New IDEAs and More ICE by Learning and Using Unconditional Permutation Factorizations

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

Abstract

Solving permutation optimization problems is an important and open research question. Using continuous iterated density estimation evolutionary algorithms (IDEAs) in combination with crossover from genetic algorithms (GAs) has recently [5] been shown to give promising results. In IDEAs, the probability distribution of the solutions is estimated based upon a selection of solutions. So far, only continuous probability theory has been applied to a continuous encoding of permutations. In this paper, we show how we can estimate and use unconditional factorization distributions in the space of permutations directly. We show that the resulting IDEAs process the permutation linkage information more effectively than previously used continuous IDEAs. As a result, deceptive permutation optimization problems of a bounded order can be solved more efficiently.

1 Introduction

In the field of evolutionary computation, three algorithms have been proposed so far to learn and use the structure of permutation optimization problems in order to achieve more efficient optimization. The first algorithm to appear was the OmeGA by Knazjew and Goldberg [11]. This algorithm is essentially a *fast messy GA* [7] that has been adapted to work with random keys. The random keys encoding is a real representation of permutations. An algorithm based upon a continuous IDEA was proposed by Bosman and Thierens [5]. In the IDEA framework, a probabilistic model is estimated over a selection of the available solutions in each generation. Subsequently, new solutions are drawn from the estimated probability distribution. Since the random keys provide a real encoding of permutations, continuous probability theory in IDEAs can be directly applied. This approach was also used by Robles, de Miguel and Larrañaga [17]. However, in the work by Bosman and Thierens, it was shown that continuous IDEAs do not scale up by far as efficiently as for instance does the OmeGA. To overcome the problems with directly applying continuous IDEAs, the ICE algorithm was proposed in which probabilistic sampling of new solutions is replaced by a specialized crossover operator to mix the building blocks. By doing so, the evolutionary algorithm (EA) directly explores the permutation space, which significantly increases its efficiency. The results obtained by ICE are comparable to those of the OmeGA. This is an indication that an IDEA designed to work directly in the space of permutations may even more effectively find and use the structure of permutation problems as opposed to continuous IDEAs and continuous ICE.

Our goal in this paper is to investigate how such a permutation IDEA can be designed. However, the permutation space is fundamentally different from the binary or the real space that most EAs have been designed for. We therefore require new definitions, which we provide in this paper. So far, we are able to learn *unconditionally* factorized probability distributions. This results in new EAs for permutation optimization problems.

The remained of this paper is organized as follows. In section 2 we discuss probabilistic models, IDEA and ICE. Next, we go over the random keys encoding of permutations in section 3 and present the permutation problems that we will use for testing. Subsequently, we present new tools for the probabilistic modeling of permutations represented by random keys in section 4. In section 5 we discuss how these tools can be used to learn unconditional factorizations. We test the resulting EAs in section 6. We discuss future research in section 7 and present our conclusions in section 8.

2 Probabilistic models, IDEA and ICE

In the following, we assume a continuous domain and write the problem variables as y_i. Assume that we have an l dimensional cost function $C(y_0, y_1, \ldots, y_{l-1})$, which without loss of generality we seek to minimize. For each y_i, we introduce a random variable Y_i. We let $P^\theta(\boldsymbol{\mathcal{Y}}) = P^\theta(Y_0, Y_1, \ldots, Y_{l-1})$ be a probability distribution that is uniform over all $\boldsymbol{y}$ with $C(\boldsymbol{y}) \leq \theta$ and 0 otherwise. Sampling from $P^\theta(\boldsymbol{\mathcal{Y}})$ gives more samples that evaluate to a value below θ. Moreover, if we know $\theta^* = \min_{\boldsymbol{y}}\{C(\boldsymbol{y})\}$, sampling from $P^{\theta^*}(\boldsymbol{\mathcal{Y}})$ gives an optimal solution. This rationale underlies the IDEA (*Iterated Density Estimation Evolutionary Algorithm*) framework [3] and other variants in both binary [8, 13, 14, 15, 16] and continuous spaces [6, 12].

The interactions between the problem variables are attempted to be *inferred* from the given sample vector to find a suitable probabilistic model $\mathcal{M}$. The notion of a probabilistic model is used as a computational implementation of a probability distribution $P_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})$. A probabilistic model can be seen to consist of some 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. An example of a structure ς is the notion of a *factorization*. A factorization *factors* the probability distribution over $\boldsymbol{\mathcal{Y}}$ into a product of pdfs. In this paper, we focus on *unconditional* factorizations. In the binary and real case, an unconditional factorization is a product of multivariate joint pdfs. This product is represented by a vector of mutually exclusive subvectors, which we call the *node vector* $\boldsymbol{\nu}$. An example in the case of $l = 3$ is $P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Y}}) = P(Y_0, Y_1)P(Y_2)$, meaning $\boldsymbol{\nu} = ((0, 1), (2))$. Once a structure ς is given, the parameters that have to be estimated can be derived from the multivariate pdfs that have to be fit. Since the way in which the parameters $\boldsymbol{\theta}$ are fit, is predefined on beforehand, we denote the parameter vector that is obtained in this manner by $\boldsymbol{\theta} \xleftarrow{fit} \varsigma$. This implies that whereas we formally define a probabilistic model to be $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$, in our practical case, we can write $\mathcal{M} = (\varsigma, \boldsymbol{\theta} \xleftarrow{fit} \varsigma)$. As a probability distribution can therefore be identified using only the structure ς, we denote it by $P_\varsigma(\boldsymbol{\mathcal{Y}})$.

