1}{|values(\boldsymbol{\mathcal{Z}})|} \prod_{i=0}^{|\boldsymbol{\nu}|-1} |values(Z_{\boldsymbol{\nu}_i})| P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})) \tag{1}$$

$$= \frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |values(Z_{\boldsymbol{\nu}_i})|}{|values(\boldsymbol{\mathcal{Z}})|} \prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i}))$$

In equation 1, $\boldsymbol{\theta}^{\boldsymbol{\nu}_i}$ denotes the parameters for the multivariate joint gpdf $P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})$, $values(Z_{\boldsymbol{a}})$ is the set of all possible values that can be assigned to random variables $Z_{\boldsymbol{a}}$ and $\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})$ is a function that reformats a subvector of a value for all random variables into a value for a selection of all random variables. Note that the random variables indicated by a node vector are independent of all other random variables in a multivariate factorization.

If the random variables define a Cartesian product value space, the definition of a multivariate factorization simplifies significantly. For binary random variables for instance, $values(Z_{\boldsymbol{a}}) = 2^{|\boldsymbol{a}|}$ and thus $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} values(Z_{\boldsymbol{\nu}_i})}{values(\boldsymbol{\mathcal{Z}})} = 1$. For real–valued random variables, $values(Z_{\boldsymbol{a}}) = |\mathbb{R}|^{|\boldsymbol{a}|}$, and we again find that the definition of a multivariate factorization simplifies to $\prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i}))$. For permutation random variables however, such a simplification does not occur and the factor $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |values(Z_{\boldsymbol{\nu}_i})|}{|values(\boldsymbol{\mathcal{Z}})|}$ serves as a normalization of the product of the gpdfs such that the resulting function over all random variables is a probability distribution. Function $\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})$ is simply the identity function for Cartesian product value spaces. However, for non–Cartesian product spaces, such as the space of permutations, the number of values is smaller for a subset of all random

variables than for all random variables. To make sure that a subvector of a value for a larger set of random variables is a value for the subset of all random variables, some type of transformation must be made. In the remainder of this paper, we will only focus on the case in which the random variables Z_i are permutation random variables, which we will denote by O_i.

3.1 Interpretation of permutation random variables

By decoding a random keys substring $\boldsymbol{r}_{\boldsymbol{a}}$, an integer permutation is obtained that indicates the relative ordering of the integers $\boldsymbol{a}$ in the complete decoded random keys genotype $\sigma(\boldsymbol{r})$. We let permutation random variable O_i represent the *position* of integer i in the integer permutation that we ultimately use as a solution to the optimization problem. Now, $\Pr(O_i < O_j)$ denotes the probability that integer i precedes integer j in a solution. Note that this interpretation for a permutation random variable O_i is valid since the positions of all integers $i \in \{0, 1, \ldots, l-1\}$ themselves form again a permutation. It should furthermore be noted that with these semantics, a permutation random variable O_i can directly be identified with a random keys random variable R_i since by definition of the random keys genotype we have that $\Pr(O_i < O_j) = \Pr(R_i < R_j)$. Because of the correspondence between random keys and the permutation random variables, we may continue to describe probability distributions over permutation random variables $O_{\boldsymbol{a}}$ instead of the random keys random variables $R_{\boldsymbol{a}}$ in the remainder of this chapter.

3.2 Permutation gpdf

In this section we define a gpdf for permutation random variables and show how we can estimate it with a maximum likelihood from a given sample vector of random keys encodings. According to the proposed interpretation of permutation random variables, the type of dependency that will be modelled, will concern the relative ordering of the integers in the actual permutations.

3.2.1 Definition

Let $\mathit{perm}(\boldsymbol{o})$ be the set that contains all possible permutations of $\boldsymbol{o}$. A vector of permutation random variables $O_{\boldsymbol{a}}$ can be assigned $|\boldsymbol{a}|!$ different values, that is $\mathit{values}(O_{\boldsymbol{a}}) = |\boldsymbol{a}|!$, since there are $|\boldsymbol{a}|!$ ways to permute $O_{\boldsymbol{a}}$ and obtain the only valid ways of querying a probability distribution over permutations. If $\boldsymbol{a} = 3$ for instance, we have 6 ways to query the probability distribution:

$$\Pr(O_{\boldsymbol{a}_0} < O_{\boldsymbol{a}_1} < O_{\boldsymbol{a}_2}), \quad \Pr(O_{\boldsymbol{a}_0} < O_{\boldsymbol{a}_2} < O_{\boldsymbol{a}_1}), \quad \Pr(O_{\boldsymbol{a}_1} < O_{\boldsymbol{a}_0} < O_{\boldsymbol{a}_2}),$$
$$\Pr(O_{\boldsymbol{a}_1} < O_{\boldsymbol{a}_2} < O_{\boldsymbol{a}_0}), \quad \Pr(O_{\boldsymbol{a}_2} < O_{\boldsymbol{a}_0} < O_{\boldsymbol{a}_1}), \quad \Pr(O_{\boldsymbol{a}_2} < O_{\boldsymbol{a}_1} < O_{\boldsymbol{a}_0}),$$

To formalize this in terms of values that can be assigned to random variables $O_{\boldsymbol{a}}$ we can thus use exactly these permutations. To ensure that indeed only $|\boldsymbol{a}|!$ values are allowed for random variables $O_{\boldsymbol{a}}$, we constrain the values to be compact permutations (e.g. permutations of $(0, 1, \ldots, |\boldsymbol{a}| - 1)$).

