Fundamenta Informaticae XXI (2001) 1001–1022
DOI 10.3233/FI-2016-0000
IOS Press

Tackling permutation-based optimization problems with an Algebraic Particle Swarm Optimization algorithm

Marco Baioletti
Department of Mathematics and Computer Science
University of Perugia, Italy
marco.baioletti@unipg.it

Alfredo Milani
Department of Mathematics and Computer Science
University of Perugia, Italy
alfredo.milani@unipg.it

Valentino Santucci
Department of Social and Human Sciences
University for Foreigners of Perugia, Italy
valentino.santucci@unistrapg.it

Abstract. Particle Swarm Optimization (PSO), though being originally introduced for continuous search spaces, has been increasingly applied to combinatorial optimization problems. In particular, we focus on the PSO applications to permutation-based problems. As far as we know, the most popular and general PSO schemes for permutation solutions are those based on random key techniques. In this paper, after highlighting the main criticalities of the random key approach, we introduce a totally discrete PSO variant for permutation-based optimization problems. The proposed algorithm, namely Algebraic PSO (APSO), simulates and bring the original PSO design in the permutations search space. APSO directly represents both particle positions and velocities as permutations. The APSO search scheme is based on a general algebraic framework for combinatorial optimization based on strong mathematical foundations. The particularities of the PSO design scheme arouse new challenges for the algebraic framework: the non-commutativity of the velocity terms, and the rationale behind the PSO inertial move. Design solutions have been

Address for correspondence: valentino.santucci@unistrapg.it

proposed for both the issues. Furthermore, an alternative geometric interpretation of classical PSO dynamics allow to introduce a major APSO variant based on a novel concept of convex combination between permutation objects. In total, four APSO schemes have been introduced. Experiments have been held to compare the performances of the APSO schemes with respect to the random key based PSO schemes in literature. Widely adopted benchmark instances of four popular permutation problems have been considered. The experimental results clearly show, with high statistical evidence, that APSO outperforms its competitors.

Keywords: Algebraic Particle Swarm Optimization, Permutation Problems, Permutation Search Space

1. Introduction

Particle Swarm Optimization (PSO) is a popular evolutionary algorithm introduced by Kennedy and Eberhart in 1995 [1]. Unlike classical evolutionary algorithms, its search scheme does not rely on genetic operators. Indeed, in PSO, a population of commonly called particles is iteratively updated by means of simple move equations whose design is inspired by swarm intelligence principles [2].

Though it has been originally proposed for continuous search spaces, PSO variants for combinatorial optimization problems have increasingly appeared in literature [3, 4, 5]. While the first discrete PSO has been introduced for binary problems [6], a large number of PSO applications to permutation-based optimization problems has been proposed. See for instance [7, 8, 9, 10].

In this paper we focus on PSO algorithms for the permutations search space. To the best of our knowledge, the vast majority of PSO applications to permutation problems are based on the random key (RK) technique, and slight variants, which has been originally proposed in [11] in the context of genetic algorithms. RK consists in a decoder function which converts a vector of $\mathbb{R}^n$ into a permutation of n integers by sorting the vector components. Using RK, permutation problems can be approached by the classical PSO scheme. The only required modification is to introduce the RK decoding step whenever a particle position has to be evaluated. However, despite of its simplicity, the PSO+RK approach has several issues:

- often, the reported good results are only obtained by hybridizing the PSO+RK scheme with a variety of additional techniques (local searches, heuristic functions, restart mechanisms, etc.) and, as far as we know, no study is provided to understand if the standalone PSO+RK is effective or not;
- due to obvious cardinality reasons, a single permutation can be encoded by a potentially infinite number of numeric vectors, thus introducing large plateaus in the fitness landscape navigated by the underlying continuous PSO;
- the intuition of how the PSO algorithm searches and moves in the continuous space, for which it has been originally designed, is completely lost when the algorithm is integrated with the RK decoding procedure and its moves are observed in the combinatorial search space of permutations.

In a previous series of papers [12, 13, 14, 15, 16, 17, 18], we have proposed very effective algebraic differential evolution schemes for permutation problems. The classical differential evolution [19], like PSO, is an algorithm for continuous optimization. The proposed discrete variants are based on an original framework that exploits the algebraic structure of the permutations search space. Here, we use the algebraic framework to introduce a totally discrete variant of PSO for the permutations search space that, in contrast to the PSO+RK approach, directly evolves a population of permutations by redefining the PSO move equations in such a way that the continuous PSO movement design is simulated in the permutations search space.

In contrast to the previously proposed algebraic differential evolution, the Algebraic PSO (APSO), which has been preliminary proposed in [20], makes use of only algebraic operators, thus representing the first "completely algebraic" evolutionary algorithm proposal. APSO directly represents both the particles' positions and velocities as permutations. Besides the direct application of the algebraic framework to PSO, further studies have been conducted.

The non-commutativity of the framework's addition operator highlighted a new design choice on the ordering of the PSO velocity terms.

Then, we studied, both analytically and experimentally, the APSO inertial move in the permutations space. Such analysis highlights some anomalous behaviors in the particles' inertial dynamic. Hence, a second variant of APSO is proposed in order to avoid such cases and preserving the particle inertial trajectory similarly to what happens in the continuous PSO.

Moreover, a major variant of APSO, namely the barycentric APSO, is introduced by modeling the discrete velocity update rule by means of an alternative geometric interpretation of classical PSO movement dynamics. In order to introduce the barycentric APSO, a meaningful process to compute convex combinations between permutations is also introduced.

In total, four variants of APSO are presented: the basic APSO, the inertia-preserving APSO-i, the barycentric APSO-b, and the inertia-preserving barycentric APSO-bi.

Experiments have been held with the aim of comparing the effectiveness of the proposed APSOs with respect to the PSO+RK approaches in literature. Problem instances have been selected from commonly adopted benchmark suites of the four most popular permutation problems: the permutation flowshop scheduling problem (PFSP), the linear ordering problem (LOP), the traveling salesman problem (TSP), and the quadratic assignment problem (QAP). The experimental results show that the proposed APSOs clearly outperform the PSO+RK schemes with high statistical evidence.

