Permutation Optimization by Iterated Estimation of Random Keys Marginal Product Factorizations

Peter A.N. Bosman and Dirk Thierens

Institute of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{*Peter.Bosman*, *Dirk.Thierens*}*@cs.uu.nl*

Abstract. In I$\mathbb{D}\mathbb{E}$As, the probability distribution of a selection of solutions is estimated each generation. From this probability distribution, new solutions are drawn. Through the probability distribution, various relations between problem variables can be exploited to achieve efficient optimization. For permutation optimization, only real valued probability distributions have been applied to a real valued encoding of permutations. In this paper, we present two approaches to estimating marginal product factorized probability distributions in the space of permutations directly. The estimated probability distribution is used to identify crossover positions in a real valued encoding of permutations. The resulting evolutionary algorithm (EA) is capable of more efficient scalable optimization of deceptive permutation problems of a bounded order of difficulty than when real valued probability distributions are used.

1 Introduction

Any reasonable optimization problem has some structure. Using this structure can aid the search for the optimal solution. In black box optimization, this structure is a priori unknown. In order to still be able to exploit problem structure, induction must be performed on previously evaluated solutions.

Holland [9] showed that in the simple GA with one point–crossover, problem structure is only exploited if it is expressed by short substrings of closely located gene positions. If such related genes are not closely located, they are likely to be disrupted by one–point crossover. Thierens and Goldberg [18] showed through the investigation of uniform crossover that the effect of such disruption on problems with non–linear interactions between groups of bits results in an exponential growth of the required population size as the problem length l increases. On the other hand, Harik, Cantú-Paz, Goldberg and Miller [7] showed that for a crossover operator that does not disrupt such building blocks of related genes, the required population size scales with $\sqrt{l}$ instead of exponentially.

In general, to prevent bad scaling behavior of an EA, we need to respect the structure of the optimization problem. Describing the structure of a set of samples can be done by estimating its probability distribution. Subsequently drawing more samples from this probability distribution leads to a statistical inductive type of iterated search. Such algorithms have been shown to give promising results for both binary and real valued spaces [2, 8, 12–17].

Our goal in this paper is to show how this technique can be used for permutation problems. The necessity for problem structure exploitation is found in permutation spaces as well, since permutation problems can be constructed that deceive simple permutation GAs [10] in a similar manner as is done for simple binary GAs [5] on which the simple GA is known to scale–up exponentially [18].

The remainder of this paper is organized as follows. In section 2 we present the representation of permutations that we will work with along with two permutation problems that are used in the experiments. A brief overview of IDEAs and its first application to permutation problems is given in section 3. Section 4 describes the estimation of probability distributions over permutations. We test our new IDEAs in section 5. Some conclusions are drawn in section 6.

2 Random keys and permutation optimization problems

We use the random keys encoding of permutations, which was proposed by Bean [1]. The main advantage of random keys is that no crossover operator can create unfeasible solutions. A random key sequence $\boldsymbol{r}$ is a vector of real values. Each of these real values is usually defined to be in $[0, 1]$. The integer permutation $\boldsymbol{p}$ that is encoded by $\boldsymbol{r}$ can be computed in $\mathcal{O}(|\boldsymbol{r}|\log(|\boldsymbol{r}|))$ time by sorting $\boldsymbol{r}$ in ascending order ($\boldsymbol{r}_{\boldsymbol{p}_0} \leq \boldsymbol{r}_{\boldsymbol{p}_1} \leq \ldots \leq \boldsymbol{r}_{\boldsymbol{p}_{|\boldsymbol{r}|-1}}$), which we denote by $\boldsymbol{p} = \textsc{sort}(\boldsymbol{r})$. As an example, we have that $\textsc{sort}(0.61, 0.51, 0.62, 0.31) = (3, 1, 0, 2)$.

