Crossing the Road to Efficient IDEAs for Permutation Problems

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

In this paper, we introduce the ICE framework in which crossover from genetic algorithms (GAs) is incorporated in iterated density estimation evolutionary algorithms (IDEAs). We focus on permutation optimization problems and show how pure continuous IDEAs can be applied using the random keys representation. The problems that are hereby encountered, motivate the use of ICE. As a result, permutation linkage information is effectively processed, resulting in efficient optimization of deceptive permutation problems of a bounded order. Experiments show that ICE outperforms pure continuous IDEAs. Furthermore, we show that ICE gives insight into how new IDEAs can be designed that efficiently work directly in the permutation search space.

1 Introduction

Finding and using the structure of the fitness landscape can aid evolutionary algorithms (EAs) in optimization. One approach to doing so, is by learning a probabilistic model from the selected samples and by using it in sampling new solutions. Such algorithms have been successfully applied to various problems in the case of discrete (binary) variables [9, 14, 15, 16, 17] and continuous (real) variables [4, 7, 13].

In this paper, we focus on the class of permutation optimization problems. This class of problems is very interesting. On the practical side, important real life problems such as scheduling and the traveling salesman are within this class. On the theoretical side, the search space grows *factorially* as the amount of decision variables increases. Furthermore, the solution space consists of permutations, which is fundamentally different from that of the binary or real space that most EAs have been designed for. EAs that directly work on permutations are usually equipped with specialized crossover operators that ensure feasibility of the solutions. An exception is the RKGA by Bean [1] in which the solutions are coded in the real space in such a way that all crossover operators generate feasible solutions. The RKGA has obtained good results especially in the field of scheduling.

With the exception of the OmeGA by Knazjew and Goldberg [12], no attempts have been made to learn and use the structure of permutation optimization problems. The OmeGA is essentially a *fast messy GA* [8] (fmGA) that has been adapted to work with the same coding as used in the RKGA. The OmeGA has been shown to effectively solve deceptive problems of bounded difficulty [11]. Furthermore, it significantly outperforms the RKGA, indicating the usefulness of finding and using problem structure.

The IDEA by Bosman and Thierens [4] is a framework for EAs in which probabilistic models are used. It has mostly been used to focus on continuous representations and optimization problems. Since the RKGA introduces a real coding of permutations, continuous IDEAs can be directly applied. However, since the permutation space is discrete, such an application is likely to encounter problems. In this paper, we investigate the performance of continuous IDEAs on permutation problems as a first attempt to design efficient probabilistic model building EAs for permutation problems. Furthermore, we explain the problems with such an approach and propose a solution for them by introducing crossover. The resulting algorithm shows efficient scaling behavior on problems of bounded complexity. The results indicate that IDEAs can be designed that efficiently work directly in the space of permutations.

The remainder of this paper is organized as follows. In section 2 we discuss the IDEA framework. Next,

we go over the coding that is used in the RKGA in section 3 and present the permutation problems that we test the algorithms on. Subsequently, we test the continuous IDEAs in section 4. In section 5, we propose to use crossover in the IDEA to overcome some of the problems of the continuous IDEA on permutation problems. We test the resulting EAs in section 6. In section 7 we reflect on the computational requirements of learning and using probabilistic models in evolutionary optimization. We discuss future research in section 8 and present our conclusions in section 9.