The rest of the paper is organized as follows. Section 2 describes the classical PSO algorithm together with the random key techniques. The algebraic framework for combinatorial optimization is briefly described in Section 3. The basic and inertia-preserving APSO schemes are introduced in Section 4, while the barycentric APSOs are motivated and introduced in Section 5. The experimental analysis is provided in Section 6, while conclusions are drawn in Section 7, where future research directions are also depicted.

2. Particle Swarm Optimization and Random Key

The description of a standard continuous PSO is provided in Section 2.1. This scheme has been used as the PSO reference implementation throughout the rest of the paper. Moreover, Section 2.2 recalls

the random key approaches available in the literature by also connecting them through a novel and simple algebraic consideration.

2.1. Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm [1] iteratively evolves a population of N particles in order to optimize the given numerical objective function $f : \mathbb{R}^n \to \mathbb{R}$.

The i-th particle of the population is composed by: the current position $x_i \in \mathbb{R}^n$, the velocity $v_i \in \mathbb{R}^n$, the personal best position $p_i \in \mathbb{R}^n$, and the neighborhood best position $g_i \in \mathbb{R}^n$.

The communication among the particles is modeled by defining a neighborhood $\mathcal{N}_i$ for every particle i. A variety of neighborhood topologies are possible. The most popular are the global topology, where all particles are connected to each other, and the ring topology, where the particles are arranged in a ring such that any particle has a neighbor to its left and its right. It is known that the global topology frequently produces a premature convergence of the population, therefore the ring topology is usually preferred [21].

Particles positions and velocities are randomly initialized. Then, following the most popular PSO update rules [21], at every generation, the velocity and the current position of every particle i are updated according to:

$$v_i \leftarrow w v_i + c_1 r_{1i}(p_i - x_i) + c_2 r_{2i}(g_i - x_i), \tag{1}$$

$$x_i \leftarrow x_i + v_i, \tag{2}$$

where $r_{1i}, r_{2i} \in [0, 1]$ are randomly generated at every step, and $w, c_1, c_2 \geq 0$ are the three PSO parameters called, respectively, inertial, cognitive and social coefficient. The objective function is evaluated on the new position x_i, i.e., $f(x_i)$ is computed, and the personal and neighborhood best are updated, in the case of minimization, according to:

$$p_i \leftarrow \arg\min \{f(x_i), f(p_i)\}, \tag{3}$$

$$g_i \leftarrow \arg\min \{f(p_j) : j \in \mathcal{N}_i\}, \tag{4}$$

where the $\arg\min$ has to be replaced with $\arg\max$ for maximization problems.

The rationale behind the design of equations (1–4) is that the move of a single particle in the search space is influenced by the superposition of three aspects:

- the inertial tendency to follow the previous search direction,
- the cognitive bias towards the best visited position so far, and
- the social disposition to move towards good solutions communicated by the neighbors.

In particular, these three aspects are modeled by the three terms of the velocity update rule (1).

Finally, note how PSO is so popular in the optimization community that a search for "particle swarm optimization" on Google Scholar produces around 221 000 results at the time of writing.

2.2. Random Key Decoders

Random key decoders are the most popular technique that supports the application of PSO to permutation-based optimization problems [7, 8, 9, 10].

The first random key decoder has been proposed in [11] in the context of genetic algorithms. Formally, by denoting with $\mathcal{S}_n$ the set of permutations of the integers in $\{1, \ldots, n\}$, [11] introduces a decoder function $RK : \mathbb{R}^n \to \mathcal{S}_n$ which can be used by PSO to optimize the objective function $f : \mathcal{S}_n \to \mathbb{R}$ of the permutation problem at hand. Therefore, the only modification to the classical PSO is to consider $f(RK(x))$ as the fitness value of every particle position $x \in \mathbb{R}^n$.

The decoder RK transforms $x \in \mathbb{R}^n$ to the permutation $\pi \in \mathcal{S}_n$ such that the sequence $x_{\pi(1)}, \ldots, x_{\pi(n)}$ is increasingly ordered. For example, if $x = (0.46, 0.91, 0.33, 0.75, 0.51)$, its corresponding permutation is $\pi = RK(x) = \langle 3, 1, 5, 4, 2 \rangle$. Therefore, RK requires to sort the component indexes of x basing on their corresponding values.

Also a simple variant of RK has been considered in the literature, see for instance [10, 22]. In this variant, a vector $x \in \mathbb{R}^n$ is decoded to the permutation $\rho \in \mathcal{S}_n$ such that $\rho(i) = r_i$, where r_i is the rank of x_i among the vector components $x_1, \ldots, x_n$ sorted in ascending order. Using the numeric vector of the previous example, $\rho = \langle 2, 5, 1, 4, 3 \rangle$. It is easy to see that the permutation ρ is the inverse permutation of π, i.e., $\rho = \pi^{-1}$, and vice versa. Hence, though not explicitly reported in the literature, this decoding scheme, to which we refer with RKI, can be obtained by inverting the result of RK and vice versa, i.e., $RKI(x) = (RK(x))^{-1}$ and $RK(x) = (RKI(x))^{-1}$.

We refer to the two random key based PSO schemes with the acronyms PSO+RK and PSO+RKI. Moreover, without loss of generality, in both RK and RKI, we only consider to sort the vector components in ascending order, while ties are randomly broken.

3. Algebraic Framework for Combinatorial Search Spaces

In this section we provide a concise description of the algebraic framework for evolutionary computation previously proposed in [12], together with its extension introduced in [14]. The framework is based on the notion of *finitely generated group* and the related algebraic and geometric concepts. Its aim is to introduce the operations $\oplus$, $\ominus$, $\odot$ on the set of discrete solutions in such a way that they simulate, as much as possible, the analogous vector operations of the Euclidean space.