These definitions are used in the IDEA by selecting $\lfloor \tau n \rfloor$ samples in each iteration t and by letting θ_t be the worst selected sample cost. The probability distribution of the selected samples is then estimated, resulting in $\hat{P}^{\theta_t}_\varsigma(\boldsymbol{\mathcal{Y}})$ as an approximation to the true probability distribution $P^{\theta_t}(\boldsymbol{\mathcal{Y}})$. New samples can then be drawn from $\hat{P}^{\theta_t}(\boldsymbol{\mathcal{Y}})$ and be used to replace some of the current samples. A formal definition and more details on the IDEA framework can be found elsewhere [3]. A special instance of the IDEA framework is obtained if selection is done by taking the best $\lfloor \tau n \rfloor$ samples, the amount of *new* samples is set to $n - \lfloor \tau n \rfloor$ and all of these new samples are used to replace the worst $n - \lfloor \tau n \rfloor$ samples. This results in the use of *elitism* such that the search for θ^* is conveyed through a monotonically decreasing series $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{t_{\text{end}}}$. We call the resulting algorithm a *monotonic* IDEA.

Since the random keys representation for permutations is essentially a continuous domain, continuous IDEAs can directly be applied to permutation problems. However, a recent study [5] showed that this does not lead to very effective optimization algorithms. The main problem with this approach is that this introduces a large overhead because the solutions are not processed in the permutation space but in the continuous space. To overcome this problem, a crossover operator was proposed. This crossover operator respects the information learned in a factorization. Two parents are first selected at random. In the case of an unconditional factorization, the crossover operator then copies the values at the positions indicated by a subvector in $\boldsymbol{\nu}$ from one of the two parents. This is repeated until all subvectors in $\boldsymbol{\nu}$ have been regarded. Thus, whereas the IDEA is used to find the unconditional factorization, 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 continuous IDEA gave significantly better results.

3 Permutation optimization problems and random keys

The *random keys* encoding of permutations was introduced by Bean [1]. The main advantage of random keys is that no crossover operator can create unfeasible solutions since each encoding represents a permutation. To encode a permutation of length l, each integer in $\{0, 1, \ldots, l-1\}$ is assigned a value (key) from some real domain, which is usually taken to be $[0, 1]$. Subsequently, the integers are sorted on the keys to get the corresponding permutation. This decoding requires $\mathcal{O}(l \log l)$ time. The random key string 0.61 0.51 0.62 0.31 is for instance mapped to 3 1 0 2.

In this paper, we test our algorithms on the general deceptive permutation problem introduced by Knazjew [10]. The optimum for this problem is the trivial permutation $(0, 1, \ldots, l_{BB} - 1)$. To define the fitness

```
ϱ(n)
IntegerToRandomKeys( n, length )
1  o ← new array of integer with size length
2  p ← new array of integer with size length
3  k ← new array of real with size length
4  for i ← 0 to length − 1 do
   4.1  o[i] ← i
   4.2  k[i] ← Random([0, 1])
5  fac ← (length − 1)!
6  for i ← 0 to length − 1 do
   6.1  pos ← n / fac
   6.2  n ← n − pos · fac
   6.3  p[i] ← o[pos]
   6.4  o[pos] ← o[length − 1 − i]
   6.5  if i < length − 1 then
        6.5.1  fac ← fac / (length − i − 1)
7  p2 ← SortKeys( k, length )
8  for i ← 0 to length − 1 do
   8.1  keys[p[i]] ← k[p2[i]]
9  Return(keys)
```

Figure 1: Converting integers to random keys.

