Alternative Representations for Codifying Solutions in Permutation-Based Problems

Mikel Malagon
University of the Basque Country
Donostia-San Sebastian, Spain
mmalagon002@ikasle.ehu.eus

Ekhine Irurozki
Basque Center for Applied Mathematics
Bilbao, Spain
eirurozki@bcamath.org

Josu Ceberio
University of the Basque Country
Donostia-San Sebastian, Spain
josu.ceberio@ehu.es

***Abstract*—Since their introduction, Estimation of Distribution Algorithms (EDAs) have proved to be very competitive algorithms to solve many optimization problems. However, despite recent developments, in the case of permutation-based combinatorial optimization problems, there are still many aspects that deserve further research. One of them is the influence of the codification employed to represent the solutions on the overall performance of the algorithm. When considering classical EDAs, optimizing permutation problems is challenging, and specific mechanisms are needed to hold the restrictions associated with the permutation nature of solutions.**

In this paper, in addition to the permutation-vector codification, we investigate alternative representations to describe solutions of permutation problems in the context of EDAs. In order to evaluate their influence, we adopted a classical EDA and conducted an experimental study on two different permutation problems and representations for codifying solutions. The results revealed a narrow relationship between the type of combinatorial problem optimized and the selected representation used to codify its solutions. Moreover, the results point out that choosing the appropriate representation to codify solutions of the given permutation problem is critical for the performance of the algorithm.

***Index Terms*—estimation of distribution algorithm, permutation, codification, inversion vector, UMDA**

I. Introduction

In optimization, problems can be divided into two categories depending on the space they are defined on: continuous or combinatorial problems. In continuous problems, solutions are defined over a real-valued space $\mathbb{R}^n$, n being the dimension of the problem. In contrast, in combinatorial problems, the search space of solutions is described as a finite set of objects.

In the last few decades, Evolutionary Algorithms (EAs) have been a popular tool to solving optimization problems, and have been postulated as a powerful option when approaching either continuous or combinatorial problems. As their name suggests, EAs are inspired by the evolution of species in nature. Their ability to approach complex, noisy or not completely defined optimization problems makes them suitable for very diverse problems [19]. Although numerous types of EAs exist, the most popular kind are Genetic Algorithms (GAs) [8]. GAs iterate over two steps: firstly, a set of solutions is selected as parents, then offspring are created from the parents by means of crossover and mutation. This process is repeated until a maximum number of iterations is reached or some other stopping criteria are met.

However, the selection of the right parameters for a GA is sometimes very difficult, and it can be considered an optimization problem itself. As a response to this drawback, in the last decades, multiple improvements have been proposed leading to different metaheuristic paradigms. To name but a few, we can find Ant Colony Optimization [13], Particle Swarm Optimization [17] or Differential Evolution [25]. In this paper, we are particularly interested in another metaheuristic paradigm proposed for the first time in [23], and later extensively investigated by [19], [21], we refer to Estimation of Distribution Algorithms (EDAs).

Unlike GAs, instead of applying crossover and mutation operators to create offspring solutions, EDAs first build a probability model from the set of selected solutions that defines a probability distribution on the solutions of the search space. Then, in a second step, new solutions are obtained by sampling the probability distribution estimated in the previous step. This procedure allows EDAs to efficiently capture and exploit the information of the variables of the problem in order to create better solutions. This characteristic makes EDAs very flexible and, as a consequence, applicable to many optimization problems, including either continuous or combinatorial problems. On the contrary, the performance of EDAs relies completely on the ability of the probability model to capture and preserve the dependencies among the variables that make a solution be of high quality. Nevertheless, as noted in [26], the chosen codification can greatly influence the final results of the EDA, regardless of the class of probability model selected.

When approaching permutation-based problems, due to the permutation nature of solutions, there is a constraint related to the codification of solutions that the algorithm needs to hold in order to create feasible solutions [7]. In this type of problems, solutions are usually described as permutations of n items, the fact that permutations do not have repeated items is denoted as *mutual exclusivity constraint*, and has been a difficult task to deal with when developing EDAs. Until recently, for such problems, EDAs had not been extensively developed, and classical EDA approaches have been adapted "naively" in

This work has been partially supported by the Research Groups 2013-2018 (IT-609-13) and Elkartek programs (Basque Government), and TIN2016-78365R and Severo Ochoa Program SEV-2013-0323 (Spanish Ministry of Economy, Industry and Competitiveness).

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

order to hold the mutual exclusivity constraint associated to the solutions represented as permutation vectors [8].

With illustrative purposes, let us consider a combinatorial optimization problem whose solutions are codified as permutations, and a Univariate Marginals Distribution Algorithm (UMDA) [24] that is going to be used to optimize it. In the classical design of UMDA, this algorithm optimizes a vector over a given objective function and calculates a joint probability distribution that factorizes over the positions in the vector as a product of the marginal probabilities of the values in each position. In other words, it assumes independence among positions. In UMDA, the characteristics of a set of selected solutions to be passed onto the next generation depend on the first order marginal probabilities of each position of the solution.