We will test our algorithms on two problems introduced by Knjazew [11]. Both problems are a sum of subfunctions f_{sub} that regard random key substrings:

$$f(\boldsymbol{r}) = \sum_{j=0}^{|\boldsymbol{\iota}|-1} f_{sub}(\boldsymbol{r}_{\boldsymbol{\iota}_j}) \tag{1}$$

In eq. 1, $\boldsymbol{\iota}$ is a vector of *index clusters* that defines the indices of the complete random keys sequence that each subfunction will regard. So $\boldsymbol{r}_{\boldsymbol{\iota}_j}$ indicates the random keys found at positions $(\boldsymbol{\iota}_j)_0, (\boldsymbol{\iota}_j)_1, \ldots, (\boldsymbol{\iota}_j)_{|\boldsymbol{\iota}_j|-1}$. Each random key index appears in at least one index cluster. For example, if $\boldsymbol{\iota} = ((0, 1), (2), (2, 3, 4))$, we have $f(\boldsymbol{r}) = f_{sub}(\boldsymbol{r}_0, \boldsymbol{r}_1) + f_{sub}(\boldsymbol{r}_2) + f_{sub}(\boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4)$.

In the problems that we use in this paper, each index cluster has length $\kappa^{\boldsymbol{\iota}}$. The optimum for the subfunction is a random keys sequence that encodes the permutation $(0, 1, \ldots, \kappa^{\boldsymbol{\iota}} - 1)$. To define the subfunction, a distance measure is used. This distance from any permutation $\boldsymbol{p}$ to the optimum equals $\kappa^{\boldsymbol{\iota}} - |\textsc{lis}(\boldsymbol{p})|$, where $\textsc{lis}(\boldsymbol{p})$ is the *longest increasing subsequence* in $\boldsymbol{p}$. For example, if $\boldsymbol{p} = (1, 2, 0, 4, 3)$, then $\textsc{lis}(\boldsymbol{p}) \in \{(1, 2, 4), (1, 2, 3)\}$. Furthermore, $\kappa^{\boldsymbol{\iota}} - |\textsc{lis}(\boldsymbol{p})| = 5 - 3 = 2$. Note that the reverse permutation $(\kappa^{\boldsymbol{\iota}} - 1, \kappa^{\boldsymbol{\iota}} - 2, \ldots, 0)$ is the only permutation with a distance of $\kappa^{\boldsymbol{\iota}} - 1$. The subfunction is defined as follows:

$$f_{sub}(\boldsymbol{r}) = \begin{cases} 1 - \frac{|\textsc{lis}(\textsc{sort}(\boldsymbol{r}))|}{\kappa^{\boldsymbol{\iota}}} & \text{if } |\textsc{lis}(\textsc{sort}(\boldsymbol{r}))| < \kappa^{\boldsymbol{\iota}} \\ 1 & \text{if } |\textsc{lis}(\textsc{sort}(\boldsymbol{r}))| = \kappa^{\boldsymbol{\iota}} \end{cases} \tag{2}$$

The fully deceptiveness of the subfunction makes the problem hard for any EA that doesn't identify the boundaries of the index clusters [10, 18] and as a result disrupts the non–linear relations between the genes imposed by f_{sub}.

In the first problem that we shall use to experiment with, the index clusters are *mutually disjoint*. This problem is therefore *additively decomposable*. To avoid the possibility that static crossover operators are biased in optimization because the genes contributing to each subfunction are closely located, the locations for each $\boldsymbol{\iota}_j^{loose}$ are chosen *loosely*, meaning as well spread as possible.

$$\boldsymbol{\iota}_j^{loose} = (j, j + |\boldsymbol{\iota}|, j + 2|\boldsymbol{\iota}|, \ldots, j + (\kappa^{\boldsymbol{\iota}} - 1)|\boldsymbol{\iota}|) \tag{3}$$

In the second problem, the j–th index cluster *shares* its first position with index cluster $j-1$ and its last position with index cluster $j+1$. This overlapping property makes the problem *significantly* more difficult as there are no clear index cluster boundaries. For simplicity, the overlapping index cluster vector $\boldsymbol{\iota}^{overlap}$ is encoded tightly. Optimal solutions are now only given by random key sequences $\boldsymbol{r}$ such that $\textsc{sort}(\boldsymbol{r}) = (0, 1, \ldots, l-1)$, where l is the problem length.

$$\boldsymbol{\iota}_j^{overlap} = (j(\kappa^{\boldsymbol{\iota}} - 1), j(\kappa^{\boldsymbol{\iota}} - 1) + 1, j(\kappa^{\boldsymbol{\iota}} - 1) + 2, \ldots, j(\kappa^{\boldsymbol{\iota}} - 1) + \kappa^{\boldsymbol{\iota}} - 1) \tag{4}$$

3 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 [2] and other named variants [8, 12–17].