2 The IDEA framework

The IDEA framework is a general definition of *Iterated Density Estimation Evolutionary Algorithms* that has mostly been used to focus on continuous representations and optimization problems. Let $\boldsymbol{\mathcal{L}} = (0, 1, \ldots, l-1)$. The rationale behind the framework can be explained by assuming to have an l–dimensional optimization problem $C(y\langle\boldsymbol{\mathcal{L}}\rangle) = C((y_0, y_1, \ldots, y_{l-1}))$. Without loss of generality, we assume that we want to *minimize* $C(y\langle\boldsymbol{\mathcal{L}}\rangle)$. With every problem variable y_i, we associate a continuous random variable Y_i. Without any prior information on $C(y\langle\boldsymbol{\mathcal{L}}\rangle)$, we might as well assume a uniform distribution over $\boldsymbol{\mathcal{Y}} = (Y_0, Y_1, \ldots, Y_{l-1})$. Therefore, we generate an initial (population) vector of n samples at random. Now we let $P^{\theta}(\boldsymbol{\mathcal{Y}})$ be a probability distribution that is uniform over all vectors $y\langle\boldsymbol{\mathcal{L}}\rangle$ with $C(y\langle\boldsymbol{\mathcal{L}}\rangle) \leq \theta$. Sampling from $P^{\theta}(\boldsymbol{\mathcal{Y}})$ gives more samples that evaluate to a value below θ. Moreover, if we know $\theta^* = \min_{y\langle\boldsymbol{\mathcal{L}}\rangle}\{C(y\langle\boldsymbol{\mathcal{L}}\rangle)\}$, a single sample gives an optimal solution. To use this in an iterated algorithm, we select $\lfloor \tau n \rfloor$ samples in each iteration t and let θ_t be the worst selected sample cost. We then estimate the distribution of the selected samples and obtain an estimate $\hat{P}^{\theta_t}(\boldsymbol{\mathcal{Y}})$ as an approximation to the true 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 [4].

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.

One of the most important parts in the IDEA is the estimation of the probability distribution of the selected samples. One way of achieving this, is by finding a factorized probability distribution, which is a product of probability density functions (pdfs).

If only multivariate joint pdfs are used, the factorization becomes a *marginal product model*. We also call this an *unconditional factorization*. For the factorization to be valid, each multivariate joint pdf must describe a unique set of variables. To this end, we define the *node vector* $\boldsymbol{\nu}$ to be a vector of mutually exclusive vectors that each hold the indices of the variables that are contained in a single multivariate joint pdf. The union of all of these mutually exclusive vectors is exactly equal to $\boldsymbol{\mathcal{L}}$. For example, a valid unconditional factorization for $l = 5$ is $\boldsymbol{\nu} = ((0, 4), (1), (3, 2))$.

If multivariate conditional pdfs are used, the product consists of exactly l factors. For each variable Y_i, we have exactly one factor $P(Y_i|Y\langle\pi(i)\rangle)$. The parent variables $Y\langle\pi(i)\rangle$ that Y_i is conditioned on, are indicated by a function $\pi(\cdot)$ that returns a vector $\pi(i) = (\pi(i)_0, \pi(i)_1, \ldots, \pi(i)_{|\pi(i)|-1})$. This can be identified with a directed graph by introducing a node i for every random variable Y_i. Furthermore, an arc $Y_i \rightarrow Y_j$ in the graph represents the fact that $i \in \pi(j)$. To be able to sample from the resulting factorized probability distribution, values must have been sampled for the parents of Y_i before sampling a value for Y_i itself. To ensure this, we use a vector of ordering variable indices $\boldsymbol{\omega} = \omega\langle\boldsymbol{\mathcal{L}}\rangle$. We can now enforce that while scanning the variables in the order $Y_{\omega_{l-1}}, Y_{\omega_{l-2}}, \ldots, Y_{\omega_0}$, the variables that some variable is conditioned on, will already have been regarded. This ordering can be found by performing a topological sort on the factorization graph. A valid conditional factorization for $l = 5$ is $\forall_i [\omega_i = i], \pi(0) = (2), \pi(1) = (2, 3, 4), \pi(2) = (3, 4), \pi(3) = (4), \pi(4) = (0)$.

To find a good factorization, we use an incremental algorithm that starts from the univariate factorization in which each variable is independent from the others. For unconditional factorizations this means $\boldsymbol{\nu} = ((0), (1), \ldots, (l-1))$. For conditional factorizations we have $|\pi(i)| = 0, i \in \boldsymbol{\mathcal{L}}$. Each iteration, a single operation is performed on the factorization that increases some metric the most. In the unconditional case, the only operation we allow is merging two vectors. In the conditional case, we only allow the addition of a variable to the vector of parents of another variable. This corresponds to adding arcs to the factorization graph. However, to ensure that the conditional factorization is still valid, an arc is only allowed to be added if it does not introduce any cycles. If no operation that increases the metric can be performed anymore, the factorization search algorithm stops. In this paper, we use two metrics that