Sampling a distribution to make a new generation is therefore done by sampling each position of the vector independently. This results in a vector that may not satisfy the mutual exclusivity constraint because of replicated items. A straightforward solution consists of modifying the sampling method to avoid non-permutation solutions. However, many recent works in the literature have considered using probability models that define probability distributions on $\mathbb{S}_n$, such as Mallows and Generalized Mallows models [9], [16] (based on distance-metrics) or Bradley-Terry and Plackett-Luce models [2], [11] (based on order statistics).

In this paper, we study alternative representations for codifying solutions in permutation problems that permit classical EDAs to be implemented without adaptations that denaturalize their application. Specifically, we consider recodifications that map every permutation in the search space of solutions to a vector of integers. Conveniently, the new search space might allow repeated values inside the same vector, and thus remove the constraints observed previously with the permutation representation of solutions. Moreover, summary statistics learnt from the set of selected solutions is different for each codification, and so we expect the performance of the EDA to vary between codifications.

This work is motivated by the lack of literature to understand how and why different codifications affect the behaviour and performance of an EDA depending on the problem to solve. In fact, most of the literature in EDAs focuses on the research of probability models, in many cases ignoring the implications of choosing an adequate representation. Conducted experiments in this work show the vast superiority of the algorithm when operating with suitable codifications[1]. Specifically, we considered a classical EDA, the Univariate Marginal Distribution Algorithm (UMDA), and two permutation problems, the Quadratic Assignment Problem (QAP) and the Permutation Flowshop Scheduling Problem (PFSP).

The remainder of the paper is organized as follows, in the next section some background on the UMDA is provided, and its adaptation to deal with permutation-coded solutions is described. Afterwards, Section III is devoted to describing alternative representations to codify the solutions in $\mathbb{S}_n$. Next, the conducted experiments on the PFSP and QAP, and the posterior analysis of the results are presented in Section IV. In Section V, a discussion on the different representations and other research lines to extend this work are introduced. Finally, the conclusions of the work are exposed in Section VI.

[1]The terms codification and representation will be used interchangeably throughout the paper.

II. Univariate Marginal Distribution Algorithm

The Univariate Marginal Distribution Algorithm (UMDA) was proposed by Pelikan and Mühlebein in 1999 [24]. Given a population of integer valued vectors of length n, at every iteration, a high-quality set of solutions $\mathbf{X} = \{x^1, x^2, \ldots, x^n\}$ is chosen, and univariate marginal frequencies of solutions in $\mathbf{X}$ are calculated. This is done by counting the number of times a specific item appears in a certain position in the solutions of $\mathbf{X}$. Thus, for every solution $\mathbf{x}$ in the search space Ω, its probability is calculated as,

$$P(x) = \prod_{i=0}^{n-1} p(x(i) = j) \tag{1}$$

where j denotes the item in the i-th position of x, and $p(x(i) = j)$ is the probability of j appearing in the i-th position of x. It is denoted *first order marginal* probability.

Once the parameters of the UMDA are learnt, the next step consists of sampling solutions from this distribution by adapting the classical sampling method for integer vectors. The classical UMDA samples a new vector x following the next iterative procedure: for each position $1 \leq i \leq n$, a value is randomly chosen in the range of $x(i)$ with probabilities $p(x(i))$. This trivial approach does not guarantee that the sampled vector is a permutation. Thus, a direct solution to this problem is the following: once the item $x(i)$ is sampled in the i-th position, the probability of sampling item $x(i)$ again in any other position is negated and the probabilities of the rest of the items are normalized (see Fig. 1).

This trivial adaptation does not necessarily generate a sample with the first marginals given by the probability matrix. Indeed, the marginals of the first position of the permutation will be accurate but, since the matrix is updated at each step, the last positions can accumulate large biases. To overcome this situation, and limit the bias produced by this sampling mechanism, a possibility consists of choosing the positions to sample uniformly at random (instead of starting from 1 and visiting all the positions until n) for each solution to be sampled. This way, the first order marginals of the generated sample converge to $P(\mathbf{x})$.

III. Alternative Codifications for permutations

When representing solutions in permutation-based problems, the usual and natural option is to use the permutation representation. A permutation σ is formally defined as a bijection from the set of natural numbers $[n] = \{1, \ldots, n\}$ onto $[n]$ and is usually represented as an ordered list of the set $[n]$. We denote by $\sigma(i) = j$ that item j is in the i-th position of permutation σ. Alternatively, we say that σ^{-1} is

