Factoradic Representation for Permutation Optimisation

Olivier Regnier-Coudert and John McCall

IDEAS Research Institute, Robert Gordon University, Aberdeen, UK
{o.regnier-coudert,j.mccall}@rgu.ac.uk

Abstract. It is known that different classes of permutation problems are more easily solved by selecting a suitable representation. In particular, permutation representations suitable for Estimation of Distribution algorithms (EDAs) are known to present several challenges. Therefore, it is of interest to investigate novel representations and their properties. In this paper, we present a study of the factoradic representation which offers new modelling insights through the use of three algorithmic frameworks, a Genetic Algorithm (GA) and two EDAs. Four classic permutation benchmark problems are used to evaluate the factoradic-based algorithms in comparison with published work with other representations. Our experiments demonstrate that the factoradic representation is a competitive approach to apply to permutation problems. EDAs and more specifically, univariate EDAs show the most robust performance on the benchmarks studied. The factoradic representation also leads to better performance than adaptations of EDAs for continuous spaces, overall similar performance to integer-based EDAs and occasionally matches results of specialised EDAs, justifying further study.

Keywords: Estimation of Distribution Algorithms, Factoradics, Permutation.

1 Introduction

The permutation representation is widely used to model solutions to optimisation problems. Although it is often seen as a natural way to represent solutions, it also appears to be a challenging domain to model because of alleles being interconnected. Recent work has highlighted this challenge for Estimation of Distribution Algorithms (EDAs) and proposed solutions that model the space of permutations by means of specialised distribution models [1].

A different approach to overcome challenges encountered when handling permutations is to introduce alternative genotypes. In Evolutionary Algorithms (EAs), the term *genotype* is often used to describe the domain searched by the algorithms, that is the search space on which operators are applied. In order to assess solutions, a phenotype is required. The *phenotype* represents the domain in which a solution can be evaluated, or in other words, a domain that can be read by the fitness function. Not only does using alternative genotypes allow

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 332–341, 2014.

some problems to be modelled efficiently by EAs, but it may also map a problem to a domain which is more adapted to these algorithms. Consequently, it has been shown that using many representations within the same search procedure may yield improved results by balancing out between the biases introduced by each representation [2].

With respect to permutations, the random key (RK) genotype has widely been used [3]. Yet, it is known to display some features that may inhibit the search in some contexts [4]. There also exist in the literature mentions of an alternative genotype for permutations based on the factorial numbering system, also referred to as factoradics [5]. Despite interesting features, the factoradic genotype has been little studied by the EA community. The present paper proposes directions to using the factoradic representation for optimisation. The paper aims to understand whether the factoradic representation can support search in EAs and identify the contexts in which it is more likely to do so, through the investigation of different specialised operators and algorithmic frameworks including a Genetic Algorithm (GA), and two distinct EDAs.

2 Factoradic Representation

The factoradic system is a numbering system of dimension n, which uniquely represents each number between 0 and $n! - 1$ as a string of factoradic digits. Each position i, $i \in [0, n-1]$ can be assigned a digit taking a value between 0 and i. The base of each position increases with i and so does its place value, i.e. the size of the factorial. Thus, the place value at position i is $i!$. The factoradic $a_{(!)}$ can be transformed into its decimal form $a_{(10)}$ as follows:

$$a_{(10)} = \sum_{i=0}^{n-1} a_{(!)_i} \times i!, \tag{1}$$

where $a_{(!)_i}$ represents the i-th element of $a_{(!)}$. The potential of factoradics goes beyond the simple numbering system as it represents a way to easily represent permutations. For example, the factoradic *422100* denotes the permutation where the 4th, 2nd, 2nd, 1st, 0th and 0th items are drawn successively without replacement from the set of items. Figure 1 illustrates how this factoradic number represents the permutation *423105*.