should be minimized, which are the *Akaike Information Criterion* (AIC) and the *Bayesian Information Criterion* (BIC). For a derivation of these metrics in the IDEA context, we refer the reader to a more detailed report [5]. The metrics are intended to provide a useful tradeoff between complexity and goodness of fit. Both metrics initially score a factorization by the negative log–likelihood of the factorized probability distribution. The AIC metric penalizes complexity by adding the amount of parameters $|\boldsymbol{\theta}|$ that has to be fit. Note that the operations on factorizations that we have proposed, always increase the amount of parameters. The BIC metric is parameterized by a regularization parameter λ that determines the amount of penalization of more complex models in order to favor more simple models. The penalization in the BIC metric increases logarithmically with the size of the sample vector $\lambda \ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|$. In this paper, we have used $\lambda = \frac{1}{2}$. As the size of the sample vector increases, the BIC metric has a stronger penalization than does the AIC metric. Since we have fixed λ to $\frac{1}{2}$, this is the case for any practical population size.

3 Random keys and permutation problems

Permutations can be encoded in the real space using random keys. In this section, we briefly go over this encoding and present some permutation problems of tunable bounded difficulty.

3.1 Permutation encoding

The encoding of permutations by random keys 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 $\boldsymbol{\mathcal{L}}$ is assigned a value (key) from some real domain, which is usually taken to be $[0, 1]$. Subsequently, the numbers in $\boldsymbol{\mathcal{L}}$ are sorted on the keys to get the resulting permutation. For example, the random key string $(0.25, 0.1, 0.9)$ represents the ordering $(1, 0, 2)$. This decoding requires $\mathcal{O}(l \log l)$ time.

3.2 Problems of bounded order

Knazjew [11] has introduced a general deceptive permutation problem. The advantage of this problem over most test problems is that the interactions between the random keys are restricted to be of a certain order l_{BB}. A further advantage is that it is defined for any l_{BB}, unlike for instance the deceptive permutation problems by Kargupta, Deb and Goldberg [10].

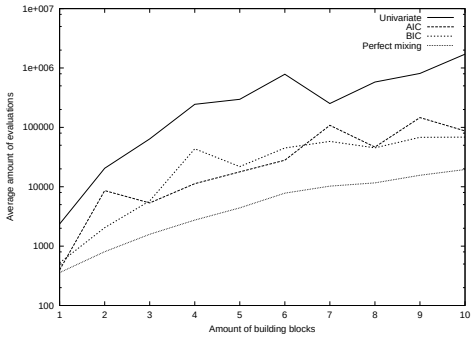

Figure 1: Amount of evaluations for the continuous IDEA, $l_{BB} = 4$.

Knazjew's deceptive permutation problem is defined using a distance between two permutations. This distance is defined as the minimum number of elements in one string to be moved to obtain the other string. Furthermore, the optimum is the trivial permutation $(0, 1, \ldots, l_{BB} - 1)$. The distance from any permutation $\boldsymbol{y}$ to the optimum equals $l_{BB} - |\textsc{lis}(\boldsymbol{y})|$, where $\textsc{lis}(\boldsymbol{y})$ is the longest increasing subsequence in $\boldsymbol{y}$. For example, if $\boldsymbol{y} = (4, 0, 3, 1, 2)$, then $\textsc{lis}(\boldsymbol{y}) = (0, 1, 2)$ and $l_{BB} - |\textsc{lis}(\boldsymbol{y})| = 5 - 3 = 2$. The two elements to move are of course 4 and 3. 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:

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

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 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 [10]. This makes the problem hard for any optimizer that doesn't identify which random key positions together constitute a building block [3]. Furthermore, because of the loose coding, simple crossover schemes such as one point crossover are not effective either.

4 IDEAs based on normal pdfs for permutation problems

Since random key strings are real numbers, we can directly apply continuous IDEAs to permutation prob-

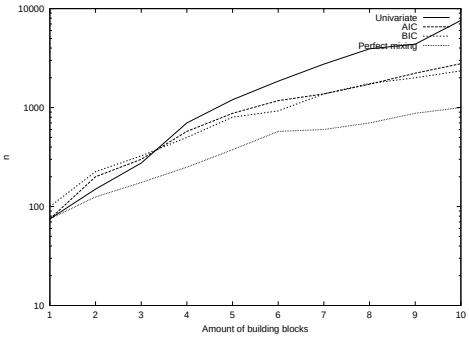

Figure 2: Required population size for the continuous IDEA, $l_{BB} = 4$.