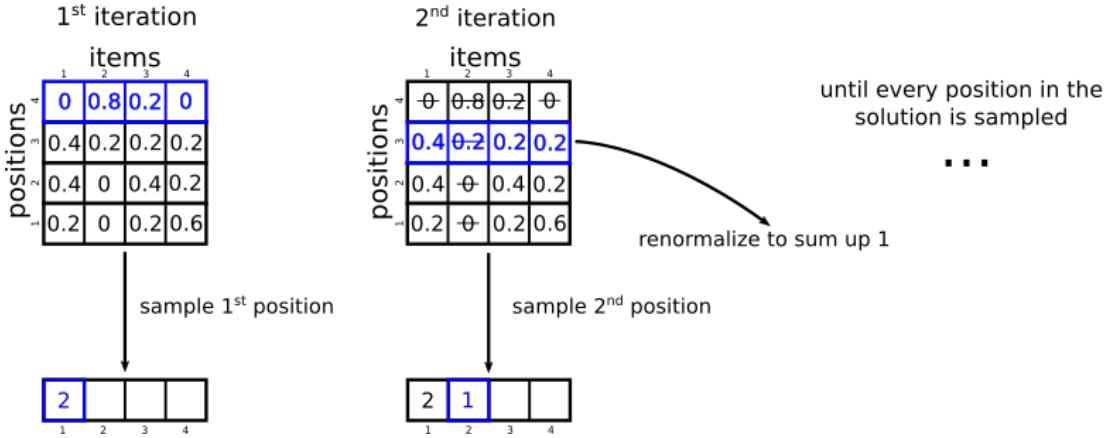


Fig. 1: Illustration of the sampling step in the UMDA

the inverse of σ, if and only if, $\sigma(i) = j$ and $\sigma^{-1}(j) = i$ for all $i, j = 1, \ldots, n$. For instance, the inverse permutation of $\sigma = (2, 4, 3, 1)$ is $\sigma^{-1} = (4, 1, 3, 2)$.

Even though permutations are intuitive and apparently the obvious representation for permutation-based problems, when considering classical EDAs some issues arise.

Given a permutation σ of length n, the only possible items are $\{1, .., n\}$, and these cannot be repeated. In the previous section we have seen that the classical sampling method for vectors has to be adapted to sample permutations. Although the first order marginal of the sample is maintained, some problems emerge. For example, the need of updating the probabilities at each step is time consuming. Moreover, the information of the first order marginals might not be relevant for the problem at hand. Therefore, we propose to use a different representation for permutations for which the mutual exclusivity constraint does not arise. Furthermore, this representation allows us to extract different summary statistics from the sample.

Alternatively to the classical representation, other types of numeric vectors that codify permutations have also been reported in the literature [3], [12]. In this paper, we focus on the *inversion vector* representation of permutations, equivalent to Lehmer vectors [20].

Given a permutation σ, its inversion vector is $I = (I(1), \ldots, I(n-1))$ where $I(i) = \sum_{j>i} \mathbb{I}[\sigma(i) > \sigma(j)]$, being $\mathbb{I}$ the indicator function. In other words, each item $I(i)$ in the inversion vector I counts the number of items lower than $\sigma(i)$ at its right. For a permutation of length n, its inversion representation is a vector of length n, however, by definition, the last item of the inversion vector is always zero, thus we will omit this item and consider the inversion vector of length $n-1$. For instance, given a permutation $\sigma = (3, 1, 4, 2)$, its inversion vector representation is defined as $I = (2, 0, 1)$. Following its definition, having $I(1) = 2$ and $\sigma(1) = 3$ means that there are two values lower than 3 in σ on its right. Likewise, $I(2) = 0$ indicates that there are no values lower than $\sigma(2)$ on its right in σ.

The algorithm to decode an inversion vector I to its permutation representation σ is described in Algorithm 1. This algorithm and the one that transforms a permutation into its inversion vector are run in $O(n^2)$, however there exist variants that do the same procedure in $O(n \log n)$ [22].

Algorithm 1 Inversion vector to permutation representation.

```
i ← 1
e ← (1, 2, ..., n)
while i < n do
    σ(i) ← e(I_i + 1)
    e ← delete item at I_i + 1 from e
    i ← i + 1
end while
σ(n) ← e(1)
return σ
```

A. *The central permutation* σ_0

In the previous paragraphs, the inversion representation of any permutation is naturally calculated with respect to the identity permutation. However, according to the literature, the appropriate way to compute the inversion representation of a set of permutations is to do it with respect to a consensus reference of the set to codify [14]. Following previous reference, in this paper, the Borda algorithm [5] is used to compute the consensus (central) permutation of a given set of permutations. Specifically, the Borda algorithm first calculates the Borda scores for each item, and then returns the ranking that orders the items by increasing Borda score. Finally, the resulting ranking is taken as the central permutation, denoted as $\hat{\sigma}_0$.

In order to compute the inversion representation of every permutation σ, firstly, these are composed on the right with the central permutation $\hat{\sigma}_0^{-1}$ (the inverse permutation of $\hat{\sigma}_0$), then the inversion vector of the resulting permutation is calculated. In summary, to represent a set of permutations as inversion vectors, the inversion vector of the i-th permutation σ_i is computed from $\sigma_i \circ \hat{\sigma}_0^{-1}$.