Problem structure in the form of dependencies between the problem variables, is *induced* from a vector of selected solutions 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 focus on *marginal product* factorizations (mpfs). In the binary and real valued case, an mpf is a product of multivariate joint pdfs. This product is represented by a vector of mutually exclusive vectors of random variable indices, which we call the *node vector* $\boldsymbol{\nu}$. An example in the case of $l = 3$ and binary random variables X_i, is given by $P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{X}}) = P(X_0, X_1)P(X_2)$, meaning $\boldsymbol{\nu} = ((0, 1), (2))$. Once a structure ς is given, the parameters for the multivariate pdfs have to be estimated. The way in which this is done, is predefined on beforehand. Often, this corresponds to a maximum likelihood estimate, such as a frequency count for binary random variables. As a probability distribution can thus be identified using only the structure ς, we denote it by $P_\varsigma(\boldsymbol{\mathcal{Z}})$.

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 $\hat{P}^{\theta_t}_\varsigma(\boldsymbol{\mathcal{Z}})$ 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 [3] as well as by Robles, de Miguel and Larrañaga [15]. However, the study by Bosman and Thierens [3] 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 [3]. This crossover operator reflects the dependency information learned in a factorization. Two parents are first selected at random. In the case of an mpf, 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 mpf, 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 [11]. 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, we propose to induce these dependencies in the space of permutations directly by interpreting the random keys as permutations. This is the topic of the next section.

4 Estimating random keys marginal product factorized probability distributions from data

For each problem variable r_i we introduce a random variable R_i. Since the random keys encode permutations, the symantics of the R_i differ from those of binary or real valued random variables. We let $\boldsymbol{\mathcal{R}} = (R_0, R_1, \ldots, R_{l-1})$ and write the vector of selected samples as $\boldsymbol{\mathcal{S}} = (\boldsymbol{r^0}, \boldsymbol{r^1}, \ldots, \boldsymbol{r^{|\mathcal{S}|-1}})$, $\boldsymbol{r^i} = (r^i_0, r^i_1, \ldots, r^i_{l-1})$.

The multivariate joint pdf over a subset of the random keys $R_{\boldsymbol{v}}$ is defined by the probability at a certain random key subsequence $r_{\boldsymbol{v}}$. It can be computed by counting the frequency in $\boldsymbol{\mathcal{S}}$ of the permutation $\text{SORT}(r_{\boldsymbol{v}})$ represented by $r_{\boldsymbol{v}}$:

$$\hat{P}(R_{\boldsymbol{v}})(r_{\boldsymbol{v}}) = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \text{SORT}((\boldsymbol{r}^{\boldsymbol{i}})_{\boldsymbol{v}}) = \text{SORT}(r_{\boldsymbol{v}}) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

To define the random keys marginal product factorized probability distribution, it should be noted that the size of the alphabet for a multivariate joint factor $\hat{P}(R_{\boldsymbol{\nu}_i})$ is $|\boldsymbol{\nu}_i|!$. Since the individual factors are taken to be *independent* of each other, the alphabet size of the whole mpf is $\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!$. However, the total number of possible permutations equals $l!$. Therefore, to construct a probability distribution over all possible permutations of length l, we must normalize the product of the multivariate marginals $\prod_{i=0}^{|\boldsymbol{\nu}|-1} \hat{P}(R_{\boldsymbol{\nu}_i})(r_{\boldsymbol{\nu}_i})$. To illustrate, assume that $r_0 < r_1$ and $r_2 < r_3$. Then there are $\frac{4!}{2!2!} = 6$ permutations of length 4 in which this is so, such as $r_0 < r_1 < r_2 < r_3$ and $r_0 < r_2 < r_3 < r_1$. This implies that the correct factorization of the probability distribution is given by $\hat{P}_{((0,1),(2,3))}(R_0, R_1, R_2, R_3) = \frac{2!2!}{4!}\hat{P}(R_0, R_1)\hat{P}(R_2, R_3)$. Concluding, the marginal product factorized probability distribution over all l variables becomes:

$$\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}})(r_{\boldsymbol{\mathcal{L}}}) = \frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!} \prod_{i=0}^{|\boldsymbol{\nu}|-1} \hat{P}(R_{\boldsymbol{\nu}_i})(r_{\boldsymbol{\nu}_i}) \tag{6}$$

To find a factorization 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 [2, 8, 12, 14].