If we now write $P(O_{\boldsymbol{a}} = \boldsymbol{o})$, this means we are querying the probability distribution for the event in which the relative ordering of the random variables is indicated by $\boldsymbol{o}$. In other words, if and only if $\boldsymbol{o}_i < \boldsymbol{o}_j$ for some $i, j \in \{0, 1, \ldots, |\boldsymbol{a}| - 1\}, i \neq j$, then $O_{\boldsymbol{a}_i} < O_{\boldsymbol{a}_j}$ holds in the query. This means that we can sort the compact permutation and order the random variables according to the ascending order to get the full ordering of the random variables that corresponds to the event that we are interested in:

$$P(O_{\boldsymbol{a}} = \boldsymbol{o}) \quad = \quad P(O_{\boldsymbol{a}})(\boldsymbol{o}) \quad = \quad P((O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_0} < (O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_1} < \ldots < (O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_{|\boldsymbol{a}|-1}}) \tag{2}$$

The permutation gpdf for random variables $O_{\boldsymbol{a}}$ can thus be characterized as having $|\boldsymbol{a}|!$ parameters that indicate the probability of a certain relative ordering $\boldsymbol{o} \in \mathit{perm}(0, 1, \ldots, |\boldsymbol{a}| - 1)$ of the random variables $O_{\boldsymbol{a}}$.

3.2.2 Parameter estimation

It is relatively easy to estimate the parameters for the permutation gpdf with a maximum likelihood. Let $\boldsymbol{\mathcal{S}}$ be a vector of random keys genotypes of length l for which our parameter estimates must be made. Since we have a discrete space with a finite number of parameters, a straightforward way to estimate the probability $P(O_{\boldsymbol{a}} = \boldsymbol{o})$ is to compute the average frequency of finding random keys in the data at the positions indicated by $\boldsymbol{a}$ such that they encode the same ordering of random variables $O_{\boldsymbol{a}}$ as does $\boldsymbol{o}$. Such proportion estimates for discrete gpdfs are known to result in a maximum likelihood estimate [1, 20]. If we now realize that $\sigma(\sigma(\boldsymbol{o})) = \boldsymbol{o}$ if $\boldsymbol{o}$ is a compact permutation, we obtain the following characterization of the maximum likelihood estimation of the permutation gpdf:

$$
\begin{aligned}
&\hat{P}(O_{\boldsymbol{a}} = \boldsymbol{o}) \qquad (3)\\
&= \\
&\hat{P}((O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_0} < (O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_1} < \ldots < (O_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_{|\boldsymbol{a}|-1}}) \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } ((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_0} < ((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_1} < \ldots < ((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}})_{\sigma(\boldsymbol{o})_{|\boldsymbol{a}|-1}} \\ 0 & \textit{otherwise} \end{cases} \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}}) = \sigma(\boldsymbol{o}) \\ 0 & \textit{otherwise} \end{cases} \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}})) = \boldsymbol{o} \\ 0 & \textit{otherwise} \end{cases}
\end{aligned}
$$

3.2.3 Sampling

Ultimately, the estimated multivariately factorized probability distribution will be used to construct a crossover operator with which new offspring are generated in an EA. Although we shall thus not need to draw new samples from the estimated probability distribution, we briefly indicate how this could be done for the sake of completeness.

Because we are using random keys instead of compact integer permutations, we can generate new samples from each estimated gpdf independently from the others. To generate a random keys sequence by sampling an estimated gpdf for random variables $O_{\boldsymbol{a}}$, first a new integer permutation $\boldsymbol{o}$ can be drawn from the estimated permutation gpdf for random variables $O_{\boldsymbol{a}}$. Subsequently, a vector $\boldsymbol{r}$ must be generated of $|\boldsymbol{a}|$ random numbers in $[0, 1]$. These random numbers must then be ordered such that $\sigma(\sigma(\boldsymbol{r})) = \boldsymbol{o}$, or equivalently, since $\boldsymbol{o}$ is a compact permutation, such that $\sigma(\boldsymbol{r}) = \sigma(\boldsymbol{o})$. This can be achieved by placing real value $\boldsymbol{r}_{\sigma(\boldsymbol{r})_i}$ at position $\sigma(\boldsymbol{o})_i$ in the final random keys vector.

3.3 Multivariate permutation factorization

In this section we define the multivariate permutation factorization and discuss representation issues and a greedy factorization selection algorithm to estimate multivariate permutation factorizations from data.

3.3.1 Definition

Defining the multivariately factorized probability distribution for discrete integer permutation random variables is not as straightforward as is the case for discrete integer representations. To get some intuition as to why this is so, we first present an example.

Consider the case in which $l = 6$ and we want to define the multivariate factorization based on $\boldsymbol{\nu} = ((0,1),(2,3,4,5))$. To this end, we first observe an example query for obtaining the probability of a permutation $\boldsymbol{o} = (0,5,3,4,2,1)$. In other words, we want to know how to compute $\hat{P}_{((0,1),(2,3,4,5))}(O_0,O_1,O_2,O_3,O_4,O_5)(0,5,3,4,2,1)$. Note that this means that we are interested in the probability of the event that $O_0 < O_5 < O_4 < O_2 < O_3 < O_1$.