On the other hand, after transforming an inversion vector to its permutation representation, the obtained permutation has to be composed with the central permutation. This step is needed to recover the original permutation, as the inversion vectors (when learning the model) were calculated from $\sigma_i \circ \hat{\sigma}_0^{-1}$, and not directly from σ_i. As composing the inverse of the central permutation $\hat{\sigma}_0^{-1}$ with the central permutation $\hat{\sigma}_0$ is equal to the identity permutation, composing on the right the sampled permutation with σ_0 will return the original permutation, i.e., $\sigma_i \circ \hat{\sigma}_0^{-1} \circ \sigma_0 = \sigma_i$. Finally, it is worth mentioning that a probability distribution learned from inversion vectors is equivalent to the multistage model described in [14].

B. I and I^* vectors

When it comes to approaching permutation problems, it is not clear whether using σ or σ^{-1} is more suitable with regard to the estimation of the parameters of the probability model. In our case, when computing the inversion vector of a solution of the problem (see Algorithm 1) the same problem arises. The inversion vector I can be computed from σ (the representation of the solutions that is given as input to the objective function, i.e., see Equations 2 and 3) or from σ^{-1}. As we do not have intuition regarding their suitability, we will test both options. In order to clarify the notation and avoid confusion in later sections, from now on, we will refer to inversion vectors that use σ as input as I, and I^* to the inversion vector that is computed from σ^{-1}.

By using either I or I^* inversion, each of the permutations in $\mathbb{S}_n$ is mapped to one vector of integers and vice-versa (the transformation is bijective). Conveniently, the new search space compound by vectors of integers of size $n-1$ removes any constraint related to the codification as repeated items inside the vectors are allowed. Thus, the probability distributions, such as in UMDA, can easily be learned, and solutions sampled without caring about any information loss or bias in the procedure.

The scheme that we propose works as follows. After evaluating the population and selecting a set $\mathbf{X}$ of permutation solutions, every solution in $\mathbf{X}$ is encoded, creating a representation of $\mathbf{X}$ in another vector space, I (or I^*). Next, we learn $p(I^\sigma(i) = j)$[2], the first order marginals of I (or I^*). Then, new solutions are sampled inside the new vector space. Note that the position dependency does not occur for the inversion vectors, each position is sampled independently. Finally, samples are decoded and converted into their permutation representation for later evaluation. Fig. 2 depicts the proposed scheme for the case of UMDA.

IV. Experimental Study

In order to evaluate the influence of the representation considered to codify the solutions to the behavior of the EDA, a set of experiments were conducted. Specifically, we tested various representations, the permutation-vector σ, the inversion vector I and the inversion vector of the inverse permutation σ^{-1}, I^*. As a test bed, we considered two permutation problems that have been reported of very different nature [10]: the Permutation Flowshop Scheduling Problem (PFSP) and the Quadratic Assignment Problem (QAP). In what follows, before presenting the conducted experiments, the definition of the two problems are given (which will be needed later).

[2] I^σ denotes the inversion vector corresponding to σ permutation. For the sake of clarity, this representation was simplified to I in the previous chapters.

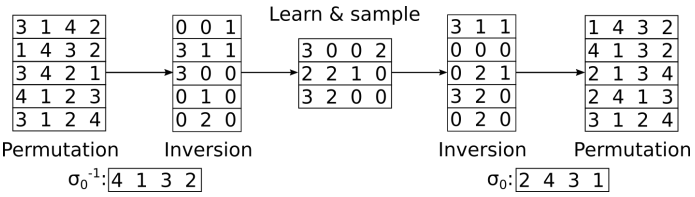


Fig. 2: Example of a UMDA operating on inversion vectors as the representation of solutions

A. Two cases of study

The Quadratic Assignment Problem (QAP) [18] is an assignment or location analysis type problem. It consists of allocating n facilities in n different locations in the map, while minimizing the cost function Eq. 2. For each pair of positions i and j there is a distance $d_{i,j}$ parameter. In addition, for each pair of facilities there is a flow parameter $h_{k,l}$ associated. The sets of distances and facilities are described by the matrices of real values $D = [d_{i,j}]_{n\times n}$ and $H = [h_{k,l}]_{n\times n}$, respectively. The objective value of any solution (allocation) σ is calculated as:

$$f(\sigma) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j} h_{\sigma(i),\sigma(j)} \tag{2}$$

For the purposes of this section, we selected a set of 20 instances obtained from QAPLIB [6] with sizes that range from 20 to 175.

In the Permutation Flowshop Scheduling Problem (PFSP) [1], [4] a set of n jobs needs to be processed on m machines. Every job must pass through every machine with no interruption, and jobs have to be scheduled in such an order that the cost function in Eq. 3 is minimized. Any job is available at time zero, and a job is processed in the j-th machine if the operation in the previous $j-1$ machine is finished and the j-th machine is free. Given a permutation σ of length n, the i-th job is scheduled in the $\sigma(i)$ position. Finally, the processing time of a job in a particular machine is defined by the matrix $P = [p_{i,j}]_{n\times m}$.