function, a distance measure is used. The distance from any permutation $\boldsymbol{p}$ to the optimum equals $l_{BB} - |\textsc{lis}(\boldsymbol{p})|$, where $\textsc{lis}(\boldsymbol{p})$ is the longest increasing subsequence in $\boldsymbol{p}$. For example, if $\boldsymbol{p} = (4, 0, 3, 1, 2)$, then $\textsc{lis}(\boldsymbol{p}) = (0, 1, 2)$ and $l_{BB} - |\textsc{lis}(\boldsymbol{p})| = 5 - 3 = 2$. Note that the reverse permutation $(l_{BB} - 1, l_{BB} - 2, \ldots, 0)$ is the only permutation with a distance of $l_{BB} - 1$. The deceptive ordering problem for a single *building block* (BB) of length l_{BB}, is defined as follows:

$$f_{BB}(\boldsymbol{p}) = \begin{cases} 1 - \frac{|\textsc{lis}(\boldsymbol{p})|}{l_{BB}} & \text{if } |\textsc{lis}(\boldsymbol{p})| < l_{BB} \\ 1 & \text{if } |\textsc{lis}(\boldsymbol{p})| = l_{BB} \end{cases} \quad (1)$$

A building block is a subsequence of the complete random key string. The actual fitness function that we use, has length $l = n_{BB} l_{BB}$, where n_{BB} is the amount of building blocks. The fitness value is the sum of the fitness values of the individual building blocks. The resulting optimization problem is therefore *additively decomposable*. The locations of the individual building blocks have been coded *loosely*. This implies that building block $0 \leq i < n_{BB}$ consists of the random keys found at locations $(i, i + n_{BB}, \ldots, i + (l_{BB} - 1)n_{BB})$. The fact that the problem is fully deceptive, means that all schemata of an order smaller than k lead to the suboptimum of the reverse permutation [9]. This makes the problem hard for any optimizer that doesn't identify which random key positions together constitute a building block [2]. Furthermore, because of the loose coding, simple crossover schemes such as one point crossover are not effective either.

Perm.	ℕ
0 3 2 1	0
0 3 1 2	1
0 1 3 2	2
0 1 2 3	3
0 2 3 1	4
0 2 1 3	5
1 0 2 3	6
1 0 3 2	7

Perm.	ℕ
1 3 0 2	8
1 3 2 0	9
1 2 0 3	10
1 2 3 0	11
2 0 3 1	12
2 0 1 3	13
2 1 0 3	14
2 1 3 0	15

Perm.	ℕ
2 3 0 1	16
2 3 1 0	17
3 0 2 1	18
3 0 1 2	19
3 1 0 2	20
3 1 2 0	21
3 2 0 1	22
3 2 1 0	23

Figure 2: The mapping as defined by function $\varrho(n)$ (after the conversion of random keys to permutations).

4 Probability theory for permutations and random keys

The better results obtained by using $\mathbb{ICE}$ are evidence that efficient IDEAs can be designed by making them process the permutation space directly. To do so, we need a way to estimate the multivariate joint distribution of a set of random keys. Since we are interested in the distribution of the permutations that are represented by the random keys, we have to count the frequencies of the permutations. This means that for k random keys, we require a frequency table of minimum size $k!$. To generate the minimum size table, we need to map permutations onto integers and vice versa. If we have a number n that represents a random key with length k, we know that the number lies in the range $\{0, 1, \ldots k! - 1\}$. Therefore, $\lfloor \frac{n}{(k-1)!} \rfloor$ lies in the range $\{0, 1, \ldots, k-1\}$. Furthermore, n rem $(k-1)!$ lies in the range $\{0, 1, \ldots, (k-1)! - 1\}$. We can generate the corresponding permutation in k steps by first creating an initial dummy permutation. Then, in each step i, we take the final permutation element at position i to be the element at position $\lfloor \frac{n}{(k-i)!} \rfloor$ in the dummy permutation. To ensure that no elements from the dummy permutation are used twice, the $(k-i)$–th element in the dummy permutation replaces the used element at position i. To finally create a random key string from the so constructed permutation, k random numbers are drawn and sorted. Using the sorting information, the keys are placed in the positions indicated by the integer permutation that was computed from n. The resulting algorithm runs in $\mathcal{O}(k \log k)$ time and is given in figure 1. In figure 2 the function results are given for $k = 4$. For mathematical convention, we denote this function by $\varrho(n)$ and assume that the length of the random key is clear from the context.

To define the inverse function $\varrho^{-1}(keys)$, we have to know in each iteration at which position the i–th permutation element was located in the dummy permutation array. To this end, we make a second dummy

```
                    ϱ^{-1}(keys)
    RandomKeysToInteger( keys, length )
1   o1 ← new array of integer with size length
2   o2 ← new array of integer with size length
3   p ← SortKeys( keys, length )
4   for i ← 0 to length − 1 do
      4.1  o1[i] ← i
      4.2  o2[i] ← i
5   n ← 0
6   fac ← (length − 1)!
7   for i ← 0 to length − 1 do
      7.1  pos ← o2[p[i]]
      7.2  n ← n + pos · fac
      7.3  o2[o1[length − 1 − i]] ← pos
      7.4  o1[pos] ← o1[length − 1 − i]
      7.5  if i < length − 1 then
             7.5.1  fac ← fac / (length − i − 1)
8   Return(n)
```

Figure 3: Converting random keys to integers.

array in which this information is stored. Before the first dummy array is altered in the same way as is done in function $\varrho(n)$, we update the location information in the second dummy array. The resulting algorithm runs in $\mathcal{O}(k \log k)$ time and is given in figure 3.