According to the definition of the multivariate factorization, we must be able to compute the required probability by using the permutation gpdfs $\hat{P}(O_0,O_1)$ and $\hat{P}(O_2,O_3,O_4,O_5)$. However, to do so, it is important to note that we cannot simply split the query vector $\boldsymbol{o}$ up into $(\boldsymbol{o}_0,\boldsymbol{o}_1)$ and $(\boldsymbol{o}_2,\boldsymbol{o}_3,\boldsymbol{o}_4,\boldsymbol{o}_5)$ respectively, because these subpermutations may no longer be compact. To remedy this, we must thus find alternative permutations such that these alternative permutations are compact, but also such that the relative ordering is the same. According to the event indicated by $\boldsymbol{o}$, the query vector $\boldsymbol{o^0}$ for $\hat{P}(O_0,O_1)$ must represent the event that $O_0 < O_1$, since $\boldsymbol{o}_0 < \boldsymbol{o}_1$. Similarly, according to the event indicated by $\boldsymbol{o}$, the query vector $\boldsymbol{o^1}$ for $\hat{P}(O_2,O_3,O_4,O_5)$ must represent that $O_5 < O_4 < O_2 < O_3$, since $\boldsymbol{o}_5 < \boldsymbol{o}_4 < \boldsymbol{o}_2 < \boldsymbol{o}_3$. Using the definition of of permutation random variables, this means that the correct compact alternative permutations are $\boldsymbol{o^0} = (0,1)$ and $\boldsymbol{o^1} = (2,3,1,0)$. To derive these permutations from the original query vector $\boldsymbol{o}$ we first take subvectors $\boldsymbol{o}_{(0,1)} = (0,5)$ and $\boldsymbol{o}_{(2,3,4,5)} = (3,4,2,1)$, since they validly indicate the relative ordering of the random variables. If we now sort these permutations, we get compact permutations, e.g. $\sigma(\boldsymbol{o}_{(0,1)}) = (0,1)$ and $\sigma(\boldsymbol{o}_{(2,3,4,5)}) = (3,2,0,1)$. Although these permutations are compact, they no longer represent the same relative ordering of the random variables. However, they do indicate how such an ordering can be constructed, since for any permutation $\boldsymbol{p}$, the sorting operation $\sigma(\boldsymbol{p})$ tells for each rank i in the sorted ordering, the position $\sigma(\boldsymbol{p})_i$ at which to find the object of rank i in $\boldsymbol{p}$. Thus, if we place the integers $(0,1,\ldots,|\boldsymbol{p}|-1)$ in a permutation $\boldsymbol{p'}$ in such a way that $\sigma(\boldsymbol{p'}) = \sigma(\boldsymbol{p})$, $\boldsymbol{p'}$ is a compact alternative to $\boldsymbol{p}$. Since $\boldsymbol{p'}$ is compact, we have $\sigma(\sigma(\boldsymbol{p'})) = \boldsymbol{p'}$, and thus, we find that we can construct $\boldsymbol{p'}$ by performing a double sort on $\boldsymbol{p}$, since $\sigma(\boldsymbol{p'}) = \sigma(\boldsymbol{p}) \Leftrightarrow \sigma(\sigma(\boldsymbol{p'})) = \sigma(\sigma(\boldsymbol{p})) \Leftrightarrow \boldsymbol{p'} = \sigma(\sigma(\boldsymbol{p}))$. In other words, the required compact alternative permutations for the example multivariate factorization are given by $\sigma(\sigma(\boldsymbol{o}_{(0,1)})) = (0,1)$ and $\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})) = (2,3,1,0)$.

Now that we know how to construct the correct query permutations for the gpdfs in a multivariate factorization given a query permutation for the multivariate factorization itself, we can focus on the actual product of permutation gpdfs. The frequency tables of the 2 individual multivariate joint permutation gpdfs in our example are of size $2! = 2$ and $4! = 24$. Assume that all parameters are uniform, e.g. both parameters for $\hat{P}(O_0,O_1)$ are $\frac{1}{2}$ and all 24 parameters for $\hat{P}(O_2,O_3,O_4,O_5)$ are $\frac{1}{24}$. We find that unlike in cases such as the discrete integer case and the case in which the normal gpdf is used for real–valued random variables, we have that

$$\hat{P}_{((0,1),(2,3,4,5))}(O_0,O_1,O_2,O_3,O_4,O_5)(\boldsymbol{o}) \neq \tag{4}$$
$$\hat{P}(O_0,O_1)(\sigma(\sigma(\boldsymbol{o}_{(0,1)})))\hat{P}(O_2,O_3,O_4,O_5)(\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})))$$

The reason why the multivariate factorization is not just the product of the individual gpdfs for the mutual exclusive factors, is that the righthandside in equation 4 is not a probability distribution over the 6 random variables combined. The reason for this is that the number of possible permutations of length 6 is $6! = 720$. Therefore, the summation of the wrongly factorized probabilities over all of these 720 permutations equals $720 \cdot \frac{1}{2} \cdot \frac{1}{24} = 15 \neq 1$. In the case of binary random variables, the number of possible combinations would be $2^6 = 64$ and the individual frequency tables would have been of size $2^2 = 4$ and $2^4 = 16$. If each individual probability would then have been $\frac{1}{4}$ and $\frac{1}{16}$ respectively for example, the summation over all possible combinations would be $64 \cdot \frac{1}{4} \cdot \frac{1}{16} = 1$.