Different objective functions has been proposed when optimizing PFSP. However, in this work we will only approach the minimization of the makespan. The makespan is the time that it takes to finish all jobs. Given a solution σ, the makespan is defined as follows:

$$f(\sigma) = c_{\sigma(n),m} \tag{3}$$

where $c_{\sigma(n),m}$ is the time it takes to finish the last job, n, in the last machine. This time is calculated recursively as

$$c_{\sigma(i),j} = \begin{cases} p_{\sigma(i),j} & i = j = 1 \\ p_{\sigma(i),j} + c_{\sigma(i-1),j} & i > 1, j = 1 \\ p_{\sigma(i),j} + c_{\sigma(i),j-1} & i = 1, j > 1 \\ p_{\sigma(i),j} + \max\{c_{\sigma(i-1),j}, c_{\sigma(i),j-1}\} & i > 1, j > 1 \end{cases}$$

For this experimentation, we considered a subset of instances of the PFSP from the well-known benchmark of Taillard [27]. Particularly, 22 instances that range between 20 and 200 jobs were selected.

B. Experimental setup

Regarding the UMDA, n being the size of the problem, the population size was set to $10n$, $5n$ best solutions were selected by truncation and $10n$ new solutions were sampled from the probability model at each iteration. The algorithm stops after $100n$ iterations. This parameters were set following the usual trends and without fine-tuning the behavior of the algorithm.

In order to draw solid conclusions, each algorithm (UMDA under the different representations) was run on each instance 20 times, and median values were collected.

C. Results for σ, I and I^*

In the first experiment, the UMDA was run under σ, I and I^* representations. In the case of σ, when sampling solutions, using the classical procedure it is possible to draw unfeasible solutions (non-permutation) because of the repetition of the items. Therefore, the sampling was adapted to hold the permutation nature of solutions (once an item has been sampled, set the probability of sampling that item in any other position to zero, as described in Fig. 1).

Results can be observed in Tables I and II. When optimizing PFSP instances, the UMDA over I^* obtains better results in almost every tested PFSP instance, being the best option in 18 instances out of 22. In contrast, when solving QAP instances, the performance of the UMDA over the permutation representation, σ, is better, thus, being the permutation-vector representation systematically the best option for this problem.

D. Understanding the results

In order to better understand the behavior of the UMDA under the three representations, in this section, a more precise analysis of the results is carried out in terms of convergence of the UMDA. Particularly, two instances of the PFSP and QAP were chosen respectively (small and large), and for each instance-representation pair two logs were plotted: 1) the best fitness value obtained in the population across the iterations, and 2) the standard deviation of the fitness of the solutions sampled at each iteration. The first log provides information about the degree and speed of the optimization, and the second gives information about the level of concentration of the probability model related to the quality of the solutions. In all the cases, the results of 20 repetitions were aggregated.

Regarding the results of PFSP (see Fig. 3), starting with the small instance, the UMDA over I^* obtains the best results.

TABLE I: Median values obtained by the UMDA using the three representations across the 20 repetitions on the instances of PFSP. Boldfaced results denoted the best result among the three representations.

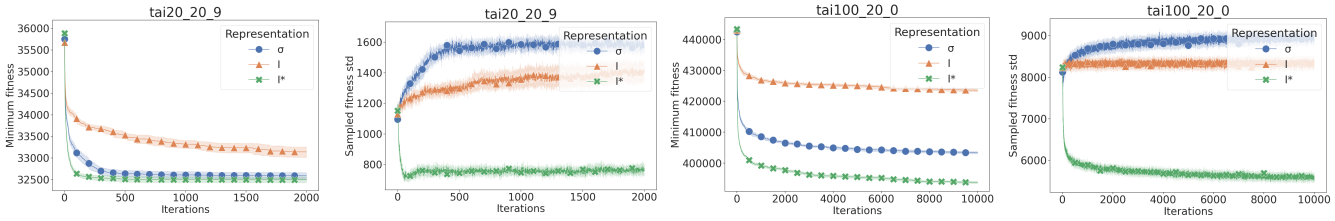
Instance	Size	σ	I	I^*
tai20_5_8	20	**1.44e4**	1.47e4	1.44e4
tai20_5_9	20	1.31e4	1.35e4	**1.30e4**
tai20_10_8	20	**2.04e4**	2.13e4	2.05e4
tai20_10_9	20	**2.14e4**	2.23e4	2.14e4
tai20_20_8	20	**3.39e4**	3.42e4	3.39e4
tai20_20_9	20	3.26e4	3.32e4	**3.26e4**
tai50_5_8	50	6.68e4	7.08e4	**6.40e4**
tai50_5_9	50	7.29e4	7.54e4	**6.99e4**
tai50_10_8	50	9.22e4	9.68e4	**8.81e4**
tai50_10_9	50	9.39e4	1.00e5	**9.04e4**
tai50_20_8	50	1.29e5	1.36e5	**1.24e5**
tai50_20_9	50	1.31e5	1.35e5	**1.27e5**
tai100_5_8	100	2.70e5	2.88e5	**2.62e5**
tai100_5_9	100	2.65e5	2.85e5	**2.57e5**
tai100_10_8	100	3.31e5	3.51e5	**3.21e5**
tai100_10_9	100	3.23e5	3.44e5	**3.12e5**
tai100_20_8	100	4.12e5	4.31e5	**4.03e5**
tai100_20_9	100	4.19e5	4.38e5	**4.08e5**
tai200_10_8	200	1.15e6	1.21e6	**1.12e6**
tai200_10_9	200	1.15e6	1.23e6	**1.12e6**
tai200_20_8	200	1.38e6	1.44e6	**1.35e6**
tai200_20_9	200	1.39e6	1.46e6	**1.36e6**

TABLE II: Median values obtained by the UMDA using the three representations across the 20 repetitions on the instances of QAP. Boldfaced results denoted the best result among the three representations.