3.1. Search Spaces and Finitely Generated Groups

The triplet $G = (X, \star, H)$ is a finitely generated group representing a combinatorial search space if and only if:

- X is the discrete set of solutions in the search space;
- $\star : X \times X \to X$ is a binary operation on X which satisfies the group properties: associativity, existence of the identity $e \in X$, and existence of the inverse $x^{-1} \in X$ for any $x \in X$; if $\star$ is also commutative, the group is Abelian;

- $H \subseteq X$ is a finite generating set of the group, i.e., any $x \in X$ can be decomposed as $x = h_1 \star \cdots \star h_l$ for some $h_1, \ldots, h_l \in H$.

A decomposition $x = h_1 \star \cdots \star h_l$ of $x \in X$ is minimal if there exists no other decomposition $x = h'_1 \star \cdots \star h'_m$ with $m < l$. The length l of a minimal decomposition of x is the weight of x and it is denoted by $|x|$.

Given a finitely generated group $G = (X, \star, H)$, its Cayley graph $\mathcal{C}(G)$ is the labelled digraph whose vertexes are the solutions in X and there exists an arc from x to y labelled by $h \in H$ if and only if $y = x \star h$.

In the Cayley graph, for all $x \in X$, every directed path from e to x corresponds to a decomposition of x: if the arcs labels occurring in the path are $\langle h_1, h_2, \ldots, h_l \rangle$, then $x = h_1 \star h_2 \star \cdots \star h_l$. As a consequence, shortest paths from e to x correspond to minimal decompositions of x. More generally, a shortest path from x to y, where $x, y \in X$, corresponds to a minimal sequence of generators $\langle h_1, h_2, \ldots, h_l \rangle$ such that $x \star (h_1 \star h_2 \star \cdots \star h_l) = y$. Hence, $\langle h_1, h_2, \ldots, h_l \rangle$ is a minimal decomposition of $x^{-1} \star y$.

The diameter D of $\mathcal{C}(G)$ is defined as the maximal weight of the elements in X. Moreover, an interesting partial order relation, which will be useful later, is defined as follows. For $x, y \in X$, $x \sqsubseteq y$ if and only if there exists (at least) a shortest path from e to y passing by x. For the sake of presentation, here we focus on groups with a unique maximal weight element ω such that $x \sqsubseteq \omega$ for all $x \in X$. The concrete group considered later belongs to such a class.

The Cayley graph has an important geometric interpretation. Indeed, a sequence of generators $\langle h_1, h_2, \ldots, h_l \rangle$ can be seen as a *vector* which connects a starting *point* $x \in X$ to the end *point* $y = x \star (h_1 \star h_2 \star \cdots \star h_l)$. On the other hand, any element $x \in X$ can be decomposed as a sequence of generators $\langle h_1, h_2, \ldots, h_l \rangle$ and therefore it can be considered as a *free vector*. The dichotomous interpretation of the elements of X, as points and as vectors, allows to define the operations $\oplus, \ominus, \odot$ on X which simulate the analogous operations of the Euclidean space.

3.2. Addition and Subtraction

The addition $z = x \oplus y$ is defined as the application of the vector $y \in X$ to the point $x \in X$. The result z is computed by choosing a decomposition $\langle h_1, h_2, \ldots, h_l \rangle$ of y and by finding the end point of the path which starts from x and whose arcs labels are $\langle h_1, h_2, \ldots, h_l \rangle$, i.e., $z = x \star (h_1 \star h_2 \star \cdots \star h_l)$. By noting that $h_1 \star h_2 \star \cdots \star h_l = y$, the addition $\oplus$ is independent from the generating set and it is uniquely defined as

$$x \oplus y := x \star y. \tag{5}$$

Continuing the analogy with the Euclidean space, the difference between two points is a vector. Given $x, y \in X$, the difference $y \ominus x$ produces the sequence of labels $\langle h_1, h_2, \ldots, h_l \rangle$ in a path from x to y. Since $h_1 \star h_2 \star \cdots \star h_l = x^{-1} \star y$, we can replace the sequence of labels with its composition, thus making the difference independent from the generating set. Therefore, $\ominus$ is uniquely defined as

$$y \ominus x := x^{-1} \star y. \tag{6}$$

Both $\oplus$ and $\ominus$, like their numerical counterparts, are consistent to each other. Indeed, $x \oplus (y \ominus x) = y$ for all $x, y \in X$. Moreover, both operations are not commutative (unless the group is Abelian), $\oplus$ is associative, and e is its neutral element.

3.3. Scalar Multiplication

Again, as in the Euclidean space, it is possible to multiply a vector by a non-negative scalar. Given $a \geq 0$ and $x \in X$, we denote their multiplication with $a \odot x$.

We first provide the conditions that $a \odot x$ has to verify in order to simulate, as much as possible, the scalar multiplication of vector spaces:

(C1) $|a \odot x| = \lceil a \cdot |x| \rceil$;

(C2) if $a \in [0, 1]$, $a \odot x \sqsubseteq x$;

(C3) if $a \geq 1$, $x \sqsubseteq a \odot x$.

Clearly, the scalar multiplication of $\mathbb{R}^n$ satisfies the slight variant of (C1) where the Euclidean norm replaces the group weight and the ceiling is omitted. Besides, similarly to scaled vectors in $\mathbb{R}^n$, (C2) and (C3) intuitively encode the idea that $a \odot x$ is the element x scaled down or up, respectively.

It is important to note that, fixed a and x, there may be more than one element of X satisfying (C1–C3). This is a clear consequence of the non-uniqueness of the minimal decomposition of x. Therefore, different strategies can be devised to compute $a \odot x$. Nevertheless, our aim is to apply the operation in an evolutionary algorithm, therefore we denote with $a \odot x$ a randomly selected element satisfying (C1–C3).