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. For the learning of random keys mpfs, we propose three possible operations. The operation that decreases the penalization metric the most, is actually performed. The first operation is a splice operation that replaces $\boldsymbol{\nu}_i$ and $\boldsymbol{\nu}_j$ $(i \neq j)$ with $\boldsymbol{\nu}_i \sqcup \boldsymbol{\nu}_j$, increasing the complexity of the factorization. The second operation is a swap operation in which two factors $\boldsymbol{\nu}_i$ and $\boldsymbol{\nu}_j$ may exchange an index. This operator allows to correct for lower order decision errors and is therefore *always* preferred over the application of a splice operation. The third operation is a *transfer* operation in which an index is removed from one factor $\boldsymbol{\nu}_i$ and added to another factor $\boldsymbol{\nu}_j$. This last operator is able to correct for some additional special cases of lower order decision errors [4]. A metric that has often proved to be successful, is known as the *Bayesian Information Criterion* (BIC). It scores a model by its negative log–likelihood, but adds a penalty term that increases with the model complexity ($|\boldsymbol{\theta}|$) and the size of the sample vector ($|\boldsymbol{\mathcal{S}}|$) [2]:

$$BIC(\mathcal{M}|\boldsymbol{\mathcal{S}}) = \underbrace{-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{R}})(\boldsymbol{r^i})\right)}_{\text{Error}(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{R}})|\boldsymbol{\mathcal{S}})} + \underbrace{\frac{1}{2}\ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|}_{\text{Complexity}(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{R}})|\boldsymbol{\mathcal{S}})} \tag{7}$$

The AIC metric is an alternative to the BIC metric. The AIC metric is similar to the BIC metric, but the complexity term is only given by the number of parameters $|\boldsymbol{\theta}|$. The penalization in the AIC metric is too weak compared to that in the BIC metric to give good results when normal probability distributions are estimated [3]. However, if we use frequency tables to estimate the pdfs in eq. 6, $|\boldsymbol{\theta}|$ grows factorially with an increase of any $\boldsymbol{\nu}_i$. Therefore, the penalization in the AIC will in this case most likely not be too weak. Because of the factorial growth of the number of parameters to be estimated however, we must limit the maximum factor size $\kappa^{\boldsymbol{\nu}}$ in any practical application (we used $\kappa^{\boldsymbol{\nu}} = 7$).

This direct limitation on the maximum order of interaction that can be processed, can be avoided by using *default tables* [6]. 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. 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 can give rise to more parameters than $|\boldsymbol{\mathcal{S}}|$. We may still require $|\boldsymbol{\mathcal{S}}| = \mathcal{O}(\kappa^{\boldsymbol{\iota}}!)$ when there are subproblems with a maximum length of $\kappa^{\boldsymbol{\iota}}$ 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 a important benefit of using local structures in probability distributions. A local structure allows for a more explicit representation of dependencies between values that can be assigned to random variables instead of dependencies between the random variables themselves. As a result, less parameters need to be estimated. Probabilistic models that are capable of expressing more complex dependencies now become eligible for selection when using a penalization metric, whereas otherwise non–local structure models expressing similar dependencies would never have been regarded because of the large number of (redundant) parameters the impose [6]. The use of local structures has been shown by Pelikan and Goldberg [14] to allow for efficient optimization of very difficult hierarchical deceptive optimization problems that exhibit dependencies between combinations of values for large groups of variables.