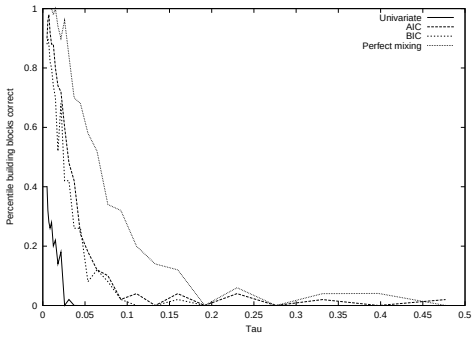

Figure 3: Optimization performance of the continuous IDEA for increasing τ, $l_{BB} = n_{BB} = 5$, $\lfloor \tau n \rfloor = 250$.

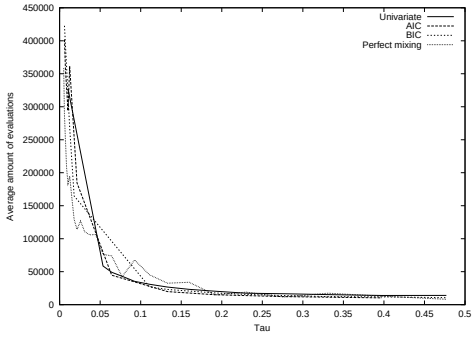

Figure 4: Function evaluations of the continuous IDEA for increasing τ, $l_{BB} = n_{BB} = 5$, $\lfloor \tau n \rfloor = 250$.

lems. In this paper, we use the normal pdf for the pdfs implied by the factorization. We refer the reader to previous work [4] for implementation details.

In all our testing, we used monotonic IDEAs. We used the rule of thumb by Mühlenbein and Mahnig [14] for FDA and set τ to 0.3. All results were averaged over 30 independent runs. As a measure of efficiency, we use the average amount of required function evaluations.

In figures 1 and 2, the average amount of evaluations and the required population size are shown respectively on a logarithmic scale for $n_{BB} \in \{1, 2, \ldots 10\}$ and $l_{BB} = 4$. It is clear that using the univariate factorization in which each variable is taken independent of all of the others, scales up significantly worse than when problem structure is exploited. The structure of the problem is best represented in an unconditional factorization in which the building blocks are perfectly separated in the node vector $\boldsymbol{\nu}$. We call this *perfect mixing* information. Using this structure can be seen to scale up polynomially. There is no obvious difference between the AIC or BIC search metric here.

Note that the amount of instances for a permutation of $l_{BB} = 4$ is $4! = 24$, which still tractable. If we move to $l_{BB} = 5$ however, the amount of instances already becomes 120, which significantly increases the problem difficulty. This is reflected by the fact that the continuous IDEAs are unable to solve the deceptive problems for $n_{BB} \in \{3, 4, \ldots 10\}$ with $n \leq 1.0 \cdot 10^5$. In the case of perfect mixing information, we got 2.0 BBs on average using $3.1 \cdot 10^6$ evaluations and $n = 3.0 \cdot 10^4$. The OmeGA significantly outperforms the continuous IDEA, which points out that the continuous IDEA approach less efficiently processes the building blocks.

5 The ICE framework

The results of the continuous IDEA cannot compete for instance with the OmeGA. Both algorithms however attempt to find and use the relations between the random keys. The main problem with the continuous IDEA is the density estimation of and sampling from a normal pdf. It has some limitations with respect to non–linearity and multimodality. Furthermore, the permutation space is inbedded in the real space, meaning that the actual search space is a (special) discretization of $[0, 1]^l$. For an l_{BB}–dimensional f_{BB} function, the optimum and the suboptimum are contained in a convex region consisting of $\frac{100}{l_{BB}!}\%$ of the $[0, 1]^{l_{BB}}$ hypercube. Furthermore, these two regions are separated by the single line on which every random key has an identical value. Finally, the other types of building blocks have the highest fitness in the neighborhood of the suboptimal block. This means that, especially as l_{BB} goes up, the geometrical approximation of the normal distribution will have great difficulty to represent for instance the trade–off between the suboptimal block and the optimal block. To increase their separability, the selection pressure should increase or the population size should increase. For instance, in order to isolate the optimal building block in a random population, a selection percentage of $\tau = \frac{100}{l_{BB}!}\%$ is required. The performance for different values of τ is displayed in figures 3 and 4. In these figures, the selection size $\lfloor \tau n \rfloor$ is kept constant. The