Note also that the diameter D induces an upper bound on the possible values for the scalar a. Indeed, for any $x \in X$, let $\overline{a}_x = \frac{D}{|x|}$, if $a > \overline{a}_x$, (C1) would imply $|a \odot x| > D$, but this is impossible. Therefore, similarly to the out-of-bounds handling techniques of continuous evolutionary algorithms [23], we define

$$a \odot x := \overline{a}_x \odot x, \quad \text{when } a > \overline{a}_x. \tag{7}$$

The multiplication $a \odot x$ can be computed by:

1. randomly selecting a shortest path from e to ω passing by x, and
2. composing the first $\lceil a \cdot |x| \rceil$ generators on its arcs.

Since any sub-path of a shortest path is itself a shortest path, and by also considering that shortest paths correspond to minimal decompositions, it is easy to see that the conditions (C1–C3) are satisfied.

Let $l = |x|$, we can observe that the sequence of generators $\langle h_1, \ldots, h_l, \ldots, h_D \rangle$ on the chosen shortest path can be divided in two parts: $\langle h_1, \ldots, h_l \rangle$ and $\langle h_{l+1}, \ldots, h_D \rangle$. The former is a minimal decomposition of x, while the latter minimally decomposes $\omega \ominus x = x^{-1} \star \omega$. Operatively, only one of the sub-paths is used to compute $a \odot x$. When $a \leq 1$, the generators to compose are all in the first sub-path $\langle h_1, \ldots, h_l \rangle$. Conversely, for $a > 1$, it is sufficient to compose x with the first $\lceil a \cdot l \rceil - l$ generators in the second sub-path.

The pseudo-codes of the two procedures for $a \in [0,1]$ and $a > 1$ are reported, respectively, in Figures 1 and 2. Both rely on the abstract procedure $RandDec$ which is assumed to return a random minimal decomposition of the element in input. An implementation of $RandDec$ has to consider the particularities of the concrete finitely generated group at hand. Note also that $Extend$ implements equation (7).

```
1: function TRUNCATE(a ∈ [0,1], x ∈ X)
2:     s ← RandDec(x)
3:     l ← Length(s)
4:     k ← ⌈a · l⌉
5:     z ← e
6:     for i ← 1 to k do
7:         z ← z ⋆ s_i
8:     end for
9:     return z
10: end function
```

Figure 1. Truncation algorithm for computing $a \odot x$ when $a \in [0,1]$

```
1: function EXTEND(a > 1, x ∈ X)
2:     s ← RandDec(x^{-1} ⋆ ω)
3:     l ← D − Length(s)                ▷ Because |x| = |ω| − |x^{-1} ⋆ ω| = D − |x^{-1} ⋆ ω|
4:     ā_x = D / l
5:     a ← min{a, ā_x}
6:     k ← ⌈a · l⌉
7:     z ← x
8:     for i ← 1 to k − l do
9:         z ← z ⋆ s_i
10:    end for
11:    return z
12: end function
```

Figure 2. Extension algorithm for computing $a \odot \pi$ when $a > 1$

3.4. Vector-like Operations for the Symmetric Group

In this paper we focus on the "symmetric group" $\mathcal{S}_n$ which is the group of all the permutations of the set $[n] = \{1, \ldots, n\}$. The group operation is the permutation composition operator $\circ$. Given $\pi, \rho \in \mathcal{S}_n$, their composition $\pi \circ \rho$ is defined as the permutation $(\pi \circ \rho)(i) = \pi(\rho(i))$ for all the indexes $i \in [n]$. $\mathcal{S}_n$ is not Abelian and its identity is the permutation e such that $e(i) = i$ for all $i \in [n]$.

Therefore, as in the abstract definitions (5) and (6), given $\pi, \rho \in \mathcal{S}_n$, $\oplus$ and $\ominus$ are implemented as:

$$\pi \oplus \rho := \pi \circ \rho, \tag{8}$$

$$\rho \ominus \pi := \pi^{-1} \circ \rho. \tag{9}$$

Moreover, different generating sets are possible in $\mathcal{S}_n$ (see [14] and [24]). Here, we focus on the subset of the $n-1$ simple transpositions, i.e., the set $ST = \{\sigma_i \in \mathcal{S}_n : 1 \leq i < n\}$ where σ_i is defined as: $\sigma_i(i) = i+1$, $\sigma_i(i+1) = i$, and $\sigma_i(j) = j$ for $j \in [n] \setminus \{i, i+1\}$. Considering ST, the maximum weight element of $\mathcal{S}_n$ is the permutation ω such that $\omega(i) = n+1-i$ for all $i \in [n]$. The diameter D is then $\binom{n}{2}$.

It is now important to note that, simple transpositions, when composed to the right of a permutation, act as adjacent swap moves. By also considering that the classical bubble sort algorithm sorts an array, thus a permutation, by employing an optimal number of adjacent swaps, the randomized decomposition algorithm for the finitely generated group $(\mathcal{S}_n, \circ, ST)$ is a randomization of the classical bubble-sort algorithm. It has been called $RandBS$ and is presented in Figure 3. $RandBS$ sorts π in increasing order (hence obtaining e) by iteratively choosing a random adjacent swap move from the set of "adjacent inversions" A. The correctness of $RandBS$ has been proven in [12].

```
 1: function RandBS(π)
 2:     s ← ⟨ ⟩
 3:     A ← {σ_i ∈ ST : i < i+1 and π(i) > π(i+1)}
 4:     while A ≠ ∅ do
 5:         σ ← select an element from A uniformly at random
 6:         π ← π ∘ σ
 7:         s ← Concatenate(⟨σ⟩, s)
 8:         A ← Update(A, σ)                          ▷ Θ(1) complexity
 9:     end while
10:     return s
11: end function
```

Figure 3. Randomized decomposition algorithms for permutations

By replacing $RandDec$ and $\star$ in the algorithms of Figures 1 and 2 with, respectively, $RandBS$ and $\circ$, we have a concrete implementation of $\odot$ for the symmetric group.

Finally note that $\oplus$ and $\ominus$ cost $\Theta(n)$, while $\odot$ costs $\Theta(n^2)$.