Instance	Size	σ	I	I^*
chr20a	20	**2.58e3**	4.69e3	3.57e3
chr20b	20	**2.72e3**	4.63e3	3.63e3
chr20c	20	**1.86e4**	3.84e4	1.89e4
tai20a	20	**7.22e5**	7.84e5	7.82e5
tai20b	20	**1.23e8**	1.27e8	1.24e8
tai45e01	45	**3.25e4**	1.60e5	1.37e5
tai45e02	45	**3.26e4**	1.40e5	1.45e5
tai45e03	45	**3.27e4**	2.35e5	2.21e5
tai45e04	45	**1.45e4**	1.33e5	1.60e5
tai45e05	45	**1.90e4**	1.44e5	1.72e5
sko100a	100	**1.61e5**	1.71e5	1.63e5
sko100b	100	**1.63e5**	1.73e5	1.66e5
sko100c	100	**1.57e5**	1.67e5	1.59e5
tai100a	100	**2.34e7**	2.35e7	2.35e7
tai100b	100	**1.30e9**	1.54e9	1.32e9
tai175e01	175	**6.42e5**	6.04e6	6.45e6
tai175e02	175	**5.51e5**	6.12e6	5.99e6
tai175e03	175	**5.62e5**	5.77e6	6.01e6
tai175e04	175	**7.30e5**	6.44e6	6.69e6
tai175e05	175	**5.64e5**	6.10e6	6.29e6

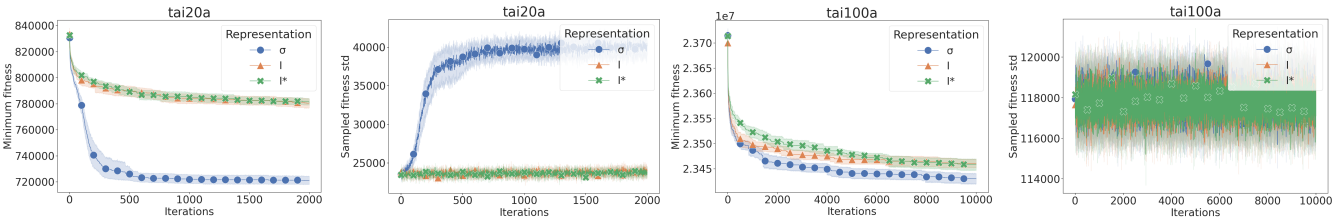
Observing Fig. 3a, the objective value of the UMDA over I^* drastically drops in the first iterations. This corresponds to the rapid fall of the standard deviation of the sampled solutions from the UMDA over the same representation, Fig. 3b. This suggests that the model over I^* gives high probability to high-quality solutions, enabling the model to rapidly optimize its solutions. While the models over σ and I tend to create more sparse solutions, as can be deduced from Fig. 3b.

In the case of the larger instance size of PFSP, the UMDA over I^* starts following the same behaviour as in the previous experiment. The standard deviation of the sampled



(a) Best fitness and $n = 20$. (b) Std. dev. and $n = 20$. (c) Best fitness and $n = 100$. (d) Std. dev. and $n = 100$.

Fig. 3: Best population fitness and standard deviation of the fitness of the sampled solutions observed throughout the optimization of two instances of the PFSP: *tai20_20_9* and *tai100_20_0*. In each plot, data obtained in 20 repetitions has been aggregated.



(a) Best fitness and $n = 20$. (b) Std. dev. and $n = 20$. (c) Best fitness and $n = 100$. (d) Std. dev. and $n = 100$.

Fig. 4: Best population fitness and standard deviation of the fitness of the sampled solutions observed throughout the optimization of two instances of the QAP: *tai20a* and *tai100a*. In each plot, data obtained in 20 repetitions has been aggregated.

solutions rapidly drops unlike with the other codifications (see Fig. 3d), and high-quality solutions are generated (see Fig. 3c). After completing more iterations, unlike the models over the other representations, the UMDA over I^* does not converge and continues to optimize its objective value. Finally, following the results from Table I, the model over I^* obtains the best results. On the contrary, models over the other representations seem to follow the opposite path. Besides generating great quality solutions in the beginning, the standard deviation of the solutions increases, thus being unable to continue sampling high-quality solutions.

With respect to the QAP (see Fig. 4), the results are clear for the small sized instance, the model over σ is the codification that best suits this type of problem, as shown in Table II.