Multivariate factorizations for permutations are different because if we know that $O_0 < O_1$ and $O_2 < O_3 < O_4 < O_5$, then there are a multiple of events regarding all six permutation random variables in which both equations hold instead of only a single permutation. Two examples of such events are $O_0 < O_1 < O_2 < O_3 < O_4 < O_5$ and $O_0 < O_2 < O_3 < O_4 < O_5 < O_1$. There are 15 of such "indistinguishable" events since the total number of possible permutations is $6! = 720$ and the total number of groups of permutations that can be made with the 2 shorter permutations is $2! \cdot 4! = 48$. This means that the correct factorization of the probability distribution is given by:

$$\hat{P}_{((0,1),(2,3,4,5))}(O_0, O_1, O_2, O_3, O_4, O_5)(\boldsymbol{o}) = \tag{5}$$
$$\frac{2!4!}{6!}\hat{P}(O_0, O_1)(\sigma(\sigma(\boldsymbol{o}_{(0,1)})))\hat{P}(O_2, O_3, O_4, O_5)(\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})))$$

The number of values that can be assigned to random variables $O_{\boldsymbol{\nu}_i}$ of a single multivariate joint factor is $values(O_{\boldsymbol{\nu}_i}) = |\boldsymbol{\nu}_i|!$. Since the individual factors in a multivariate factorization are taken to be totally *independent* of each other, the total number of values that the product of the gpdfs can uniquely assign a probability to, is $\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!$. On the other hand, the number of values that can be assigned to all random variables $\boldsymbol{\mathcal{O}}$ equals the number of permutations of length l, that is, $values(\boldsymbol{\mathcal{O}}) = l!$. Therefore, to construct a probability distribution over all possible permutations of length l, the product of multivariate gpdfs $\prod_{i=0}^{l-1} \hat{P}(O_{\boldsymbol{\nu}_i})(\sigma(\sigma(\boldsymbol{o}_{\boldsymbol{\nu}_i})))$ must be normalized by multiplication with $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!}$. The multivariate factorization for l permutation random variables $\boldsymbol{\mathcal{O}}$ can now be defined as follows:

$$\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\boldsymbol{o}) = \frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!} \prod_{i=0}^{l-1} \hat{P}(O_{\boldsymbol{\nu}_i})(\sigma(\sigma(\boldsymbol{o}_{\boldsymbol{\nu}_i}))) \tag{6}$$

3.4 Parameter representation

The parameters that need to be estimated for the multivariate factorization for permutations need to be stored somehow. One way of doing so is to store them in a frequency table, just as is common practice in the use of binary random variables or in the use of the histogram gpdf for real–valued random variables. An alternative approach is given by default tables. In this subsection, we discuss the realization of both approaches.

3.4.1 A direct representation through frequency tables

From the definition of the multivariate factorization, and the maximum likelihood permutation gpdf estimate, we have to be able to count the frequencies for a selection of discrete integer permutation random variables $O_{\boldsymbol{\nu}_j}$. The overhead that we at least have for this task according to equation 3, is in the decoding of all random keys sequences at the positions indicated by the j–th factor $\boldsymbol{\nu}_j$. This can be done in $\mathcal{O}(|\boldsymbol{\mathcal{S}}||\boldsymbol{\nu}_j|\log(|\boldsymbol{\nu}_j|))$ time by sorting all random keys sequences. To construct a frequency table for permutations of length $|\boldsymbol{\nu}_j|$, we require a frequency table of minimum size $|\boldsymbol{\nu}_j|! - 1$. In order to generate such a frequency table, we can take two approaches, which we discuss next in turn.

Bijective integer mapping
We can construct a bijective mapping between permutations and integers and use this mapping to index an array of size $|\boldsymbol{\nu}_j|! - 1$. The reason why this mapping must be bijective is that once we have computed the frequency tables, we must also be able to sample from them. To do so, we must know for each index what permutation it is associated with. Details on how such a bijective mapping can be constructed can be found in an earlier publication [6].

Permutation–indexed tables
An alternative to use of the bijective integer mapping is to allocate an array of size $|\boldsymbol{\nu}_j|^{|\boldsymbol{\nu}_j|}$. However, this approach is very inefficient seen to the large number of non–permutations that can be

$\boldsymbol{\mathcal{S}}$					
0	1	2	3	4	5
3	5	4	2	1	0
3	4	2	0	1	2
3	5	4	2	1	0
0	1	2	3	4	5
0	1	2	3	4	5
3	4	5	1	2	0

Default table						
0	1	2	3	4	5	$\frac{3}{7}$
default						$\frac{4}{5033}$

Default table						
0	1	2	3	4	5	$\frac{3}{7}$
3	4	2	0	1	2	$\frac{1}{7}$
3	4	5	1	2	0	$\frac{1}{7}$
3	5	4	2	1	0	$\frac{2}{7}$
default						0

Figure 1: Two examples of default tables. On the left, a sample vector is shown that contains 7 permutations. The center default table has only an entry for the single most frequently appearing permutation. On the right, a default table is shown in which the same information is stored as would be contained in a frequency table, only more efficient due to the use of a default entry.

represented. It is more efficient to make a table of size $|\boldsymbol{\nu}_j|! - 1$ that can be indexed using a permutation. To estimate the frequencies as efficiently as possible in this case, the available permutations must be sorted, which can be done in $\mathcal{O}(|\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}|\log(|\boldsymbol{\mathcal{S}}|))$ time. By scanning the sorted list of permutations simultaneously with the full frequency table of size $|\boldsymbol{\nu}_j|!$, the required average frequencies can then be counted in $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}|)$ time. The total amount of required time using permutation indexed tables is thus given by $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}|(1 + \log(|\boldsymbol{\mathcal{S}}|)))$.