The factoradic representation allows representation of a permutation by a string of integers of similar size, in which each digit is a number in $[0, i]$. In addition, it introduces different weights between positions. These characteristics of factoradics allow a straightforward application of EA and more precisely EDA techniques in the permutation domain without losing problem properties. To our knowledge, most of the applications of factoradics in EAs have focused on Particle Swarm Optimisation (PSO) to turn permutations into a usable form for the algorithms [6]. Factoradics have also proved useful in allowing restriction of the search to sub spaces [5]. We refer the reader to the latter study for a more detailed description of factoradics.

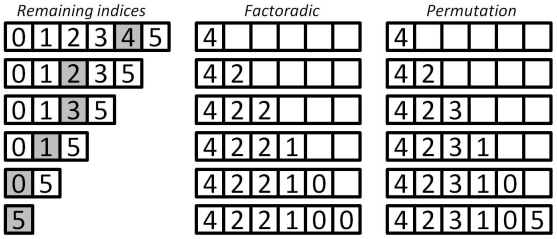


Fig. 1. Mapping from factoradic to permutation

We consider simple operators for the factoradic representation. We note first that standard single-point, multi-point and uniform crossover can be applied to factoradics without alteration. Mutation operators require more care because of the variable limit on the ith digit. We present three mutation operators suitable for the factoradic representation. First, the point mutation (PM) only affects one allele of the mutated solution. However, because of the characteristics of the factoradic representation, the position of the allele influences the amount of disruption to the solution. Hence, PM needs to be defined in conjunction with a mutation distance as defined in [7] for permutations. The mutation distance d denotes the position of the gene to mutate. Note that the gene at position zero can only take the zero value and is thus never considered during operations. An allele is mutated by sampling randomly its value from the range $[0, d]$. The second mutation defined for the factoradic domain is the multi-point mutation (MPM), which performs a PM on all alleles at a position of similar or lower value than the specified mutation distance. Consequently, MPM is expected to be more disruptive than the simple PM. Finally and in order to offer an even more disruptive operator, the random multi-point mutation (RMPM) is introduced. RMPM selects at random d alleles to mutate, regardless of their order.

3 Factoradic Algorithms

Factoradic Genetic Algorithm. The basic concept of the GA developed for the experiments is presented in Algorithm 1. ϵ denotes the size of the elitism, while crossover and mutation rates are referred to as α and β. Starting from an initial randomly generated population *pop*, the GA copies the best solutions according to ϵ, select solutions par_1 and par_2 for recombination and performs successively crossover and mutation with respect to α and β. Generated solutions are added to the new population pop_{new} until it reaches the population size, in which case it replaces the old population before being re-evaluated.

Factoradic Univariate Estimation of Distribution Algorithm. An EDA is based on the concept of evaluating a population of solutions and building a model from a selected subset of this population. This model can then be used to sample new solutions. In a univariate EDA, we construct a fully factorised

Algorithm 1. *GA*

```
Generate and evaluate pop
for each generation g do
  pop_new = ∅
  Add ϵ best solutions to pop_new
  repeat
    Select parents par_1 and par_2
    Generate offspring off by applying crossover to par_1 and par_2, with probability
    α or by copying par_1 with probability (1 − α)
    Apply mutation to off with probability β
    Add off to pop_new
  until (|pop_new| = |pop|)
  pop = pop_new
  Evaluate pop
end for
Return pop
```