Even though the random keys are actually continuous values, they represent discrete permutations. Therefore, we write a single solution string (sample) as $(r_0, r_1, \ldots, r_{l-1})$ instead of $(y_0, y_1, \ldots, y_{l-1})$. Similarly, we introduce a random variable R_i for each problem variable r_i. Since the random keys encode permutations, the symantics of the R_i are essentially different from those of binary or continuous random variables. We define $\boldsymbol{\mathcal{R}} = (R_0, R_1, \ldots, R_{l-1})$ and write the selected sample vector in the IDEA as $\boldsymbol{\mathcal{S}} = (\boldsymbol{r^0}, \boldsymbol{r^1}, \ldots, \boldsymbol{r^{|\mathcal{S}|-1}})$, $\boldsymbol{r^i} = (r_0^i, r_1^i, \ldots, r_{l-1}^i)$. The multivariate joint pdf over a subset of the random keys $R\langle \boldsymbol{v} \rangle = (R_{\boldsymbol{v}_0}, R_{\boldsymbol{v}_1}, \ldots, R_{\boldsymbol{v}_{|\boldsymbol{v}|-1}})$ is defined by the probability at a certain random key subsequence $r\langle \boldsymbol{v} \rangle$. It can be computed by counting frequencies in $\boldsymbol{\mathcal{S}}$:

$$\hat{P}(R\langle \boldsymbol{v} \rangle)(r\langle \boldsymbol{v} \rangle) = \tag{2}$$

$$\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \varrho^{-1}(r^i\langle \boldsymbol{v} \rangle) = \varrho^{-1}(r\langle \boldsymbol{v} \rangle) \\ 0 & \text{otherwise} \end{cases}$$

Defining the unconditionally factorized probability distribution over all random keys is not as straightforward as is the case for binary or continuous representations. For instance, regard the case in which $l = 4$ and $\boldsymbol{\nu} = ((0,1),(2,3))$. The frequency tables of the 2 individual multivariate joint pdfs are of size $2! = 2$. Assume that $\hat{P}(r_0 < r_1) = \hat{P}(r_2 < r_3) = \frac{1}{2}$. Unlike in the binary and continuous cases, we have $\hat{P}_{((0,1),(2,3))}(R_0, R_1, R_2, R_3) \neq \hat{P}(R_0, R_1)\hat{P}(R_2, R_3)$, because the righthandside is not a probability distribution over the 4 random variables combined. The reason for this is that the amount of possible permutations for all 4 random keys is $4! = 24$. Therefore, the summation of the wrongly factorized probabilities over all of these 24 permutations equals $24 \cdot \frac{1}{2} \cdot \frac{1}{2} = 6 \neq 1$. In the case of binary variables, the amount of possible combinations would be $2^4 = 16$ and the individual frequency tables would have been of size $2^2 = 4$. If each individual probability would then have been $\frac{1}{4}$ for example, the summation over all possible combinations would be $16 \cdot \frac{1}{4} \cdot \frac{1}{4} = 1$. The reason why there is a difference with permutations is that if $r_0 < r_1$ and $r_2 < r_3$, then there are a multiple of permutations in which this is so instead of a single one. Two examples are $r_0 < r_1 < r_2 < r_3$ and $r_0 < r_2 < r_3 < r_1$. There are 6 of such "indistinguishable" permutations since the total amount of possible permutations is $4! = 24$ and the total amount of groups of permutations that can be made with the 2 shorter permutations is $2! \cdot 2! = 4$. This means that the correct factorization of the probability distribution is given by $\hat{P}_{((0,1),(2,3))}(R_0, R_1, R_2, R_3) = \frac{2!2!}{4!}\hat{P}(R_0, R_1)\hat{P}(R_2, R_3)$. In general, the unconditionally factorized probability distribution is capable of expressing $\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!$ different groups of permutations. On the other hand, the total amount of possible permutations equals $l!$. Therefore, the unconditional factorized random key probability distribution becomes:

$$\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}})(r\langle \boldsymbol{\mathcal{L}} \rangle) = \frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!} \prod_{i=0}^{|\boldsymbol{\nu}|-1} \hat{P}(R\langle \boldsymbol{\nu}_i \rangle)(r\langle \boldsymbol{\nu}_i \rangle) \tag{3}$$