3.4.2 Introducing local structures through default tables

The *default table* [9] offers an alternative to the frequency table. In a default table, the probabilities are explicitly specified for a subset of all available entries. For the absent entries, a *default value* is used, which is the average probability of all absent values. Examples of default tables for permutations are given in Figure 1. One straightforward way to use default tables, is to only specify the average frequency for each entry that occurs in the sample vector. By doing so, no factor in a multivariate factorization can give rise to more parameters than $|\boldsymbol{\mathcal{S}}|$. Although the information in the so constructed default table is the same as that contained in a frequency table, the number of parameters that need to be estimated equals the number of different samples in $\boldsymbol{\mathcal{S}}$. For this reason, if the number of different samples is significantly less than the total number of different possible samples, a default table containing the same information as a frequency table can be estimated more efficiently than can a frequency table. For permutations for instance, the list of permutations is first sorted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|\log(|\boldsymbol{\mathcal{S}}|))$ time, after which the frequencies are counted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|)$ time, completing the construction of the default table. The total running time for constructing a default table is thus $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|(1 + \log(|\boldsymbol{\mathcal{S}}|)))$, whereas the construction of a frequency table requires $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\mathcal{S}}||\boldsymbol{\nu}_j|\log(|\boldsymbol{\nu}_j|))$. Note that we cannot map permutations to integers for faster sorting because the integers would become too large to efficiently represent with increasing $|\boldsymbol{\nu}_j|$, which is likely to happen using default tables.

Default tables containing the same information as frequency tables may not always be estimated faster than a frequency table since it may still be required that $|\boldsymbol{\mathcal{S}}| = \mathcal{O}(\kappa^{\iota}!)$ when there are subproblems with a maximum length of κ^{ι} that need to be exhaustively sampled. However, when we must combine lower order solutions to get solutions of a higher order, the default tables can give us a much more efficient representation of the few good solutions to the subproblems.

This latter issue is an important benefit of using *local structures* in probability distributions. A local structure allows for a more explicit representation of dependencies between the *values* that can be assumed by random variables instead of dependencies between the random variables *themselves*. As a result, less parameters need to be estimated. Probability distributions that are capable of expressing more complex dependencies now become eligible for selection when using a penalization metric, whereas otherwise non–local structure probability distributions expressing similar dependencies would never have been regarded because of the large number of (redundant)

parameters they impose [9]. Experimental verification of the usefulness of local structures has been given by Pelikan and Goldberg [16]. They show that using local structures in the iterated estimation of probability distributions for binary random variables allows for efficient optimization of very difficult *hierarchical* deceptive optimization problems that exhibit dependencies between combinations of values for large groups of variables.

3.5 Factorization selection

To find a good multivariate factorization for permutation random variables given a sample vector $\boldsymbol{\mathcal{S}}$ of data, we use an incremental greedy algorithm to minimize a metric that represents a trade–off between the likelihood and the complexity of the estimated probability distribution. This is a common approach that has been observed to give good results [4, 11, 13, 16].

Two of such commonly known metrics that have often proved to be successful, are known as the *Akaike Information Criterion* (AIC) and the *Bayesian Information Criterion* (BIC). Both metrics score a probability distribution by its negative log–likelihood, but add a penalty term. This penalty term increases with the complexity of the probability distribution (e.g. the number of parameters $|\boldsymbol{\theta}|$). In the case of the BIC metric, the penalty term also increases with the size of the sample vector (e.g. $|\boldsymbol{\mathcal{S}}|$):

$$AIC(\mathcal{M}|\boldsymbol{\mathcal{S}}) = \underbrace{-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right)}_{\text{Error}(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})|\boldsymbol{\mathcal{S}})} + \underbrace{|\boldsymbol{\theta}|}_{\text{Complexity}(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})|\boldsymbol{\mathcal{S}})} \tag{7}$$

$$BIC(\mathcal{M}|\boldsymbol{\mathcal{S}}) = \underbrace{-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right)}_{\text{Error}(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})|\boldsymbol{\mathcal{S}})} + \underbrace{\frac{1}{2}\ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|}_{\text{Complexity}(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})|\boldsymbol{\mathcal{S}})} \tag{8}$$

3.5.1 Greedy multivariate factorization selection by splicing factors

The factorization learning algorithm starts from the univariate factorization in which all variables are independent of each other $\boldsymbol{\nu} = ((0), (1), \ldots, (l-1))$. Each iteration, an operation that changes $\boldsymbol{\nu}$ is performed such that the value of the penalization metric decreases. This procedure is repeated until no further improvement can be made. One operation that by means of which the factorization may be altered, is the splicing (joining) of two factors $\boldsymbol{\nu}_{s_0}$ and $\boldsymbol{\nu}_{s_1}$. The splice operation that decreases the metric the most, is actually performed.