4. Algebraic Particle Swarm Optimization

Here we introduce the Algebraic Particle Swarm Optimization (APSO) meta-heuristic. Basing on the previously described vector-like operations for permutation objects, APSO directly works on the discrete space of permutations without using any decoder function. Most importantly, APSO meaningfully brings the search dynamics of continuous PSO in the combinatorial space of permutations.

In Section 4.1, we first propose a basic version of APSO, then Section 4.2 argues about an anomaly in its inertial movement that we address by proposing the inertia-preserving APSO.

4.1. Basic APSO

The basic version of Algebraic Particle Swarm Optimization, namely APSO, analogously to its continuous counterpart, iteratively evolves a population of N particles, each one with its own position, velocity, personal and neighborhood best. However, differently from continuous PSO, APSO particles navigate the space of permutations and they aim to optimize the objective function $f : \mathcal{S}_n \to \mathbb{R}$ of a given permutation-based optimization problem.

Formally, the i-th particle of APSO population is composed by: the current position $\chi_i \in \mathcal{S}_n$, the velocity $\nu_i \in \mathcal{S}_n$, the personal best position $\pi_i \in \mathcal{S}_n$, and the neighborhood best position $\gamma_i \in \mathcal{S}_n$. The neighborhood structure $\mathcal{N}_i$ is defined exactly as in classical PSO (see Section 2.1). Importantly, we underline that in APSO both particle positions and velocities are permutations. Indeed, they have to be intended as, respectively, vertexes and paths on the Cayley graph of permutations, i.e., the search space navigated by APSO.

In principle, APSO can be defined by simply replacing the numeric operations in the PSO move equations (1) and (2) with the algebraic operators introduced in Section 3. Hence, by separately defining the three velocity terms of the i-th particle as:

$$\theta_i^{(I)} = w \odot \nu_i, \tag{10}$$

$$\theta_i^{(C)} = (c_1 r_{1i}) \odot (\pi_i \ominus \chi_i), \tag{11}$$

$$\theta_i^{(S)} = (c_2 r_{2i}) \odot (\gamma_i \ominus \chi_i), \tag{12}$$

APSO iteratively updates the velocity and position of the i-th particle according to:

$$\nu_i \leftarrow \theta_i^{(I)} \oplus \theta_i^{(C)} \oplus \theta_i^{(S)}, \tag{13}$$

$$\chi_i \leftarrow \chi_i \oplus \nu_i, \tag{14}$$

where, exactly as in continuous PSO, $r_{1i}, r_{2i} \in [0, 1]$ are randomly generated at every step, and $w, c_1, c_2 \geq 0$ are the three inertial, cognitive and social parameters.

However, care has to be taken due to the non-commutativity of the $\oplus$ operator. While, in the position update equation (14) it is reasonable to interpret the first term as a "point" and the second term as a "vector", the three terms of the velocity update equation (13) are all "vectors", thus, apparently, there is no preferred ordering to add them. Besides, some preliminary experiments confirm that no specific ordering clearly outperforms the others. Therefore, in order to handle the non-commutativity of $\oplus$, in equation (13) we add the three terms $\theta_i^{(I)}, \theta_i^{(C)}, \theta_i^{(S)}$ by randomly selecting one of their $3! = 6$ possible orderings.

After a particle has been moved, its new position in the Cayley graph is evaluated, i.e., $f(\chi_i)$ is computed, and the personal and neighborhood best are updated as in classical PSO, i.e., in the case of minimization:

$$\pi_i \leftarrow \arg\min \{f(\chi_i), f(\pi_i)\}, \tag{15}$$

$$\gamma_i \leftarrow \arg\min \{f(\pi_j) : j \in \mathcal{N}_i\}. \tag{16}$$

4.2. Inertia-preserving APSO

Since the algebraic structure induced by our framework is not exactly a vector space, the basic APSO previously introduced cannot faithfully simulate the inertial movement of its continuous counterpart.

In particular, let analyze the pathological case depicted in Figure 4 where, for the sake of clarity, we slightly modified the notation by omitting the particle index and introducing the generation index.

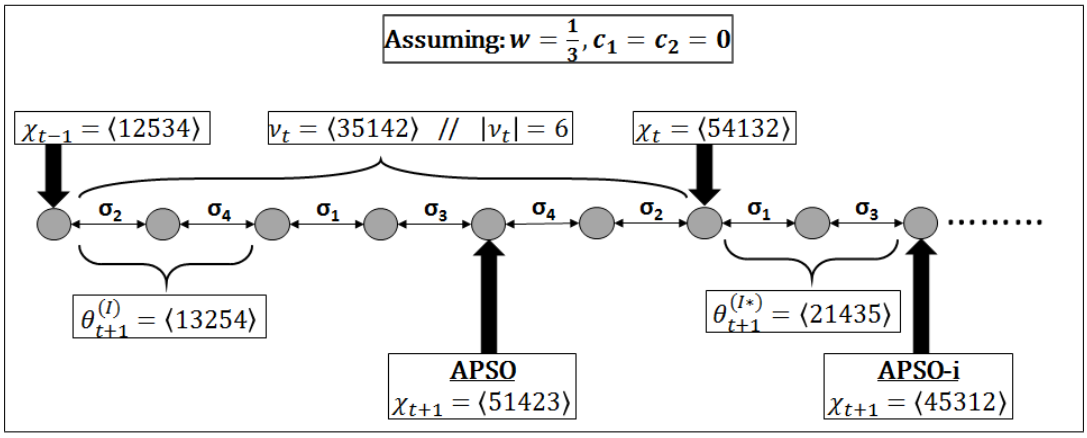

Figure 4. Pathological example of the inertial move

We imagine to be at time step $t+1$. We know the previous positions χ_{t-1}, χ_t and, by equation (14), the previous velocity $\nu_t = \chi_t \ominus \chi_{t-1}$. Now, we want to compute ν_{t+1} and χ_{t+1}. We simplified the scenario by choosing $c_1 = c_2 = 0$, thus we have $\nu_{t+1} = \theta^{(I)}_{t+1}$ and we only consider the computation $\chi_{t+1} = \chi_t \oplus \theta^{(I)}_{t+1}$.