rationale for this is that the goodness of a normal pdf fit does no longer significantly increase if the amount of samples increases. The optimization performance increases if the selection pressure increases. However, this comes at the expense of the amount of required evaluations. Therefore, there must be some range for the best choice for τ in the sense of required amount of evaluations to reach the optimum. In section 7, we investigate this further.

To overcome the problems with the normal pdf in the real space with respect to fitting the embedded permutation space, the random key strings should be interpreted in the permutation space. By sampling from the continuous normal pdf, we introduce a lot of redundancy, since the random key string $(0.1, 0.2, 0.3)$ codes the same permutation as $(0.91, 0.99, 0.999)$. To cope with this redundancy, instead of sampling *new* building blocks from a normal pdf, we propose to *mix* them using crossover. Only combinations of the initial strings are thereby generated, just as is done in GAs.

The way in which crossover is done, determines the rate of success. First, we note that *if* the building blocks are exchanged between parents as a whole, the information is mixed in the most efficient mixing manner, resolving the redundancy problem. However, on beforehand we generally don't know the location and size of the building blocks. This information is called *linkage information*. To find this information, we rely on the remainder of the IDEA framework to find a structure that contains this linkage information. For instance, if we have an unconditional factorization, we are given groups of random keys that should be processed in a multivariate joint pdf. Therefore, these random keys should be processed together as a block. We hope that these blocks are a good approximation of the true building blocks in the problem.

Before copying a block to the offspring, each block may be transformed using a function $\varrho(\cdot)$. This function rescales the random keys to a subinterval of $[0, 1]$ with probability p_ϱ. If we for instance scale $(0.1, 0.2, 0.3)$ to $[0.9, 0.95]$, we get $(0.9, 0.925, 0.95)$. Note that this doesn't change the permutation that is encoded. The optimization problem can require a relative ordering of the building blocks. Without rescaling, we have to rely on the random key combinations that are generated initially. Rescaling the blocks increases the probability that they will be combined properly. To ensure a large enough amount of intervals so that the blocks themselves can be ordered, we set this amount to l.

We call the framework for the resulting algorithms $\mathbb{ICE}$ (IDEA *Induced Chromosome Elements Exchanger*). Its definition equals that of the IDEA, with the exception of the creation of offspring. In $\mathbb{ICE}$, this is not done by sampling, but by randomly selecting two parents from the selected samples and crossing over blocks in the solutions. Which blocks we actually perform crossover with, is determined by the type of factorization. Since the resulting algorithm uses crossover, it can validly be argued that we have designed a GA. The specialty of this GA is that it attempts to learn linkage information for permutations and use this linkage information in a linkage preserving crossover operator.

$\mathbb{ICE}$
1 $par_0 \leftarrow$ RANDOM($\{0, 1, \ldots, \lfloor \tau n \rfloor - 1\}$)
2 $par_1 \leftarrow$ RANDOM($\{0, 1, \ldots, \lfloor \tau n \rfloor - 1\} - \{par_0\}$)
3 $\boldsymbol{\mathcal{B}} \leftarrow$ CROSSOVERBLOCKS(ς)
4 ***for*** $i \leftarrow 0$ ***to*** $\lvert\boldsymbol{\mathcal{B}}\rvert - 1$ ***do***
4.1 $par \leftarrow$ RANDOM($\{par_0, par_1\}$)
4.2 ***for*** $j \leftarrow 0$ ***to*** $\lvert\boldsymbol{\mathcal{B}}_i\rvert - 1$ ***do***
4.2.1 $\boldsymbol{off}_{(\boldsymbol{\mathcal{B}}_i)_j} \leftarrow \varrho((\boldsymbol{y}^{\boldsymbol{par}})_{(\boldsymbol{\mathcal{B}}_i)_j})$
5 RETURN($\boldsymbol{off}$)

6 Specific crossover operators in $\mathbb{ICE}$

Having introduced the $\mathbb{ICE}$ framework, we test it on the deceptive permutation functions. First however, we elaborate on how we have selected the blocks to be crossed over in our experiments.