Regarding Fig. 4a, the algorithm with the model over σ representation optimizes its solutions efficiently, and rapidly converges. On the other hand, unlike to the previous case, EDAs over I and I^* do not perform so well. In fact, their convergence is slower and worse results are returned. As can be seen in Fig. 4b, the standard deviation of the EDAs over inversion codifications is very low from the beginning, suggesting a premature convergence of these models. In the greater instance size, the performance of the model over σ continues being superior compared to the models over other codifications, Fig. 4c. However, as opposed to the small instance size, the performance of the considered models over the different codifications is quite similar. Moreover, the standard deviation of the sampled solutions is practically the same (see Fig. 4d) and the final results are not as different as in the case of the small instance, Fig. 4a.

In summary, in the case of PFSP instances, as can be seen in Fig. 3, the minimum fitness of the population rapidly drops in models over I^*, and the standard deviation of the results is always lower than the other models over σ and I. This indicates that models over I^* are able to generate very high-quality solutions for PFSP problems. Sometimes, this behavior, (mainly in small instance sizes) led the models over I^* to prematurely converge and lose performance, as seen in few instances in Table I. When considering QAP instances, regardless of the size of the problem, models over σ show clear superiority when compared to models over I and I^*.

V. Discussion

Previous experiments demonstrated the influence of the representation used to codify the solutions when optimizing the problem. On one hand, we observed that in the case of the PFSP, the inversion vector allows the UMDA to obtain better results. On the other hand, contrarily to the PFSP, in the QAP the usual permutation codification turned out to be clearly preferred. Based on these results, it is obvious that when converting a permutation to its corresponding inversion vector, the explicit information that the model receives changes, and this affects the performance of the algorithm. Some times this effect is beneficial (such as in PFSP), and other times it is negative (like for QAP).

An interesting point is that the distribution over permutations learnt and sampled by UMDA under the different representations differs drastically. Under the permutation vector representation, the information that is propagated across generations is the first order marginal, i.e., the probability of the items to appear at each position of the solution σ, $p(\sigma(i) = j)$. Conversely, when using the inversion vector representation I, for each position i, the number of items

smaller than $\sigma(i)$ that lie to the right of position i are counted, and $p(\sum_{j>i} \mathbb{I}[\sigma(j) < \sigma(i)])$ is propagated.

Nevertheless, beyond the propagated information for each representation, the reason for the variations observed in the results is not clear. Moreover, the influence of each type of representation for each problem is a pending task. Ultimately, the relevant questions is: *which is the best representation to codify my solutions?*

This paper aims at throwing light in a wide research line that analyze the characteristics of the solutions that have to be taken into account so that a standard algorithm (local search, probabilistic evolutionary algorithms, ...) successes at optimizing a particular problem. For example, in an EA the best solutions are selected and the next generation is supposed to keep the characteristics that made the solutions well adapted. In this way, the question is *which are the characteristics of the solutions that the EA should preserve in order to improve the current solution?*

In this paper, the inversion vector was used, however, there exist other types of representation that could be also used under the same setting. One of them is the codification of permutations used in the Repeated Insertion Model (RIM) [12].

The representations introduced previously are bijective, which means that for each permutation there is a unique transformation, and this holds also in the opposite direction. Nevertheless, there are transformations that are not necessarily bijective and could be used in the context presented in this paper. Other possible representations are the decomposition vectors of the Cayley [15] and Hamming [16] distances in the permutations space. The first one is related to the cycle structure of permutations and the second one with the fixed points of the permutation.

In the case of the Cayley and Hamming transformations, unlike with Inversion vectors or RIM, there are multiple permutations that can generate the same decomposition vectors (the transformations are not bijective). Then, using such transformations may introduce large redundancies in the codification that usually provoke a poor performance of the algorithm.

VI. Conclusions & Future Work

In this paper, we use the UMDA over two representation of permutations in two different optimization problems. The UMDA is an optimization algorithm originally developed for integer vectors. It is based on the assumption that the joint distribution over all the positions in the vector can be approximated as the product of the distributions of each of the positions, i.e., UMDA assumes independence of the positions.

Permutations of n items are usually represented as vectors but have the peculiarity that the items $[n]$ are not repeated. We denote this characteristic along this paper as the mutual exclusivity constraint. It is the reason because the UMDA can no be directly applied to permutation problems. Indeed, the problems arise when sampling a permutation, where the sampling process has to be adapted so that the sampled vector is a permutation.

In this paper, we propose to use an UMDA over an alternative representation of permutations, the inversion vectors. Inversion vectors are $n-1$ length vectors in which position i ranges in $[0, n-j]$. It it easy to see that there is a bijection between permutations of n items and inversion vectors of $n-1$ and the conversion from one to the other can be done efficiently. Interestingly, inversion vectors do not suffer from mutual exclusivity constraints, which means that when permutations are represented as inversion vectors it is trivial to define an UMDA over them: (1) instead of learning a probability distribution over the set of the best permutations, we learn a probability distribution over the inversion vectors of the permutations and (2) to sample a new population we sample inversion vectors and then transform them to permutations.