From equations 7 and 8 we find that the greedy algorithm searches for the largest value of the penalized negative log–likelihood of the current factorization $\boldsymbol{\nu^0}$ *minus* the penalized negative log–likelihood of the candidate factorization $\boldsymbol{\nu^1}$. Since the penalization is additive to this difference, it can easily be shown that for the AIC and BIC metrics the penalization for this difference may be determined separately [3]. This difference equals $\delta\left(|\boldsymbol{\theta} \overset{\leftarrow fit}{} (\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1})| + |\boldsymbol{\theta} \overset{\leftarrow fit}{} \boldsymbol{\nu}_{s_0}| + |\boldsymbol{\theta} \overset{\leftarrow fit}{} \boldsymbol{\nu}_{s_1}|\right)$ where $\delta = 1$ for the AIC metric and $\delta = \frac{1}{2}\ln(|\boldsymbol{\mathcal{S}}|)$ for the BIC metric. We can now determine the difference in negative log–likelihood separately.

Using the definition of the multivariate factorization for permutation random variables from equation 6, and by realizing that $\sigma(\sigma(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i))_{\boldsymbol{a}})) = \sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}}))$, the negative log–likelihood of the multivariately factorized probability distribution can be written as:

$$-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) = -\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\frac{\prod_{j=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_j|!}{l!} \prod_{j=0}^{|\boldsymbol{\nu}|-1} \hat{P}(O_{\boldsymbol{\nu}_j})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_j})))\right) = \tag{9}$$

$$|\boldsymbol{\mathcal{S}}| \ln(l!) - \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \sum_{j=0}^{|\boldsymbol{\nu}|-1} \ln\left(|\boldsymbol{\nu}_j|! \hat{P}(O_{\boldsymbol{\nu}_j})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_j})))\right)$$

Now let $\boldsymbol{\nu^0}$ and $\boldsymbol{\nu^1}$ be node partition vectors for multivariate factorizations such that $\boldsymbol{\nu^1}$ contains all node vectors in $\boldsymbol{\nu^0}$ except two node vectors ${\boldsymbol{\nu^0}}_{s_0}$ and ${\boldsymbol{\nu^0}}_{s_1}$. Furthermore, the only additional node vector that is contained in $\boldsymbol{\nu^1}$ is the node vector ${\boldsymbol{\nu^0}}_{s_0} \sqcup {\boldsymbol{\nu^0}}_{s_1}$. The negative log–likelihood for the multivariate factorization based on $\boldsymbol{\nu^0}$ minus the negative log–likelihood for the multivariate factorization based on $\boldsymbol{\nu^1}$ can be found by cancelling terms to be:

$$\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu^1}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) - \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu^0}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) \tag{10}$$

$$=$$

$$\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \Big[\ln\left((|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)! \hat{P}(O_{\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}})))\right)$$

$$-\ln\left(|\boldsymbol{\nu}_{s_0}|! \hat{P}(O_{\boldsymbol{\nu}_{s_0}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}})))\right) - \ln\left(|\boldsymbol{\nu}_{s_1}|! \hat{P}(O_{\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_1}})))\right)\Big]$$

$$=$$

$$|\boldsymbol{\mathcal{S}}| \ln\left(\frac{(|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)!}{|\boldsymbol{\nu}_{s_0}|! |\boldsymbol{\nu}_{s_1}|!}\right) + \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \Big[\ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}})))\right)$$

$$-\ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_0}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}})))\right) - \ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_1}})))\right)\Big]$$

3.5.2 Problems with the use of the splice operation

It follows from the result in equation 10 that there is a problem if *only* the splice operator is used in the greedy search algorithm on additively decomposable optimization problems. The following example serves to makes this intuitively clear.

Consider a sample vector $\boldsymbol{\mathcal{S}}$ of random keys for which the indices in two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of length 5 only occur in the sample vector in a single permutation; e.g. $(\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}_0} < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}_4} < \ldots < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}_0}$ and $(\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{b}_0} < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{b}_1} < \ldots < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{b}_4}$ for all $i \in \{0, 1, \ldots, |\boldsymbol{\mathcal{S}}| - 1\}$. It is then quite likely that $(\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}_0} < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{b}_4}$ holds for all $i \in \{0, 1, \ldots, |\boldsymbol{\mathcal{S}}| - 1\}$. If this is so, the greedy splice algorithm for multivariate factorization selection is in its first stages just as likely to choose to splice the singleton node vectors that contain $(\boldsymbol{a}_0)$ and $(\boldsymbol{a}_4)$, which we want, as it is to splice the singleton node vectors that contain $(\boldsymbol{a}_0)$ and $(\boldsymbol{b}_4)$, which we do *not* want. When the splicing algorithm has mixed up the index vectors in this way, drawing new random key sequences from the estimated multivariate factorization is *very* unlikely to generate new correct permutations. This is a problem that does not occur for binary random variables, because once all 10 bits have for instance fully converged to 1, it doesn't matter for reproduction whether we use a full joint perfect factorization $\boldsymbol{\nu} = \boldsymbol{a} \sqcup \boldsymbol{b}$ or a univariate factorization $\boldsymbol{\nu} = ((\boldsymbol{a}_0), (\boldsymbol{a}_1), (\ldots), (\boldsymbol{a}_4), (\boldsymbol{b}_0), (\boldsymbol{b}_1), (\ldots), (\boldsymbol{b}_4))$. Upon sampling, both factorizations will return a binary genotype that has a 1–symbol at all positions of index vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

This problem occurs because *decision errors* are made at a lower dependency level.These lower order errors cannot be avoided and only become visible at a higher dependency level. When regarding dependencies of level 5 for instance, it *does* become clear that the index vectors must be separated. The reason for this is that the individual random key sequences for the index vectors have converged to a single permutation, but other combinations of length 5 lead to random key sequences that represent different permutations throughout $\boldsymbol{\mathcal{S}}$. By definition, the likelihood of the correct factorization is therefore larger.