5 Learning unconditional permutation factorizations

To find a good unconditional factorization, often an incremental greedy algorithm is used to minimize the negative log–likelihood of the estimated probability distribution. Furthermore, the negative log–likelihood is usually penalized with increasing complexity. This results in a *penalization metric* to be minimized. A simple but often effective greedy learning algorithm starts from the univariate factorization in which each variable is independent of the others, $\boldsymbol{\nu} = ((0),(1),\ldots,(l-1))$. Each iteration, the eligible operations to be performed on the factorization are the splicing (joining) of two vectors $\boldsymbol{\nu}_{s_0}$ and $\boldsymbol{\nu}_{s_1}$. The splice operation that decreases the penalized negative log–likelihood the most, is actually performed. This process is repeated until there are no splice operations left that further improve the metric. In this paper, we use the *Akaike Information Criterion* (AIC). For a derivation of this metric in the IDEA context, we refer the reader to a more detailed report [3]. This

metric should be minimized. It scores a factorization by its negative log–likelihood, but adds the term $|\boldsymbol{\theta}|$ as a penalization of more complex models. For various applications, the AIC penalization is not strong enough [3, 5]. However, since in this case $|\boldsymbol{\theta}|$ grows factorially, this is most likely not the case. Furthermore, since the frequency tables also grow factorially, it is of no use to allow very large subvectors in the node vector. Therefore, we limit the size of each vector in the node vector to $\kappa = 7$. This is a limit on the maximum allowed order of interaction that can be processed.

In order to use the greedy algorithm, we can search for the largest value of the negative log–likelihood of the current factorization $\boldsymbol{\nu^0}$ *minus* the negative log–likelihood of the candidate factorization $\boldsymbol{\nu^1}$. Since the penalization is additive to this difference, it can be shown [3] that for the AIC metric, this difference should be penalized by adding $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_{s_0}| + |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_{s_1}| - |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_{s_0 \sqcup s_1}|$. Before we derive the required difference, we first note that the likelihood of the multivariate joint pdf from equation 2 equals:

$$\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}(R\langle \boldsymbol{v} \rangle)) = \prod_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \hat{P}(R\langle \boldsymbol{v} \rangle)(r^i\langle \boldsymbol{v} \rangle) \tag{4}$$

Since the multivariate joint pdf from equation 2 is estimated from a discrete frequency table, the estimate is of a maximum likelihood, just as is the case for binary variables. Therefore, it can be shown [4] that the negative log–likelihood of the multivariate joint pdf equals $|\boldsymbol{\mathcal{S}}|$ times the entropy of the multivariate joint pdf:

$$\begin{aligned} -\ln\left(\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}(R\langle \boldsymbol{v} \rangle))\right) = \qquad (5)\\ -\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}(R\langle \boldsymbol{v} \rangle)(r^i\langle \boldsymbol{v} \rangle)\right) = \\ -|\boldsymbol{\mathcal{S}}| \sum_{n=0}^{|\boldsymbol{v}|!-1} \hat{P}(R\langle \boldsymbol{v} \rangle)(\varrho(n)) \ln(\hat{P}(R\langle \boldsymbol{v} \rangle)(\varrho(n))) = \\ |\boldsymbol{\mathcal{S}}| H(\hat{P}(R\langle \boldsymbol{v} \rangle)) \end{aligned}$$

The likelihood that we want to work with, is that of the factorized probability distribution over all of the variables $\boldsymbol{\mathcal{R}}$. The negative log–likelihood of this probability distribution can be written as follows:

$$\begin{aligned} \mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}})) = -\ln\left(\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}}))\right) = \qquad (6)\\ -\ln\left(\prod_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}})(\boldsymbol{r^i})\right) = \\ |\boldsymbol{\mathcal{S}}| \ln(l!) - \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \sum_{i=0}^{|\boldsymbol{\nu}|-1} \ln\left(|\boldsymbol{\nu}_i|!\hat{P}(R\langle \boldsymbol{\nu}_i \rangle)(r^i\langle \boldsymbol{\nu}_i \rangle)\right) \end{aligned}$$

The difference between the negative log–likelihood of $\boldsymbol{\nu^0}$ minus the negative log–likelihood of $\boldsymbol{\nu^1}$ can now be expressed in terms of the entropy for multivariate joint random keys pdfs:

$$\begin{aligned} \ln(\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}_{\boldsymbol{\nu}^1}(\boldsymbol{\mathcal{R}}))) - \ln(\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}_{\boldsymbol{\nu}^0}(\boldsymbol{\mathcal{R}}))) = \qquad (7)\\ \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \Big[\ln\left((|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)!\hat{P}(R\langle \boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1} \rangle)(r^i\langle \boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1} \rangle)\right) \\ -\ln\left(|\boldsymbol{\nu}_{s_0}|!\hat{P}(R\langle \boldsymbol{\nu}_{s_0} \rangle)(r^i\langle \boldsymbol{\nu}_{s_0} \rangle)\right) \\ -\ln\left(|\boldsymbol{\nu}_{s_1}|!\hat{P}(R\langle \boldsymbol{\nu}_{s_1} \rangle)(r^i\langle \boldsymbol{\nu}_{s_1} \rangle)\right)\Big] = \\ |\boldsymbol{\mathcal{S}}| \Big[H(\hat{P}(R\langle \boldsymbol{\nu}_{s_0} \rangle)) + H(\hat{P}(R\langle \boldsymbol{\nu}_{s_1} \rangle)) - \\ H(\hat{P}(R\langle \boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1} \rangle)) + \ln\left(\frac{(|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)!}{|\boldsymbol{\nu}_{s_0}|!|\boldsymbol{\nu}_{s_1}|!}\right)\Big] \end{aligned}$$