We have run several experiments to compare the original UMDA setting adapted to permutations, which is denoted as σ along the paper, and two UMDA over the inversion vectors, called I and I^*. The optimization problems considered have been QAP and PFSP. It is interesting to notice that the first UMDA is very adequate to solve the QAP while the other two are suited for the PFSP. Indeed, this paper is devoted to throw light in this direction, and show why probabilistic evolutionary algorithms that use certain summary statistics are well fitted for certain problems while those that use other summary statistics are well fitted for others.

This work was the first step in the evaluation of different representations, and there is still room for future investigations. On one hand, in addition to the permutation and inversion vectors, the literature has published other representations to codify permutation/ranking solutions that could be similarly used in this context. The *Repeated Insertion Model (RIM)*, *Fixed points*, the *pairwise representation* or the *factoradic representation* [26] are some examples that could be investigated in the future. Going beyond already published representations, a relevant research line could be designing ad-hoc representations for the permutation problem at hand. On the other hand, the work presented in this paper used the UMDA as a benchmark algorithm. However, it would be interesting to see whether other classes of EDAs may be as equally influenced as the UMDA.

Acknowledgments

This work has been partially supported by the Research Groups 2013-2018 (IT-609-13) and Elkartek programs (Basque Government), and TIN2016-78365R and Severo Ochoa Program SEV-2013-0323 (Spanish Ministry of Economy, Industry and Competitiveness) and BERC 2014-2017 program and by the Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa excelence accreditation SEV-2017-0718, and through the project TIN2017-82626-R funded by (AEI/FEDER, UE).

References

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[2] J. Alza, J. Ceberio, and B. Calvo. Balancing the diversification-intensification trade-off using mixtures of probability models. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, July 2018.

[3] Z. Arnavut. Move-to-front and inversion coding. In *Proceedings DCC 2000. Data Compression Conference*, pages 193–202, March 2000.

[4] K. R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[5] J. C. Borda. Memoire sur les elections au scrutin. *Histoire de l'Academie Royale des Science*, 1784.

[6] R.E. Burkard, S. Karisch, and F. Rendl. Qaplib-a quadratic assignment problem library. *European Journal of Operational Research*, 55(1):115 – 119, 1991.

[7] J. Ceberio. *Solving permutation problems with estimation of distribution algorithms and extensions thereof*. PhD thesis, Universidad del País Vasco-Euskal Herriko Unibertsitatea, 2014.

[8] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A review on Estimation of Distribution Algorithms in Permutation-based Combinatorial Optimization Problems. *Progress in Artificial Intelligence*, 1(1):103–117, January 2012.

[9] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A distance-based ranking model estimation of distribution algorithm for the flow-shop scheduling problem. *IEEE Transactions Evolutionary Computation*, 18(2):286–300, 2014.

[10] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A review of distances for the Mallows and Generalized Mallows estimation of distribution algorithms. *Computational Optimization and Applications*, 62(2):545–564, November 2015.

[11] J. Ceberio, A. Mendiburu, and J. A. Lozano. The plackett-luce ranking model on permutation-based optimization problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 494–501, June 2013.

[12] J. P. Doignon, A. Pekeč, and M. Regenwetter. The repeated insertion model for rankings: Missing link between two subset choice models. *Psychometrika*, 69(1):33–54, 2004.

[13] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, Nov 2006.

[14] M. A. Fligner and J. S. Verducci. Multistage ranking models. *Journal of the American Statistical association*, 83(403):892–901, 1988.

[25] K. V. Price. *Differential Evolution*, pages 187–214. Springer Berlin Heidelberg, 2013.

[15] E. Irurozki, B. Calvo, and J. A. Lozano. Sampling and learning mallows and generalized mallows models under the cayley distance. *Methodology and Computing in Applied Probability*, 20(1):1–35, Mar 2018.

[16] Ekhine Irurozki, Borja Calvo, Jose A Lozano, et al. Permallows: An r package for mallows and generalized mallows models. *Journal of Statistical Software*, 71(12):1–30, 2016.

[17] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[18] T. C. Koopmans and M. Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25(1):53, January 1957.

[19] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[20] D. H. Lehmer. Teaching combinatorial tricks to a computer. In *Sympos. Appl. Math. Combinatorial Analysis*, pages 179–193. American Mathematical Society, Providence, 1960.

[21] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., 2006.

[22] H. W. Martin. Transformations between tree permutations and inversion tables. In *Proceedings of the 1990 ACM Annual Conference on Cooperation*, CSC '90, pages 140–146, New York, NY, USA, 1990. ACM.

[23] H. Mühlenbein and G. Paaß. From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, 1996.

[24] M. Pelikan and H. Mühlenbein. Marginal distributions in evolutionary algorithms. In *In Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pages 90–95, 1999.

[26] O. Regnier-Coudert and J. McCall. Factoradic representation for permutation optimisation. In *Parallel Problem Solving from Nature – PPSN XIII*, pages 332–341. Springer International Publishing, 2014.

[27] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993. Project Management anf Scheduling.