probabilistic model which is sampled as a set of independent marginals. Algorithm 2 describes the univariate EDA used in the present study, based on the Population-Based Incremental Learning algorithm (PBIL) [8], but applied to the factoradic representation. First, the model $\mathcal{M}(i,j)$ is initialised with uniform probabilities. $\mathcal{M}(i,j)$ essentially gathers the marginal probability for each item j to be in position i. A population *pop* is generated at random and evaluated. At each generation g, a subset of *pop*, pop_{sel} is selected and the model is updated. This is done using the relative frequency of each item j at each position i in pop_{sel}. Note that the notation $pop_{sel}(k)$ is used to denote the kth solution of pop_{sel}. A model $\mathcal{M}_{temp}(i,j)$ is first created, considering only the frequencies obtained from the current population. This model is then used to update the previous model $\mathcal{M}(i,j)$. The learning rate γ defines how conservative the update is. A high γ results in the model being mostly based on the current population, while a low γ implies that the model keeps a lot of features from the previous generation. Note that setting γ to 1 results in the algorithm to be the Univariate Marginal Distribution Algorithm (UMDA) [9], where the model used at each generation is only built from the information obtained from the current population. Once updated, the model is sampled to generate the new population and the process repeated over several generations.

Factoradic COMpetitive Mutating Agents. The COMMA framework [7] evolves a population of agents. Each agent is assigned a solution whose fitness is used to rank agents within the population. COMMA generates a distribution of solutions spaced within a disruption distance of each agent' solutions. This distribution is geometrically sampled by means of mutation operators to produce a new solution for each agent. As illustrated in Algorithm 3 for minimisation optimisation, the principle of COMMA is to apply different operators to the agents according to their rank in order to perform both exploration and exploitation of

Algorithm 2. *PBIL*

Initialize model $\mathcal{M}(i,j)$ with uniform probabilities
Generate and evaluate *pop*
for each generation g **do**
  Select pop_{sel} from *pop*
  for each index i, $i < n$ **do**
    for each item j, $j < i$ **do**
      $\mathcal{M}_{temp}(i,j) = \frac{\sum_{k=0}^{|pop_{sel}|} x_i}{|pop_{sel}|}$, with $\begin{cases} x_i = 1, \text{ if } pop_{sel}(k) = j \\ x_i = 0, \text{ otherwise} \end{cases}$
      $\mathcal{M}(i,j) = \gamma \mathcal{M}_{temp}(i,j) + (1-\gamma)\mathcal{M}(i,j)$
    end for
  end for
  $pop_{new} = \varnothing$
  repeat
    Sample solution from $\mathcal{M}(i,j)$ and add to pop_{new}
  until $(|pop_{new}| = |pop|)$
  $pop = pop_{new}$
  Evaluate *pop*
end for

the search space. Applying such operators allows to sample new solutions more or less distant in the search space from the agents' solutions. The combinations of solutions and operators represent the model in COMMA. In the case of the factoradic implementation, the operators defined in Section 2 are adapted because they present a notion of mutation distance. COMMA operates as follows. For each position pos_j in the population *pop* sorted in ascending order, a mutation distance d_j is set such that for two agents at positions e and f, $d_e \leq d_f$ if $e < f$. Each agent a_i is initially assigned a random solution s_i. The population is then sorted by fitness. At each generation, each agent mutates s_i using the distance $dist_i \in [1, d_r]$ defined according to its position r in the population. Note that if the boolean parameter $fixedDistance$ is true, $dist_i = d_r$ If the mutated solution s_{new} has a better fitness than s_i , a_i replaces s_i with s_{new}.

4 Experiments

4.1 Test Problems

Travelling Salesman Problem. Based on a given set of k cities and a matrix of distances d_{ij} between all pairs of cities $\{i,j\}$, the TSP aims to determine a shortest possible route r that visits each city exactly once. The route may start at any city, but should end where it started. Hence r is a vector of length k. Formally and using r_a to denote the a-th city in r, the TSP is expressed as:

$$min\{(\sum_{a=0}^{k-1} d_{r_a, r_{a+1}}) + d_{r_k, r_0}\} \tag{2}$$

Algorithm 3. *COMMA* (for minimisation)