Given the above definitions, there is a problem with using *only* the splice operator in the greedy search algorithm. Consider two building blocks $(bb_0^0, bb_1^0, \ldots, bb_4^0)$ and $(bb_0^1, bb_1^1, \ldots, bb_4^1)$ of length 5 that are already converged optimally. Furthermore, without loss of generalization, assume that the optimal permutation is $r_{bb_0^i} < r_{bb_1^i} < \ldots < r_{bb_4^i}$, $i \in \{0, 1\}$. It is then quite likely that $r_{bb_0^0} < r_{bb_4^1}$ also holds for every solution in $\boldsymbol{\mathcal{S}}$. If this is so, a splicing algorithm in its first stages is just as likely to choose to join positions bb_0^0 and bb_4^0, which we want, and positions bb_0^0 and bb_4^1, which we do *not* want. When the splicing algorithm has mixed up the building blocks in this way, using function ϱ to generate new random key sequences is *very* unlikely to generate new correct building blocks. This is a problem that does not occur for binary 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 factorization[1] or a univariate factorization[2]. The reason why this problem occurs, is that at a lower level of interaction, *decision errors* are made. These lower order errors cannot be avoided and only become visible at a higher level of interaction. When regarding interactions of level 5 for instance, it *does* become clear that the building blocks should be separated. The reason for this is that the individual random key sequences of the building blocks are 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.

[1] $\boldsymbol{\nu} = ((bb_0^0, bb_1^0, \ldots, bb_4^0, bb_0^1, bb_1^1, \ldots, bb_4^1))$
[2] $\boldsymbol{\nu} = ((bb_0^0), (bb_1^0), (\ldots), (bb_4^0), (bb_0^1), (bb_1^1), (\ldots), (bb_4^1))$

To overcome this problem, either the size of the sample vector has to increase significantly to reduce the probability of $r_{bb_0^0} < r_{bb_4^1}$ or we require a way to correct for lower order decision errors. The problem can be avoided by the splice 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 scaling 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.

6 Experiments

We have tested the new approaches on the deceptive permutation problems with $l_{BB} = 5$. In all our testing, we used monotonic IDEAs. We used the rule of thumb by Mühlenbein and Mahnig [13] for FDA and set τ to 0.3. All results were averaged over 30 independent runs. Each run was allowed a maximum of 10^6 evaluations. We have applied both pure IDEAs as well as hybrid versions in the form of ICE. We compare the results of the new permutation based algorithms with the recently tested variant of ICE in which the normal pdf was used to find unconditional factorizations. To observe the usefulness of using normal pdfs to detect useful factorizations, we have also used an IDEA in which unconditional factorizations are learned by using normal pdfs but in which new random key sequences are generated by function ϱ. Finally, we have also computed the best possible result by using a fixed factorization. The structure of the problem is best represented when the building blocks are perfectly separated in $\boldsymbol{\nu}$. We call this *perfect mixing* information.

Figures 4 and 5 show the average required amount of function evaluations and the minimal required population size n respectively. Three of all tested algorithms have only been tested with n_{BB} up to 4. They have not been investigated further because of their very bad scaling behavior. Two of those algorithms are the IDEA variants that use normal pdfs for finding unconditional factorizations. The other algorithm is also a pure IDEA, but uses the permutation probability theory. However, this algorithm does not use the swap operator. The bad performance is therefore due to the lower order decision errors that are not corrected as we explained earlier in section 5. The only pure IDEA that does scale up competitively, uses the swap operation and permutation pdfs. We therefore state that using normal pdfs on the real random keys to find the dependencies between the problem variables is less effective than using permutation pdfs.

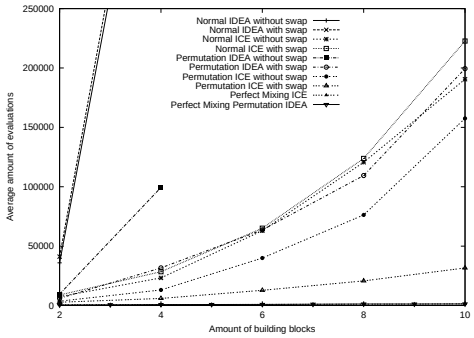

Figure 4: Average amount of evaluations.