To use default tables, the random keys for each factor $\boldsymbol{\nu}_i$ are converted into integer permutations. The list of selected permutations is then sorted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|\log(|\boldsymbol{\mathcal{S}}|))$ time and the frequencies are counted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|)$ time. Note that we can't map the random keys to integers for faster sorting because the integers would become too large to efficiently represent as the factor size increases.

An additional operation that can be useful when using crossover on random keys, is *random rescaling*. With random rescaling, a block of random keys that is transferred to an offspring in crossover, is scaled to a subinterval of $[0, 1]$

with probability p_ϱ. If for instance $(0.1, 0.2, 0.3)$ is scaled to $[0.9, 0.95]$, we get $(0.9, 0.925, 0.95)$. Note that this doesn't change the permutation that is encoded. Rescaling allows for dependencies *between* the building blocks to be exploited and increases the chance that they are combined properly. To ensure a large enough number of intervals so that the blocks can be ordered, we set this number to l.

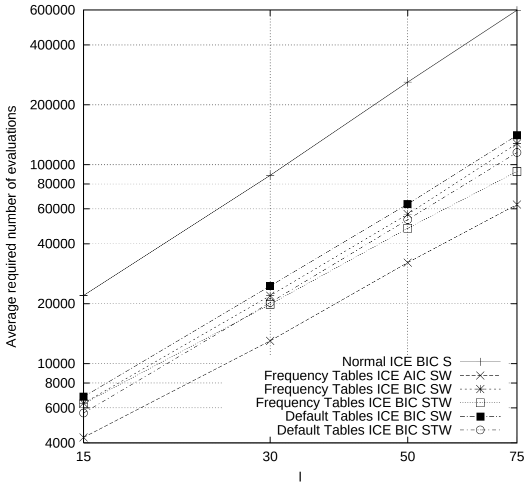
5 Experiments

We have tested the new approach to learning a probability distribution over random keys by using it in monotonic $\mathbb{ICE}$. We applied the algorithms to the additively decomposable deceptive permutation problem with $\kappa^\iota = 5$ and the overlapping problem with $\kappa^\iota = 4$. We used the rule of thumb by Mühlenbein and Mahnig [12] and set τ to 0.3. All results were averaged over 30 runs.

An mpf over random keys as defined in eq. 6, differs from the traditional definition for binary or real valued domains. This difference results in an additional requirement for using default tables for random keys, which is a cutoff value $\xi \in [0, 1]$ indicating the maximum default table length $\xi|\boldsymbol{\mathcal{S}}|$. During the creation of a factorization in the greedy factorization learning algorithm, no operation is allowed to create a factor with a default table longer than $\xi|\boldsymbol{\mathcal{S}}|$. Without this restriction, there would be an premature drift towards large factors. In this paper, we use a cutoff value of $\xi = 0.3$. For a thorough explanation, we refer the interested reader to a specialized technical report [4].

To get a good impression of the impact of different model building choices, we tested a varied ensemble of combinations. Figure 1 shows the scale–up behavior of all tested algorithms to obtain the optimal solution in all of the 30 runs on a log–log scale. A linear relationship on this scale indicates a polynomial scale–up behavior. Similar figures are obtained for the minimally required population size and the running time. The actual scaling coefficients are computed with a least squares line fit. The best scaling behavior is obtained when frequency tables are used in combination with the AIC metric. The use of the transfer operation shows a benefit over using only the splice and swap operations.

In figure 2, the results for the overlapping problem are tabulated. The maximum tested population size was $n \leq 10^5$. The results show the minimal requirements on the algorithms to solve the problem optimally or the performance at $n = 10^5$. No algorithm was capable of optimizing the problem without using *random rescaling*. If random rescaling is used, the overlapping deceptive problem can be solved optimally for $n \leq 10^5$ quite efficiently. However, we have now applied random rescaling, knowing that overlapping subproblems exist. Since we normally are not aware of this information, it is of great interest to see the implication of random rescaling on solving the additively decomposable problems. If the structure of additively decomposable problems is correctly found, introducing random rescaling should not matter. However, if the index cluster boundaries are not completely found, random rescaling may introduce additional disruptiveness to the crossover operation. This can indeed be seen in figure 1 as the scaling coefficients worsen as p_ϱ increases. The variant of $\mathbb{ICE}$ that uses nor-