6.1 Crossover operators

Conditional factorizations allow for more precise probabilistic modelling because unconditional factorizations can be expressed using conditional factorizations, but not vice versa. Therefore, it is interesting to base crossover operators on both types of factorizations.

6.1.1 Unconditional: block mixing

By crossing over the random keys based on the node vector $\boldsymbol{\nu}$, we attempt to directly exchange and mix the important blocks of information. This results in an approach similar to the ECGA by Harik [9].

CROSSOVERBLOCKS($\boldsymbol{\nu}$)
1 RETURN($\boldsymbol{\nu}$)

6.1.2 Conditional: position biased TPX

To use conditional factorizations, we learn a chain of dependencies in which random keys that are important together, are placed close to each other. Subsequently, we can apply for instance two point crossover such that the linkage information in the chain is respected. This approach was recognized earlier [3] to be interesting for processing linkage information. To find a chain,

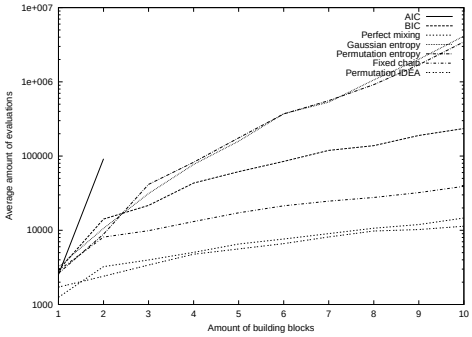

Figure 5: Amount of evaluations for $\mathbb{ICE}$, $l_{BB} = 5$.

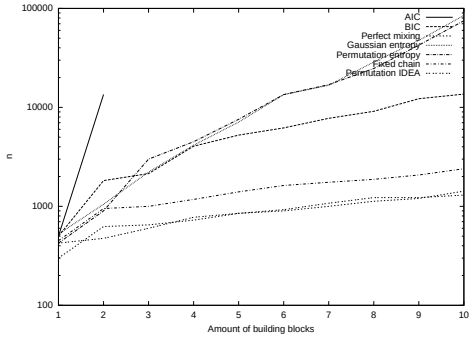

Figure 6: Required population size for $\mathbb{ICE}$, $l_{BB} = 5$.

the greedy entropy algorithm in MIMIC [2] can be used. The entropy, which equals the average negative log–likelihood for normal pdfs [6], can be computed using the normal pdf as we have done for all continuous IDEAs, but we can also use permutation entropy. This amounts each time to finding the unselected node Y_{i+1} that occurs more often in one ordering than in the other with respect to the lastly selected node Y_i (maximize $|\hat{P}(Y_i < Y_{i+1}) - \hat{P}(Y_{i+1} < Y_i)|$). The resulting blocks to be exchanged, can now be found as follows:

CrossoverBlocks$((\pi, \boldsymbol{\omega}))$
1 $pos_0 \leftarrow$ Random$(\{0, 1, \ldots, l\})$
2 $pos_1 \leftarrow$ Random$(\{0, 1, \ldots, l\})$
3 ***if*** $pos_1 > pos_0$ ***then***
  3.1 $pos_0 \leftrightarrow pos_1$
4 Return$(((\omega_0, \omega_1, \ldots, \omega_{(pos_0-1)}),$
  $(\omega_{pos_0}, \omega_{(pos_0+1)}, \ldots, \omega_{(pos_1-1)}),$
  $(\omega_{pos_1}, \omega_{(pos_1+1)}, \ldots, \omega_{l-1})))$

6.2 Results

$\mathbb{ICE}$ gives significantly better results (figures 5 and 6). All tested algorithms, with the exception of unconditional factorizations in combination with the AIC metric, were able to solve all tested problems up to $n_{BB} = 10$ for $l_{BB} = 5$, with population sizes (well) below $1 \cdot 10^5$. The entry of the permutation IDEA is explained in section 8. Because the deceptive problem is better separable if the selection pressure increases, it was again observed that the performance increases if τ decreases for a fixed $\lfloor \tau n \rfloor$ in a similar fashion as observed for continuous IDEAs (figures 3 and 4).