To overcome this problem, either the size of the sample vector has to increase significantly to reduce the probability of $(\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{a}_0} < (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{b}_4}$ or we require a way to correct for lower order decision errors. The problem can be avoided by the greedy incremental algorithm if we allow the splicing

of more than 2 vectors at once. However, by allowing the splicing of k vectors, we get a running time complexity of $\mathcal{O}(l^{k+1})$ for the greedy search algorithm. Since this significantly influences the scale–up behavior of the algorithm, we propose to extend the greedy search algorithm by allowing a second operator. This operator allows to correct for lower order decision errors that were made at an earlier stage. This is enforced by allowing two subvectors of the node vector to exchange an element. We call this the *swap* operator. The swap operation that decreases the negative log–likelihood the most is performed first. Since the complexity of the factorization does not increase, no penalization is required for this operation. To ensure that splice operations are only performed when lower order decision errors are no longer visible at the current stage of the greedy algorithm, a swap operation is always preferred over a splice operation.

However, there are situations in which even a swap operation cannot undo a low order decision error. To this end, we propose to allow a third operator in the greedy learning algorithm, which we call the *transfer* operator. With this operator, we allow a factor $\boldsymbol{\nu}_i$ to transfer an index to another factor $\boldsymbol{\nu}_j$.

3.5.3 Additional problems with the use of default tables

Another problem arises when we regard the case in which we use default tables to store the probabilities for each factor. If the length of the default table for factor $\boldsymbol{\nu}_i$ is close to $|\boldsymbol{\mathcal{S}}|$, there is no telling whether this length is representative of the number of *true* permutations or whether this is due to the maximum length of $|\boldsymbol{\mathcal{S}}|$ that any default table can have. This reduces the reliability of the BIC metric, because if factor $\boldsymbol{\nu}_i$ becomes even larger, it *cannot* have a much larger default table length because of the limiting size of the sample vector. However, because the number of possible permutations increases factorially, the likelihood of the probability distribution will increase significantly due to the term that is additional to the difference in the negative log–likelihood of the gpdfs (see equation 10). Since $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_i|$ now equals the default table length instead of $|\boldsymbol{\nu}_i|!$, there is hardly any complexity penalization.

To remedy this problem, we require a cutoff value $\xi \in [0, 1]$ that defines the maximum default table length to be $\xi|\boldsymbol{\mathcal{S}}|$. No operation is allowed to create a factor that has a default table longer than $\xi|\boldsymbol{\mathcal{S}}|$. The rationale behind ξ is that if the default table length becomes larger than $\xi|\boldsymbol{\mathcal{S}}|$, we decide that the reason for this length is uncertain. Without this restriction, there would be an *early* drift towards large factors when $|\boldsymbol{\nu}_i|!$ starts to get larger than $\boldsymbol{\mathcal{S}}$.

4 IDEAs and ICE

We assume to have a cost function $C(\boldsymbol{z})$ of l problem variables $z_0, z_1, \ldots, z_{l-1}$ that, without loss of generality, should be minimized. For each z_i, we introduce a stochastic random variable Z_i and let $P^{\theta}(\boldsymbol{\mathcal{Z}})$ be a probability distribution that is uniform over all $\boldsymbol{z}$ with $C(\boldsymbol{z}) \leq \theta$ and 0 otherwise. Sampling from $P^{\theta}(\boldsymbol{\mathcal{Z}})$ gives more samples with an associated cost $\leq \theta$. Moreover, if we have access to $P^{\theta^*}(\boldsymbol{\mathcal{Z}})$ such that $\theta^* = \min_{\boldsymbol{z}}\{C(\boldsymbol{z})\}$, drawing only a single sample results in an optimal solution. This rationale underlies the IDEA (*Iterated Density Estimation Evolutionary Algorithm*) framework [4] and other named variants [11, 13, 14, 16, 17, 18, 19].

Problem structure in the form of dependencies between the problem variables, is *induced* from a vector of samples by finding a suitable probabilistic model $\mathcal{M}$. A probabilistic model is used as a computational implementation of a probability distribution $P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})$. A probabilistic model consists of a structure ς and a vector of parameters $\boldsymbol{\theta}$. The elementary building blocks of the probabilistic model are taken to be probability density functions (pdfs). A structure ς describes what pdfs are used and the parameter vector $\boldsymbol{\theta}$ describes the values for the parameters of these individual pdfs. A *factorization* is an example of a structure ς. A factorization *factors* the probability distribution over $\boldsymbol{\mathcal{Z}}$ into a product of pdfs. In this paper, we have focused on multivariate factorizations.

These definitions are used in the IDEA by selecting $\lfloor \tau n \rfloor$ samples ($\tau \geq \frac{1}{n}$) in each iteration t and by letting θ_t be the worst selected sample cost. The probability distribution $\hat{P}_{\varsigma}^{\theta_t}(\boldsymbol{\mathcal{Z}})$ of the selected samples is then estimated, which is an approximation to the uniform probability

distribution $P^{\theta_t}(\boldsymbol{\mathcal{Z}})$. New samples can then be drawn from $\hat{P}^{\theta_t}(\boldsymbol{\mathcal{Z}})$ to replace some of the current samples. A special instance of the IDEA framework is obtained if selection is done by taking the top $\lfloor \tau n \rfloor$ best samples from the population, $\tau \in [\frac{1}{n}, 1]$, we draw $n - \lfloor \tau n \rfloor$ new samples, and the new samples replace the current worst $n - \lfloor \tau n \rfloor$ samples in the population. This results in the use of *elitism* such that $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{t_{\text{end}}}$. We call the resulting algorithm a *monotonic* IDEA.