The rationale of the classical PSO inertial move is that the distance between χ_{t+1} and χ_{t-1} should be larger than the distance between χ_t and χ_{t-1}. Practically, in the figure we should have χ_{t+1} at the right of χ_t. However, since all the simple transpositions are inverses of themselves (i.e., $\sigma = \sigma^{-1}$ for any $\sigma \in ST$), in Figure 4, it happens that χ_{t+1} is at the left of χ_t, thus closer to χ_{t-1}. Essentially, the particle, instead of moving beyond χ_t and away from χ_{t-1}, moves back towards χ_{t-1}.

The pathological situation happens for all the velocities with a minimal decomposition s such that $s = prp^R$, i.e., s is the concatenation of three subsequences (p, r, and p^R), and the suffix p^R is the reverse of the prefix p.

One may argue that this is an infrequent scenario and we can simply ignore it. However, the non-uniqueness of minimal decompositions makes the issue very intrusive.

More in general, anomalous behaviors, with respect to the inertial interpretation, arise whenever, given $\pi \in \mathcal{S}_n$ and considering the first k generators $\sigma_{j_1}, \ldots, \sigma_{j_k}$ in one of the minimal decompositions of π, we have that the permutation $\rho = \pi \circ (\sigma_{j_1} \circ \ldots \circ \sigma_{j_k})$ is such that $|\rho| < |\pi| + k$.

In order to quantify these anomalies, we have conducted the following experiment. We considered $n = 50$ and a number of generators k from 1 to 10. For each value of k, we randomly sampled 10 000 permutations and, for each one, we verified the previously provided property. The percentage of anomalies on the considered sample, as the value of k changes, is depicted in the chart of Figure

5. The chart shows that, when $k = 1$ around half of the sampled permutations present anomalies. Moreover, the percentage quickly increases with k and, yet from $k \geq 7$, almost all the totality of the sampled permutations is anomalous. Therefore, these anomalous situations are the rule and not an exception, thus an inertia-preserving mechanism is of paramount importance.

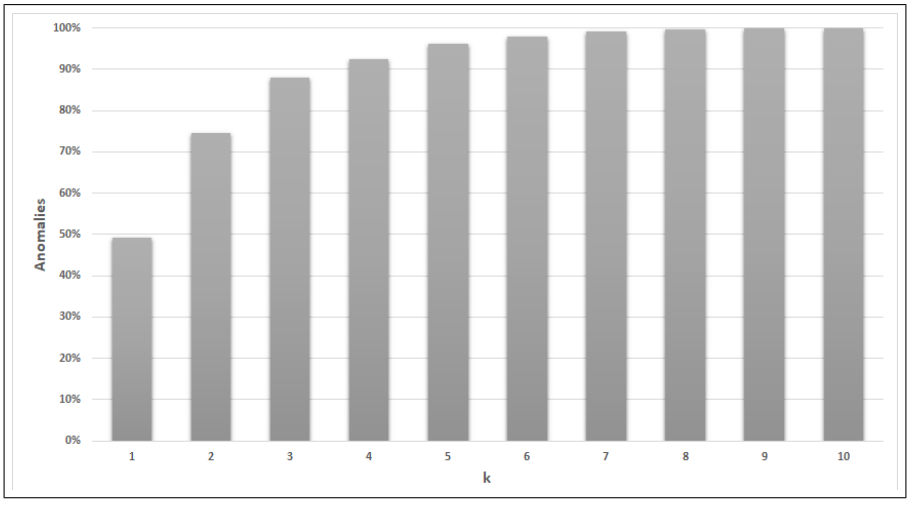

Figure 5. Percentages of permutations π, ρ such that $|\rho| < |\pi| + k$

The algebraic reason behind the pathology is that, in our framework, differently from the real vector space, we have that

$$w \odot \nu \neq [(1 + w) \odot \nu] \ominus \nu. \tag{17}$$

Therefore, in order to fix these situations, we propose an inertia-preserving variant of APSO, namely APSO-i, which redefines equation (10) by setting the inertial term to be the right hand side of the inequality (17).

Formally, APSO-i employs the same move equations (13) and (14) of APSO, except that the term $\theta_i^{(I)}$ is replaced by $\theta_i^{(I*)}$, defined according to

$$\theta_i^{(I*)} = [(1 + w) \odot \nu_i] \ominus \nu_i. \tag{18}$$

Figure 4 also shows how the pathological example is avoided using the inertia-preserving scheme APSO-i. Indeed, $\theta^{(I*)}$ can be considered to work in two steps:

1. the operation $(1 + w) \odot \nu_i$ extends the decomposition of ν_i, and
2. the subtraction by ν_i removes the first $|\nu_i|$ generators already "consumed" in the previous movement of the particle.

Hence, exactly as for $\theta^{(I)}$, $|\theta^{(I*)}| = \lceil w \cdot |\nu_i| \rceil$, but now the generators are selected in the same trajectory followed by the particle at the previous step.

5. Barycentric APSO

Here we introduce the Barycentric APSO (APSO-b) which is a major variant of APSO mainly based on: (i) a geometrically consistent formulation of convex combinations between permutations, and (ii) an alternative interpretation of the continuous PSO movement scheme.

Section 5.1 shows how to implement a convex combination between two permutations, while Section 5.2 provides the barycentric interpretation of classical PSO moves. Basing on these ideas, APSO-b and its inertial counterpart APSO-bi are presented in Section 5.3.

5.1. Convex combination of permutations

In the Euclidean space, the vector $z = ax + by$ is a convex combination of the vectors $x, y \in \mathbb{R}^n$ if and only if the two scalar factors are such that $a, b \geq 0$ and $a + b = 1$. In geometric terms, the point $z \in \mathbb{R}^n$ lies in the segment connecting the points x and y, while the scalars a and b indicate how close or far z is with respect to the two endpoints. Hence, z is a weighted barycenter of the points x and y.