Since the AIC metric did not penalize the likelihood effectively, it resulted in unconditional factorizations that combined entire building blocks to sometimes form the complete joint factorization, which does not lead to efficient exploration of the permutation space. In this case there is thus a preference for the BIC metric. All approaches can be seen to scale up polynomially. However, the chain approaches that use search metrics scale up significantly worse. There is not much to choose between the normal entropy or the permutation entropy. If the right structure is found (fixed chain), the building blocks are propagated effectively using position biased TPX. However, it is hard to find the right second order linkage information, as these approaches clearly scale up a lot worse for increasing n_{BB}. We conclude in a similar fashion as has been done for binary spaces [3], that finding and using lower order linkage information is less efficient than finding and using higher order linkage information.

From the results shown so far, it seems that the best approach is to use an unconditional factorization with the BIC information criterion. However, we have not tested it yet on problems with conditional dependencies. To test this, along with the influence of random rescaling, we have used a difficult permutation problem with overlapping building blocks. All of the building blocks are now coded sequentially instead of loosely. For $n_{BB} = 3$ and $l_{BB} = 4$, the individual building blocks are found at positions $(0, 1, 2, 3)$, $(3, 4, 5, 6)$ and $(6, 7, 8, 9)$, giving $l = 10$. Note that the sole optimal solution is the complete trivial permutation of length l. We tested the algorithms on a problem with $l = 40$, giving $n_{BB} = 13$ at $l_{BB} = 4$. We tested population sizes up to $n = 10^5$ and allowed for a maximum of 10^9 evaluations. The results are shown in figure 7. It becomes clear that using conditional dependencies in a chain can be efficient, especially when random rescaling is used. However, the results also show again that learning the chain to use for position biased TPX does not lead to the best results. Furthermore, introducing random rescaling *does* improve the results of the unconditional factorization. The best reported results of the OmeGA are approximately $\overline{BBs} = 9.60$ after $300 \cdot 10^6$ evaluations.

7 A note on the running times

Algorithms that build and use probabilistic models take up more time every iteration as the complexity

Unconditional				
f	p_e	$n \cdot 10^4$	$\overline{BBs}$	$\overline{Eval} \cdot 10^6$
BIC	0	7.5	8.23	2.0
BIC	0.5	10.0	10.61	481.1

Conditional				
f	p_e	$n \cdot 10^4$	$\overline{BBs}$	$\overline{Eval} \cdot 10^6$
NE	0	9.5	7.60	3.2
NE	0.5	4.5	7.23	3.9
FIX	0	8.5	11.17	2.9
FIX	0.5	0.1	13.00	0.023

Figure 7: Results on the overlapping deceptive permutation problem, $l_{BB} = 4$, $n_{BB} = 13$, ($l = 40$, NE = normal pdf entropy, FIX = fixed optimal chain).

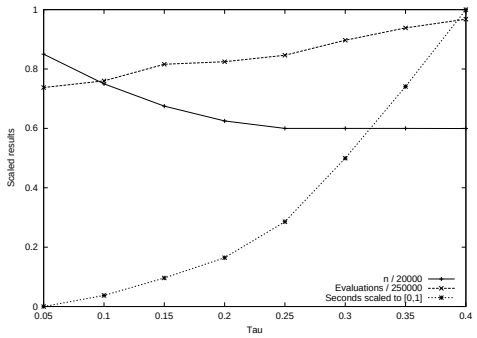

Figure 8: $\mathbb{ICE}$, $l_{BB} = 5$, $n_{BB} = 10$, 100% successrate.

of the models increases. It is therefore important to be aware of the running time next to the amount of function evaluations. If we have a very costly evaluation function, this is of less importance. However, there are plenty practical optimization problems, such as knapsack and traveling salesman, that can be evaluated efficiently and easily result in a very large dimensionality l. Since the learning of higher order factorizations often scales as $\mathcal{O}(l^3)$, exploiting problem structure by learning factorizations will not result in fast algorithms as l increases.