Initialize *pop* of σ agents with random solutions, distance vector d of size σ
repeat
  Sort *pop* by fitness in descending order
  for each agent $a_i, i \in [0, \sigma - 1]$ **do**
    Get position r of a_i in *pop*
    if $fixedDistance$ **then**
      $dist_i = d_r$
    else
      Select $dist_i$ with uniform probability from $[1, d_r]$
    end if
    Sample new solution s_{new} with fitness fit_{new} by mutating s_i with distance $dist_i$
    if $fit_{new} < fit_i$ **then**
      Assign $s_i = s_{new}$
    end if
  end for
until Stopping condition met

Permutation Flowshop Scheduling Problem. In the PFSP [10], a set of jobs is given that need to be run on a set of machines. Each of the jobs has to be processed on every machine exactly once and only one job can be handled by a given machine at a given time. It is assumed that each job is processed on the machines in a set order and that the time required to process jobs varies between jobs and between machines. The objective of the PFSP is to minimize the time of completion of the last submitted job on the last machine. The general expression to calculate the time of completion c_{π_i} of a given job π_i on a machine j, given its corresponding processing time $t_{\pi_i,j}$ is given in (3). The fitness of a sequence of jobs represented as the permutation π can be derived as in (4), where n and m respectively stand for the total number of jobs and machines.

$$c_{\pi_i,j} = max\{c_{\pi_{i-1},j}, c_{\pi_i,j-1}\} + t_{\pi_i,j} \tag{3}$$

$$c_\pi = c_{\pi_n,m} \tag{4}$$

Quadratic Assignment Problem. In QAP [11], n facilities are to be assigned to n locations in such way that the total amount of resources being transfered between locations is minimized. Flows $f_{a,b}$ between all pairs of facilities (a, b) and distances $d_{l(a),l(b)}$ between all pairs of locations $l(a), l(b)$ are known. Mathematically the problem can be formulated as (5).

$$min\{\sum_{a,b} f_{a,b} d_{l(a),l(b)}\} \tag{5}$$

Linear Ordering Problem. The aim of the LOP is to find a simultaneous permutation ω of the rows and columns of a matrix $D = (d_{ij})$ that maximizes the sum of the superdiagonal entries. Formally, its objective is defined as follows.

$$max\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{\omega_i,\omega_j}\} \tag{6}$$

4.2 Experimental Settings

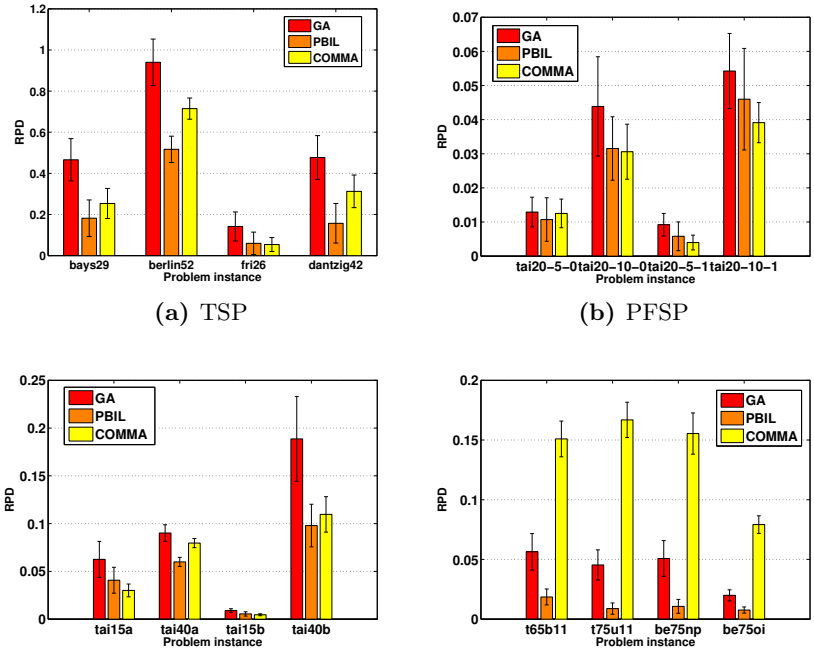
Because of the wide range of EDAs for permutation problems considered, [12] was chosen as the basis for comparison. A similar limit on fitness evaluations was set, that is $1000n^2$ for each problem of size n and 10 runs were needed to compute each result set. In order to reduce the importance given to parameter setting, default parameter values were initially chosen and the algorithms run with every possible combination. Default values are given in Table 1. Note that crossover and mutation rates were respectively set to 0.9 and 0.1. Algorithm performance was measured by best fitness found at the end of the run and by computing the relative percentage deviation (RPD) to the known optimum, as described in [1]. For all comparisons, statistical significance (95% confidence interval) was measured by means of unpaired t-test, applying Bonferroni correction.