We now need a meaningful way to replicate the concept of convex combination in the permutations space. Actually, we require to preserve, in the Cayley graph, the geometric property held by the barycenter in the continuous space, i.e., being a point in the segment between the two given endpoints. Since a segment is a shortest path between two points of $\mathbb{R}^n$, we can analogously consider the shortest paths of the Cayley graph. Therefore, given two generic permutations $\pi, \rho \in \mathcal{S}_n$, their barycenter $\beta \in \mathcal{S}_n$ has to be a permutation lying in a shortest path between π and ρ in the Cayley graph.

Unfortunately, the algebraic framework for permutations introduced in Section 3 does not allow to directly discretize the convex combination formula by also preserving its geometric interpretation. Indeed, if the weighted barycentric permutation is computed as $\beta = (a \odot \pi) \oplus (b \odot \rho)$ (where, as before, $a, b \geq 0$ and $a + b = 1$), then there is no guarantee that β will reside in a shortest path between π and ρ. This is due to the fact that $\odot$ has a meaningful interpretation when applied to a vector-like permutation, while $\oplus$ is meaningful when its operands are, respectively, a point-like and a vector-like permutation (see Sections 3.2 and 3.3).

Nevertheless, we can exploit a simple equivalence of the numerical convex combination formula and discretize this one. Indeed, it is easy to see that, given the points $x, y \in \mathbb{R}^n$ and $a, b \geq 0$ such that $a + b = 1$, we have the two equivalences:

$$ax + by = y + a(x - y), \tag{19}$$

$$ax + by = x + b(y - x), \tag{20}$$

which exploit that, respectively, $b = 1 - a$ and $a = 1 - b$.

Let analyze what happens if we set the barycentric permutation to the discretized version of the right hand side of one of the two equations (19) and (20). For instance, let us choose equation (19). Given $a \in [0, 1]$ and $\pi, \rho \in \mathcal{S}_n$ playing the role of, respectively, x and y, the weighted barycenter β is computed as $\beta = \rho \oplus a \odot (\pi \ominus \rho)$. It is now easy to prove that β has the required geometric property. Indeed, a minimal decomposition of $\pi \ominus \rho$ produces the sequence of generators in a shortest path (of the Cayley graph) from ρ to π. The multiplication by the scalar $a \in [0, 1]$ truncates this sequence that, when right-summed to ρ, produces the permutation β residing in a shortest path between ρ and

π in the Cayley graph. The discretization of equation (20) works similarly, though in the opposite direction, i.e., from π to ρ.

Note that, since the discretized convex combination formula contains $\odot$, actually, the barycentric permutation is a random variable. Furthermore, discretizing the right hand sides of equations (19) or (20) is almost equivalent. Indeed, since $a + b = 1$, and due to the Cayley graph properties, both random variables have the same support, though their densities may slightly differ. Therefore, for the sake of fairness, given $\pi, \rho \in \mathcal{S}_n$ and $a, b \geq 0$ such that $a + b = 1$, we define the computation of the weighted barycentric permutation β as

$$\beta = \begin{cases} \rho \oplus a \odot (\pi \ominus \rho) & \text{with probability } 1/2, \\ \pi \oplus b \odot (\rho \ominus \pi) & \text{with probability } 1/2. \end{cases} \tag{21}$$

5.2. Barycentric interpretation of PSO moves

Here we further analyze the continuous PSO by showing an equivalent formulation of the velocity update rule that allows to derive an alternative interpretation of particles' dynamics.

Indeed, it is possible to rewrite the velocity update rule (1) of classical PSO in such a way that the sum of the cognitive and social terms is replaced by a unique "barycentric term".

For the sake of presentation, we remove the particle index i and we denote $a_1 = c_1 r_1$, $a_2 = c_2 r_2$, and $a = a_1 + a_2$. Then, the sum $c_1 r_1 (p - x) + c_2 r_2 (g - x)$ in equation (1) can be rewritten as follows:

$$\begin{aligned} & c_1 r_1 (p - x) + c_2 r_2 (g - x) = \\ & = a_1 (p - x) + a_2 (g - x) = \\ & = a_1 p + a_2 g - a x = \\ & = a \left[\left(\tfrac{a_1}{a} p + \tfrac{a_2}{a} g \right) - x \right]. \end{aligned} \tag{22}$$

Now, it is easy to see that the two factors $\frac{a_1}{a}$ and $\frac{a_2}{a}$ are non-negative and sum up to 1, thus the part inside the round brackets is actually a convex combination of the personal and neighborhood best positions for the considered particle. We denote this barycentric point with β, i.e., $\beta = \frac{a_1}{a} p + \frac{a_2}{a} g$.

Now, plugging β into the equivalence (22), we have

$$c_1 r_1 (p - x) + c_2 r_2 (g - x) = a(\beta - x), \tag{23}$$

thus the whole PSO velocity update rule (1) is equivalent to

$$v \leftarrow w v + a(\beta - x). \tag{24}$$

Therefore, a particle in continuous PSO actually moves by combining two search directions: the inertial one, and the tendency to follow the trajectory going from x to β, i.e., from the previous particle position to the weighted barycenter of the particle's personal and neighborhood best positions.

5.3. APSO-b and APSO-bi

By putting it all together the notions of Sections 5.1 and 5.2, we provide the definitions of, respectively, APSO-b and APSO-bi, i.e., the barycentric variants of APSO and APSO-i.

To this aim, note that the velocity update rules (1) and (24), though being equivalent in $\mathbb{R}^n$, differ when discretized by means of the $\oplus$, $\ominus$ and $\odot$ operations. Indeed, the vector-like operations of $\mathcal{S}_n$ do not satisfy the commutative and distributive properties, thus the equivalences in equation (22) do not hold in the discrete case.

Nevertheless, the new barycentric interpretation allows us to introduce APSO-b as a variant of APSO based on the discretization of the "barycentric" velocity update rule (24).