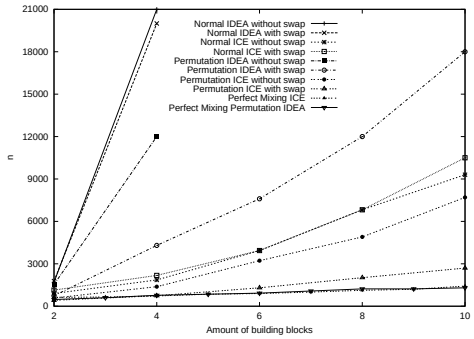

Figure 5: Average population size (n).

The most successful algorithms are all variants of ICE. One of the most important reasons for this is that the crossover operator is less disruptive than using function ϱ. This is especially true if we have an additively decomposable fitness function. Assume for a building block of length 4, that the factorization has spliced no further than $((bb_0, bb_1), (bb_2, bb_3))$. If only one of the parents has a good solution for the building block, the probability that it is crossed over to the offspring is $\frac{1}{4}$. But even if the block has *fully* converged in $\boldsymbol{\mathcal{S}}$, the probability that the optimal block will be constructed using the factorization given above, is only $\frac{2!2!}{4!} = \frac{1}{6}$. A similar argument holds when lower order decision errors have found their way into the final factorization.

Using the swap operator as well as crossover in permutation ICE significantly outperforms all other tested algorithms in terms of the required amount of eval-

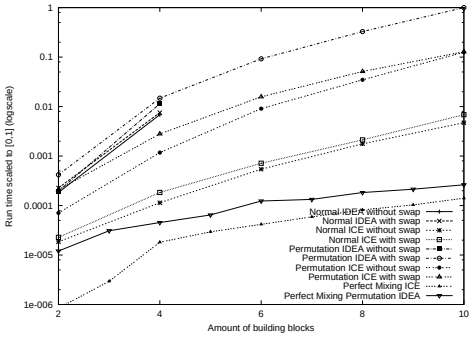

Figure 6: Average actual running time scaled to $[0, 1]$ on a logarithmic scale.

uations and population size. It shows a much better scaling behavior that is quite close to the optimal performance when perfect mixing information is used. There is however a negative side to the use of permutation pdfs. Because the permutation frequency tables are of maximum size $\kappa!$ and because we require $\mathcal{O}(\kappa \log \kappa)$ time to convert random keys to integers, the actual running time of the permutation based algorithms is much larger than when the normal pdfs are used. This can be seen in figure 6 in which the running times scaled to $[0, 1]$ are shown on a logarithmic scale. Because of the $\mathcal{O}(l^3)$ factorization learning algorithm, the performance of the permutation based algorithms in this graph is much worse. At $l = 50$, the best performing normal $\mathbb{ICE}$ runs at a factor of 27.13 times faster than the best performing permutation $\mathbb{ICE}$. Because of the better scaling behavior in evaluations, if l goes up far enough, permutation $\mathbb{ICE}$ will eventually outperform normal $\mathbb{ICE}$. Furthermore, if the fitness function is a highly time consuming simulation, permutation $\mathbb{ICE}$ is preferable.

7 Discussion

We have seen that the swap operator is an effective way to correct for lower order decision errors when optimizing additively decomposable problems. However, this operator will only be successful if the building blocks are decomposable. Assume that the building blocks of the deceptive permutation problem are fully overlapping such that the last random key of block i is the first random key of block $i + 1$. In this case, the optimal permutation is only encoded by completely ascending random key strings. Now assume that a few neighbouring building blocks have converged optimally. This means that all of the involved random keys will always occur in a fully ascending fashion. In order to generate new correct building blocks with probability 1 using function ϱ, all of the involved random key positions will have to be placed in a single joint pdf. However, as the amount of converged building blocks increases, the amount of positions will surely surpass κ. Even if the building blocks are correctly separated at their boundaries, the probability that the right combination of building blocks is sampled, becomes smaller for an increasing amount of building blocks. To overcome this problem, $\mathbb{ICE}$ must clearly be used. An additional problem is that once multiple overlapping building blocks have converged, using the greedy learning algorithm will not be able to separate the building blocks *at any level*. Lower order decision errors cannot even be corrected at a higher level by using the swap operator since there is only one permutation for all the converged building blocks. Therefore, all splice operations of subvectors regarding the converged indices give equally good results according to the negative log–likelihood. Again, using $\mathbb{ICE}$ may overcome this problem since the probability at survival of building blocks is larger. However, since building blocks can still be destroyed, the $\mathbb{ICE}$ strategy is *not optimal* for such difficult overlapping problems.