Additively decomposable problem						
Algorithm				$\mathcal{O}(l^{x.xx})$		
$\mathbb{ICE}$	Oper.	Metric	p_ϱ	n	$\overline{\text{Eval.}}$	$\overline{\text{Time}}$
N	S	BIC	0.0	1.61	2.06	3.70
N	S	BIC	0.1	1.71	2.79	4.33
N	S	BIC	0.5	—	—	—
P	SW	AIC	0.0	1.23	1.68	4.07
P	SW	AIC	0.1	1.42	2.14	4.58
P	SW	AIC	0.5	1.64	4.14	7.07
P	SW	BIC	0.0	1.34	1.82	4.26
P	SW	BIC	0.1	1.43	1.69	4.51
P	SW	BIC	0.5	2.13	3.87	6.84
P	SWT	BIC	0.0	1.23	1.68	4.08
P	SWT	BIC	0.1	1.33	1.71	4.01
P	SWT	BIC	0.5	1.75	2.77	5.61
PDT	SW	BIC	0.0	1.25	1.87	4.71
PDT	SW	BIC	0.1	1.71	2.21	5.13
PDT	SW	BIC	0.5	3.80	4.36	8.19
PDT	SWT	BIC	0.0	1.32	1.87	4.80
PDT	SWT	BIC	0.1	1.81	2.28	4.88
PDT	SWT	BIC	0.5	3.32	4.45	8.29

Fig. 1. Results on the additively decomposable deceptive problem ($\kappa^{\iota} = 5$). On the left, the required average number of evaluations as a function of the problem length l on a log–log scale are shown. The straight lines indicate polynomial scale–up behavior. On the right, the polynomial scaling coefficients with respect to the population size n, the average required evaluations and the average actual running time for the tested $\mathbb{ICE}$ variants are shown. The algorithms are indicated by N for the use of normal pdfs, P for the use of permutation pdfs using frequency tables and PDT for the use of permutation pdfs using default tables. The factorization search operations are indicated by S for the splice operation, W for the swap operation and T for the transfer operation.

Additively overlapping problem									
Algorithm				$\lvert\boldsymbol{\iota}\rvert = 3$, $\boldsymbol{\kappa^{\iota}} = 4$			$\lvert\boldsymbol{\iota}\rvert = 6$, $\boldsymbol{\kappa^{\iota}} = 4$		
$\mathbb{ICE}$	Oper.	Metric	p_ϱ	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$
N	S	BIC	0.0	100000	2.17	643674	100000	4.07	1579354
N	S	BIC	0.1	100000	2.70	1066013	100000	4.87	2585035
N	S	BIC	0.5	100000	2.87	4111057	100000	5.40	45423314
P	SW	AIC	0.0	21000	3.00	222893	100000	5.20	1654022
P	SW	AIC	0.1	1900	3.00	21022	16000	6.00	328135
P	SW	AIC	0.5	1500	3.00	18982	11000	6.00	442256
P	SW	BIC	0.0	21000	3.00	250336	100000	5.00	1224018
P	SW	BIC	0.1	1600	3.00	19088	22000	6.00	576436
P	SW	BIC	0.5	1500	3.00	22520	18000	6.00	1116807
P	SWT	BIC	0.0	19000	3.00	210534	100000	5.00	1259755
P	SWT	BIC	0.1	1400	3.00	16998	20000	6.00	538037
P	SWT	BIC	0.5	1200	3.00	16759	12000	6.00	798334
PDT	SW	BIC	0.0	70000	3.00	684962	100000	5.00	1374018
PDT	SW	BIC	0.1	4750	3.00	42999	18000	6.00	302783
PDT	SW	BIC	0.5	1500	3.00	13586	80000	6.00	1116019
PDT	SWT	BIC	0.0	50000	3.00	473512	100000	5.00	1416019
PDT	SWT	BIC	0.1	4000	3.00	35371	13000	6.00	202301
PDT	SWT	BIC	0.5	1500	3.00	12641	70000	6.00	1050020

Fig. 2. Results for all algorithms on the overlapping deceptive permutation problem with $\kappa^{\iota} = 4$; $\overline{\text{subs}}$ stands for the average number of subfunctions solved optimally. The abbreviations are the same as those in figure 1.

mal distributions is not capable of solving the problem at all for $p_\varrho = 0.5$. For $p_\varrho = 0.1$, the scaling behavior is not effected too much, so in general we would suggest the use of a small p_ϱ. However, it would be interesting to see whether the results can be improved by setting p_ϱ adaptively by for instance looking at the rate of success of applying random rescaling and by changing p_ϱ accordingly.