Table 1. Default parameter values

Parameter	Default values	Parameter	Default values
pop size	50, 100, 500, 1000	tournament size	0.1, 0.25, 0.5
elitism	0, 1	selection ratio	0.1, 0.25, 0.5, 0.75
mutation	PM, MPM, RMPM	learning rate	0.5, 0.7, 0.9, 1
crossover	1-point, 2-point	fixed distance	true, false

5 Results and Discussion

Suitability of Frameworks for Factoradics. Figure 2 shows the RPD of all methods on each problem. Over all instances, COMMA and PBIL show the best performance, with the exception of TSP. On PFSP and QAP, COMMA and PBIL are the most robust methods, although COMMA presents smaller standard deviations than PBIL. LOP is the problem on which the biggest difference in performance is observed. While COMMA shows relatively poor results, PBIL is significantly better than the other methods. Overall, PBIL appears as a good compromise to handle the factoradic representation across problems. It was also observed that setting high learning rates brought enhanced results. Consequently and as can be seen in Table 2, UMDA often outperforms PBIL.



(a) TSP **(b)** PFSP

(c) QAP **(d)** LOP

Fig. 2. RPD of the factoradic frameworks on the four problems

Suitability of Factoradics for the Selected Problems. To evaluate the suitability of the factoradic representation for permutation optimisation, the results of the best algorithm with the most efficient settings is compared with the suite of methods investigated in [12] for each problem. Results are presented in Table 2, showing the best factoradic framework and the fitness of the best obtained solution. Table 2 also shows for each problem the EDAs that outperform the factoradic methods and the EDAs that match their results. Overall, the performance obtained using factoradics is close to the one of the best integer-based EDAs used for permutation optimisation, that is UMDA, MIMIC and $EBNA_{BIC}$, described as providing good solutions. Continuous EDAs, such as $UMDA_c$ and $EGNA_{ee}$, generally show poor performance on permutation problems. As one might expect of a more natural representation, the factoradic implementations exhibit solutions of greater quality. Experiments from [12] showed that specialised EDAs are the most efficient and more precisely the EDAs using edge and node histogram models, EHBSA and NHBSA. Although the comparison with these algorithms shows that factoradic methods do not always match their results, there exist problems where performances are of the same magnitude. Also note that the IDEA-ICE permutation-based EDA never exhibits better outcome than the factoradic frameworks.

Finally, the recursive EDA (REDA) and OmeGA, a GA based on the RK representation, are generally behind the proposed methods. Direct comparison between the factoradic GA and OmeGA shows that the two algorithms perform at the same level on four instances, mostly on PFSP. However, the factoradic GA outperforms OmeGA to a significant extent on all LOP and TSP instances and most of the QAP ones. This comparison highlights the advantage of using factoradics over RK in a GA.

Table 2. Results from unpaired t-tests. A ✓symbol denotes the algorithms whose performance is not significantly different from the best factoradic implementation on the problem. ■ shows algorithms whose results outperform those of the factoradic algorithms. Empty cells show methods that are outperformed by implementations using the factoradic representation.