Since the random keys are essentially a real valued domain, real valued IDEAs can directly be applied to permutation problems. Such an approach based upon normal probability distributions was proposed by Bosman and Thierens [5] as well as by Robles, de Miguel and Larrañaga [17]. However, the study by Bosman and Thierens [5] showed that this does not lead to very effective permutation optimization. The main problem with this approach is that solutions are not processed in the permutation space but in the largely redundant real valued space. To overcome this problem, a crossover operator was proposed [5]. This crossover operator reflects the dependency information learned in a factorization. Two parents are first selected at random. In the case of an multivariate factorizations, the crossover operator then copies the values at the positions indicated by a vector in $\boldsymbol{\nu}$ from one of the two parents. This is repeated until all vectors in $\boldsymbol{\nu}$ have been regarded. Thus, whereas the IDEA is used to find the multivariate factorizations, crossover is used instead of probabilistic sampling to generate new solutions. The resulting algorithm is called ICE (IDEA *Induced Chromosome Elements Exchanger*). Using ICE instead of a pure real valued IDEA gives significantly better results. The results are comparable with the only other EA that learns permutation structure information, which is the OmeGA by Knjazew [12]. The OmeGA is essentially a fmGA that works with random keys. The dependency information in this normal ICE is however still induced using normal distributions estimated over a largely redundant space, which may introduce false dependency information. To improve induction in ICE, these dependencies can be induced in the space of permutations directly by interpreting the random keys as permutations. This is what has been described in this paper in detail in the previous sections. Experiments with multivariate factorizations for permutation random variables have indicated a significant improvement in the scalability of ICE with respect to the minimally required population size and the number of required function evaluations to find the optimal solution for two difficult relative–ordering permutation optimization problems [7].

References

[1] T. W. Anderson. *An Introduction to Multivariate Statistical Analysis*. John Wiley & Sons Inc., New York, New York, 1958.

[2] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.

[3] P. A. N. Bosman and D. Thierens. Mixed IDEAs. Utrecht University Technical Report UU–CS–2000–45., 2000.

[4] P. A. N. Bosman and D. Thierens. Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In M. Pelikan and K. Sastry, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2001*, pages 208–212. Morgan Kaufmann, 2001.

[5] P. A. N. Bosman and D. Thierens. Crossing the road to efficient IDEAs for permutation problems. In L. Spector, E.D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, M. H. Garzon S. Pezeshk, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO–2001*, pages 219–226. Morgan Kaufmann, 2001.

[6] P. A. N. Bosman and D. Thierens. New IDEAs and more ICE by learning and using unconditional permutation factorizations. In *Late–Breaking Papers of the Genetic and Evolutionary Computation Conference GECCO–2001*, pages 16–23, 2001.

[7] P. A. N. Bosman and D. Thierens. Permutation optimization by iterated estimation of random keys marginal product factorizations. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-J. Fernández-Villicañas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VII*, pages 331–340. Springer Verlag, 2002.

[8] L. Davis. Applying adaptive algorithms to epistatic domains. In A. Joshi, editor, *Proceedings of the Nineth International Joint Conference on Artificial Intelligence*, pages 162–164. Morgan Kaufmann, 1985.

[9] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In E. Horvits and F. Jensen, editors, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence UAI–1996*, pages 252–262. Morgan Kaufmann, 1996.

[10] D. E. Goldberg and R. Lingle, Jr. Alleles, loci, and the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 154–159. Lawrence Erlbaum Associates, 1985.

[11] G. Harik and D. E. Goldberg. Linkage learning through probabilistic expression. *Comp. methods in applied mechanics and engineering*, 186:295–310, 2000.

[12] D. Knjazew. Application of the fast messy genetic algorithm to permutation and scheduling problems. IlliGAL Technical Report 2000022, 2000.

[13] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.

[14] A. Ochoa, H. Mühlenbein, and M. Soto. A factorized distribution algorithm using single connected Bayesian networks. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 787–796. Springer Verlag, 2000.

[15] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 224–230. Lawrence Erlbaum Associates, 1987.

[16] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 511–518. Morgan Kaufmann, 2001.

[17] V. Robles, P. de Miguel, and P. Larrañaga. Solving the traveling salesman problem with EDAs. In P. Larrañaga and J.A. Lozano, editors, *Estimation of Distribution Algorithms. A new tool for Evolutionary Computation.* Kluwer Academic, 2001.

[18] R. Santana, A. Ochoa, and M. R. Soto. The mixture of trees factorized distribution algorithm. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 543–550. Morgan Kaufmann, 2001.

[19] S.-Y. Shin and B.-T. Zhang. Bayesian evolutionary algorithms for continuous function optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation – CEC2001*, pages 508–515. IEEE Press, 2001.

[20] M. M. Tatsuoka. *Multivariate Analysis: Techniques for Educational and Psychological Research*. John Wiley & Sons Inc., New York, New York, 1971.

[21] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufman, 1989.