Once the building blocks have converged, separating them is difficult, especially for overlapping problems. One possible way to overcome this problem, may be by using the distance between the random keys. The need for such information becomes clear especially when we regard *conditional* factorizations. The definitions in this paper readily allow us to search for conditional factorizations. Sampling a new solution for a node v in such a factorization can be done by observing the already sampled values for the parents that v is conditioned on. The value for v will either be smaller or larger than all parent random key values, or it will lie between two subsequent values. For these options, frequency tables can be used to select an option just as is done in the binary case. This can lead to very efficient solving of both the additively decomposable deceptive trap functions as well as the fully overlapping ones because random key values can be sampled *relatively* to the already sampled values in a solution. If we model each building block $(b_0, b_1, \ldots, b_{l_{BB}-1})$ by the conditional factorization in which R_{b_i} is conditioned on $\bigsqcup_{i<j<l_{BB}}(R_{bj})$, the permutation IDEA results in a population size of $n = 750$ and an average amount of evaluations of 6745 for the additive deceptive problem with length $l = 50$. For the fully overlapping deceptive problem of length $l = 40$ and $l_{BB} = 4$, we get $n = 350$ and 2884 evaluations, which is extremely much better than the best obtained results by normal $\mathbb{ICE}$ so far of 10.61 building blocks correct (out of 13), $n = 10^4$ and $481.1 \cdot 10^6$ evaluations. However, it can be expected that the use of a greedy learning algorithm that introduces one conditional dependency at a time, will also have lower order decision error problems. To overcome

these problems, new operators will be required or some effective means of using the random key distance.

We note that in this paper we have only tested the newly proposed approaches on a limited amount of problems. Even though the results are encouraging, verification on other problems is desired. None of the $\mathbb{ICE}$ algorithms have so far been tested on real–life problems such as scheduling. It would be interesting to investigate the performance of the new algorithms and compare them with other EA approaches.

8 Conclusions

In a previous study [5], it was shown that finding and using the structure of permutation problems can aid EAs in optimization. In this paper, we have enhanced the tools for finding this structure by learning it in the space of permutations instead of an embedding real encoding space. By using this structure to exchange the building blocks, we get the $\mathbb{ICE}$ algorithm. $\mathbb{ICE}$ has been shown to efficiently solve permutation problems of a bounded difficulty. With respect to the requirements on the population size and the amount of evaluations, the resulting algorithms outperform previously proposed EAs. However, the time requirements for probabilistic modelling have significantly increased. For lower order to moderate order dimensional permutation optimization problems with non–time consuming fitness functions, normal $\mathbb{ICE}$ is preferable over permutation $\mathbb{ICE}$.

Although good results have been obtained for additively decomposable problems, unconditional factorizations are not well suited for problems with overlapping building blocks. To this end, conditional factorizations seem more appropriate. Preliminary results indicate that IDEAs based on conditional permutation factorizations may yet more effectively find and use the structure of a wider range of permutation problems.

References

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

[2] P.A.N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proc. of the GECCO–1999 Genetic and Evol. Comp. Conference*, pages 60–67. M.K. Pub., 1999

[3] P.A.N. Bosman and D. Thierens. Mixed IDEAs. Utr. Univ. Tech. R. UU–CS–2000–45. ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-45.ps.gz, 2000

[4] P.A.N. Bosman and D. Thierens. Negative log–likelihood and statistical hypothesis testing as the basis of model selection in IDEAs. In A. Feelders, editor, *Proceedings of the Tenth Belgium–Netherlands Conference on Machine Learning*. Tilburg University, 2000

[5] P.A.N. Bosman and D. Thierens. Crossing the road to efficient IDEAs for permutation problems. In *Proc. of the GECCO–2001 Genetic and Evolutionary Computation Conference*. M.K. Pub., 2001 (*To appear*)

[6] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 840–846. Morgan Kaufmann Publishers, 1999

[7] D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. M.K. Pub., 1993

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

[9] H. Kargupta, K. Deb, and D.E. Goldberg. Ordering genetic algorithms and deception. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature – PPSN II*, pages 47–56. Springer, 1992

[10] D. Knazjew. Application of the fast messy genetic algorithm to permutation and scheduling problems. IlliGAL Technical Report 2000022. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/2000022.ps.Z, 2000

[11] D. Knazjew and D.E. Goldberg. Large–scale permutation optimization with the ordering messy genetic algorithm. 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 631–640. Springer, 2000

[12] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization by learning and simulation of bayesian and gaussian networks. Univ. of the Basque Country Tech. R. EHU-KZAA-IK-4/99. http://www.sc.ehu.es/ccwbayes/postscript/kzaa-ik-04-99.ps, 1999

[13] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evol. Comp.*, 7:353–376, 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, 2000

[15] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 525–532. M.K. Pub., 1999

[16] M. Pelikan, D.E. Goldberg, and K. Sastry. Bayesian optimization algorithm, decision graphs and occam's razor. IlliGAL Tech. R. 2000020. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/2000020.ps.Z, 2000

[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 Publishers, 2001