In this paper, we have used the rule of thumb by Mühlenbein and Mahnig [14] for FDA. However, this rule of thumb is based on observations on non–deceptive binary problems. Furthermore, the elitism in FDA is restricted to the single best solution of the previous generation. In figure 8, we have plotted the most efficient result of $\mathbb{ICE}$ using the BIC search metric on unconditional factorizations for different values of τ when each of 30 independent runs reached all building blocks correct. When only the amount of evaluations or the actual complete running time is important, $\tau = 0.3$ is clearly not optimal. Since the separability of the deceptive problem becomes easier when a greater selection pressure is applied, this is not surprising. For the best memory–computation time trade–off, $\tau \in [0.15, 0.25]$ seems to be an effective choice. Concluding, we suggest from empirical observervations to use $\tau \leq 0.25$ in future experiments.

8 Discussion and future research

In this paper, we have used penalization metrics to guide the search for a good factorization. The difficulty with such an approach is choosing the right amount of regularization. The amount of regularization can be seen as a parameter that defines how much time the algorithm is allowed to spend on model building. In a way, this is a substitute for limiting the maximum order of interactions in the factorizations. However, using a metric in addition influences the decision of which operations on the factorization are beneficial. Seen in this way, a penalization metric is a practical approach to using exact statistical hypothesis tests to determine whether some operation is truly beneficial. Even though the use of statistical hypothesis tests is exact, it is also of less practical use because of its computational requirements [6]. Therefore, it is practically more interesting to use a metric as an approximation. Still, it should be noted that selecting the settings for such a metric is always subject to user experience.

We note that in this paper we have only run tests over a limited amount of problems. Even though the results are encouraging, verification on other problems is desired. It would for instance be interesting to see how $\mathbb{ICE}$ algorithms perform on real life scheduling problems as opposed to other EA approaches.

The $\mathbb{ICE}$ framework has pointed out that effective EAs for permutation problems can be constructed if we find and use linkage information in the permutation space directly. Since crossover on permutations is used in $\mathbb{ICE}$, this is evidence that efficient ID$\mathbb{E}$As 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. This is sufficient to also process conditional factorizations, since a conditional probability is a quotient of two multivariate joint probabilities. In order to estimate the probability of a set of k random keys, we can count the frequency of a certain permutation instance. This means that we require a frequency table of minimum size $k!$. To generate such a table of minimum size, we need to map permutations onto integers. This can be done in $\mathcal{O}(k \log k)$ time whereas a new random key sequence can be made from an integer in $\mathcal{O}(k)$ time. Given this correspondence, the frequencies of the possible permutations can be counted from the selected samples. Subsequently, new samples can be drawn in a similar fashion as is done for binary variables.

In figures 5 and 6, the results of testing a permutation based IDEA with the best possible fixed unconditional factorization, are plotted. At this point, the development of the permutation based IDEA only allows fixed factorizations as input as is the case for the FDA [14]. However, the results in this section indicate that IDEAs may be constructed by learning factorizations and thereby no longer assuming a priori knowledge on the optimization problem. The results of the permutation IDEA are slightly better than those of ICE with perfect mixing information. However, we have provided the permutation IDEA with the perfect a priori structure information. It would be interesting to see if a permutation IDEA that learns this structure can outperform ICE on various problems.

9 Conclusions

Finding and using problem structure can aid EAs. We have shown that this is also the case when random keys are used to tackle permutation problems. Furthermore, finding and using a structure in which a multiple of variables interact, is superior to using only second order linkage information when the optimization problem contains higher order interactions.

We have shown by introducing ICE that by mixing blocks of random keys, permutation problems of a bounded difficulty can be solved efficiently. To this end, we have used continuous IDEAs to find the structure of the optimization problem together with crossover from GAs to mix the building blocks. By doing so, the EA directly explores the permutation space, which significantly increases its efficiency. This is an indication that IDEAs that are designed to work directly in the space of permutations may effectively find and use the structure 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] J.S. De Bonet, C. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing*, 9, 1996

[3] 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

[4] P.A.N. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. 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 767–776. Springer, 2000

[5] 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

[6] 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 Dutch–Netherlands Conference on Machine Learning*. Tilburg University, 2000

[7] 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

[8] 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

[9] G. Harik. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Tech. R. 99010. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99010.ps.Z, 1999

[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, 1992

[11] 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

[12] 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

[13] 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

[14] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evol. Comp.*, 7:353–376, 1999

[15] 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

[16] 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

[17] 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.