Problem	Best Algo.	Best Solution	$UMDA$	$MIMIC$	$EBNA_{BIC}$	$TREE$	$UMDA_c$	$EGNA_{ee}$	$IDEA-ICE$	$EHBSA_{WT}$	$EHBSA_{WO}$	$NHBSA_{WT}$	$NHBSA_{WO}$	$REDA_{UMDA}$	$REDA_{MIMIC}$	$OmeGA$
TSP-bays29	PBIL	2387.4 (179.5)	■	✓	✓					■	■	■	■			
TSP-berlin52	PBIL	11440.7 (481.6)	■	✓	■					■	■	✓	■			
TSP-fri26	COMMA	982.3 (38.8)	✓		✓		✓	✓		■	■	✓	✓		✓	
TSP-dantzig42	PBIL	805.3 (72.9)	✓		✓		■	■	✓	■	■		✓		■	
PFSP-tai20-5-0	PBIL	1291.7 (8.2)	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓
PFSP-tai20-10-0	COMMA	1630.4 (12.8)	✓	✓	✓					■	✓	■	■			
PFSP-tai20-5-1	COMMA	1364.4 (2.9)	✓	✓		✓			✓	✓	✓	✓	✓	✓		✓
PFSP-tai20-10-1	COMMA	1723.9 (9.8)	✓	✓	✓					■	✓	■	■			
QAP-tai15a	COMMA	399889 (2610)	✓	✓	✓	✓				✓		■	✓			
QAP-tai40a	PBIL	3327464 (14966)	■	■	■							✓	■			
QAP-tai15b	COMMA	52002721 (54819)	✓	✓	✓	✓				✓		■	■			
QAP-tai40b	UMDA	699677162 (14205322)	✓	✓	■	✓				■	✓	■	✓			
LOP-t65b11	UMDA	350134 (2379)	✓	✓	✓					■		■	■			
LOP-be75np	UMDA	709328 (4182)	✓	✓	✓					■		■	■			
LOP-be75oi	PBIL	110323 (287)	✓	✓	✓					■		■	■			

6 Conclusions

In this paper, we have presented the factoradic representation as a genotype for permutations, along with frameworks that can be employed to make use of it. Experiments on benchmark problems have shown that the factoradic representation is suitable for permutation optimisation, especially when used within algorithms such as univariate EDAs. Comparison with other studies has demonstrated that algorithms using factoradics present matching performance with integer-based EDAs and can compete with some specialised EDAs. Future work should focus on building a deeper understanding of the relation between factoradics and

problem characteristics. Alternative ways to model factoradics such as tree-based approaches could also be explored. Finally, given the promising results of PBIL, the development of multivariate factoradic-based EDAs represents an interesting avenue for further research.

References

1. Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. IEEE Transactions on Evolutionary Computation (2013)
2. Schnier, T., Yao, X.: Using multiple representations in evolutionary algorithms. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, pp. 479–486. IEEE (2000)
3. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 6(2), 154–160 (1994)
4. Ashlock, D.: Evolutionary computation for modeling and optimization. Springer (2006)
5. Mehdi, M.: Parallel hybrid optimization methods for permutation based problems. PhD thesis, Université des Sciences et Technologie de Lille (2011)
6. Samarghandi, H., ElMekkawy, T.Y.: A meta-heuristic approach for solving the no-wait flow-shop problem. International Journal of Production Research 50(24), 7313–7326 (2012)
7. Regnier-Coudert, O., McCall, J., Ayodele, M.: Geometric-based sampling for permutation optimization. In: Proceeding of the 2013 Annual Conference on Genetic and Evolutionary Computation Conference, pp. 399–406. ACM (2013)
8. Baluja, S.: Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University (1994)
9. Mühlenbein, H.: The equation for response to selection and its use for prediction. Evolutionary Computation 5(3), 303–346 (1997)
10. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 165(2), 479–494 (2005)
11. Lawler, E.L.: The quadratic assignment problem. Management science 9(4), 586–599 (1963)
12. Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A.: A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. Progress in Artificial Intelligence 1, 103–117 (2012)