The overhead in the scaling results are smaller for the AIC metric than for the BIC metric. This is a result of the smaller penalization in the AIC metric, which results in larger factors for smaller population sizes. This can lead to overly complex models, for which reason the BIC metric is often preferred. In our case however, it introduces a useful bias for correctly finding the index clusters. The best scaling results are expected to be obtained when the AIC metric is used in combination with the splice, swap and transfer operation.

6 Discussion and conclusions

We have proposed a new tool for finding and using the structure of permutation problems in evolutionary optimization by estimating mpfs in the space of permutations. By using this probabilistic information to exchange the random keys that encode the permutations in a crossover operator, we obtain the $\mathbb{ICE}$ algorithm. $\mathbb{ICE}$ has been shown to scale up efficiently on deceptive additively decomposable permutation problems of a bounded difficulty and to furthermore give promising results on difficult overlapping deceptive permutation problems.

The use of default tables indicates a slightly larger overall requirement on the computational resources used by $\mathbb{ICE}$. However, the scaling behavior is not effected too much. The advantage of default tables in that they allow more complex models to compete in model selection, is more likely to stand out on hierarchical deceptive problems. Empirical verification of this expectation, combined with the results presented in this paper, would lead us to conclude that the use of default tables with a small probability at using random rescaling is the most effective allround optimization variant of $\mathbb{ICE}$ for permutation problems.

Although good results have been obtained, mpfs are not well suited for problems with overlapping building blocks. To this end, Bayesian factorizations in which gene positions may be dependent on other gene positions are likely to be more appropriate. Using the tools proposed in this paper, Bayesian factorizations may be learned, although it is to be expected that the use of a greedy learning algorithm that introduces one dependency at a time, will also have lower order decision error problems. To overcome this problem, new operators will be required or some effective means of using the distances between random keys.

Finally, although our results are encouraging, we have only used a limited number of test problems. An interesting next effort would be to investigate the performance on real–world problems and provide a comparison with other EAs.

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. 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.
3. P. A. N. Bosman and D. Thierens. Crossing the road to efficient IDEAs for permutation problems. In L. Spector et al., editor, *Proc. of the Genetic and Evolutionary Computation Conf. – GECCO–2001*, pages 219–226. Morgan Kaufmann, 2001.
4. P. A. N. Bosman and D. Thierens. Random keys on ICE : Marginal product factorized probability distributions in permutation optimization. Utrecht University Technical Report UU–CS–2002–xx., 2002.
5. K. Deb and D. E. Goldberg. Sufficient conditions for deception in arbitrary binary functions. *Annals of Mathematics and Artificial Intelligence*, 10(4):385–408, 1994.
6. N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In E. Horvitz and F. Jensen, editors, *Proc. of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 252–262. Morgan Kaufmann, 1996.
7. G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.
8. G. Harik and D. E. Goldberg. Linkage learning through probabilistic expression. *Comp. methods in applied mechanics and engineering*, 186:295–310, 2000.
9. John H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
10. 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 Verlag, 1992.
11. D. Knjazew. Application of the fast messy genetic algorithm to permutation and scheduling problems. IlliGAL Technical Report 2000022, 2000.
12. H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evol. Comp.*, 7(4):353–376, 1999.
13. 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.
14. 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.
15. 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.
16. R. Santana, A. Ochoa, and M. R. Soto. The mixture of trees factorized distribution algorithm. In L. Spector et al., editor, *Proc. of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 543–550. Morgan Kaufmann, 2001.
17. 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.
18. D. Thierens and D.E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proceedings of the fifth conference on Genetic Algorithms*, pages 38–45. Morgan Kaufmann, 1993.