Using the same notation of APSO, for each particle i, APSO-b modifies the velocity update rule (13) by replacing the two velocity terms $\theta_i^{(C)}$ and $\theta_i^{(S)}$ with the single barycentric term $\theta_i^{(B)}$ as follows:

$$\beta_i \leftarrow \begin{cases} \gamma_i \oplus \left(\frac{c_1 r_{1i}}{c_1 r_{1i} + c_2 r_{2i}} \odot (\pi_i \ominus \gamma_i) \right) & \text{with probability } 1/2, \\ \pi_i \oplus \left(\frac{c_2 r_{2i}}{c_1 r_{1i} + c_2 r_{2i}} \odot (\gamma_i \ominus \pi_i) \right) & \text{with probability } 1/2, \end{cases} \tag{25}$$

$$\theta_i^{(B)} = (c_1 r_{1i} + c_2 r_{1i}) \odot (\beta_i \ominus \chi_i), \tag{26}$$

$$\nu_i \leftarrow \theta_i^{(I)} \oplus \theta_i^{(B)}. \tag{27}$$

First, note that we expanded back the coefficients $a_1 = c_1 r_{1i}$, $a_2 = c_2 r_{2i}$, and $a = a_1 + a_2$. By following definition (21), the barycenter β_i, between particle's personal and neighborhood best π_i and γ_i, is computed by means of equation (25). The barycentric term $\theta_i^{(B)}$ of equation (26) is obtained as the discretization of the right hand side of equation (23). Then, APSO-b adopts the velocity update rule (27) which is the discretization of the alternative PSO velocity formulation of equation (24). Moreover, similarly to the basic APSO, the two possible orderings of rule (27) are randomly chosen at every velocity computation. Regarding position update, the rule (14) of basic APSO is kept unchanged in APSO-b.

Finally, it is now straightforward to introduce APSO-bi, i.e., the inertia-preserving variant of APSO-b. Indeed, APSO-bi uses a simple variant of the velocity update rule (27), where the term $\theta_i^{(I)}$ is replaced with $\theta_i^{(I*)}$ (see Section 4.2).

6. Experiments

The aim of the experimental comparison is not to provide a state-of-the-art algorithm for the problem at hand, but to verify if our proposal, though being closer to the search philosophy at the basis of the continuous PSO, is also competitive with respect to the standalone PSO+RK schemes, i.e., the other general approach that allows to apply PSO to permutation-based optimization problems.

Experiments have been held on the four most popular permutation-based optimization problems, i.e.: the linear ordering problem (LOP), the permutation flowshop scheduling problem (PFSP), the quadratic assignment problem (QAP), and the traveling salesman problem (TSP). A total of 32 instances, equally divided among the four problems, have been selected: 16 instances for the parameters tuning and 16 instances for the algorithms comparison.

The name of the instances, together with the objective function formulations, are provided in Table 1. All the selected instances come from widely adopted benchmark suites: LOLIB[1] for LOP, Taillard benchmark suite[2] for PFSP, QAPLIB[3] for QAP, and TSPLIB[4] for TSP.

The PFSP problem has been investigated using the total flowtime as objective criterion [12], while, for TSP, we have adopted the objective function formulation that fixes the last city in the tour [25], thus allowing a one-to-one correspondence between TSP tours and permutations of $n-1$ cities.

Table 1. Benchmark Problems and Instances

Problem	Objective Function	Definition of symbols	Tuning Instances	Test Instances
LOP	$\max_{\pi\in\mathcal{S}_n}\left\{\sum_{i=1}^{n}\sum_{j=i+1}^{n} M_{\pi(i),\pi(j)}\right\}$	M is the $n\times n$ I/O matrix	N-be75np N-be75tot N-tiw56n66 N-tiw56r72	N-be75eec N-stabu74 N-tiw56n54 N-t69r11xx
PFSP	$\min_{\pi\in\mathcal{S}_n}\left\{\sum_{i=1}^{n} c_{\pi(i),m}\right\}$	$c_{\pi(i),j}=p_{\pi(i),j}+\max\left\{c_{\pi(i-1),j}, c_{\pi(i),j-1}\right\}$ $c_{i,0}=c_{0,j}=0$ $p_{i,j}$ is the processing time of job i on machine j n and m are the number of jobs and machines	tai50_5_1 tai50_10_0 tai50_10_5 tai50_20_1	tai20_10_0 tai50_5_0 tai50_20_0 tai100_10_0
QAP	$\min_{\pi\in\mathcal{S}_n}\left\{\sum_{i=1}^{n}\sum_{j=1}^{n} F_{i,j}D_{\pi(i),\pi(j)}\right\}$	F is the $n\times n$ flow matrix D is the $n\times n$ distance matrix	tai25a tai25b tai30a tai30b	tai15a tai15b tai40a tai40b
TSP	$\min_{\pi\in\mathcal{S}_{n-1}}\left\{\sum_{i=1}^{n-2} d_{\pi(i),\pi(i+1)}+d_{\pi(n-1),n}+d_{n,\pi(1)}\right\}$	$d_{i,j}$ is the distance between cities i and j n is the number of cities	bayg29 swiss42 gr48 hk48	fri26 bays29 dantzig42 berlin52

The algorithms investigated are the four APSO implementations here presented, i.e., APSO, APSO-i, APSO-b and APSO-bi, together with the two random key based PSOs, i.e., PSO+RK and PSO+RKI. The widely adopted ring topology has been consider in order to connect the particles population. Furthermore, all the algorithms have been implemented without using any additional technique such as local searches, heuristic functions, restart mechanisms, etc. In this way we are able to clearly compare the effectiveness of APSOs and PSO+RKs dynamics.

Due to the different characteristics of the search spaces navigated by the algebraic and random key based algorithms, in order to perform a fair comparison, the parameters of the six algorithms have been separately tuned using SMAC [26], i.e., a popular software tool for automatic algorithm configuration based on statistical and machine learning techniques. To avoid the over-tuning phenomenon [27], SMAC calibrations have been run using a separate set of instances with respect to those used in the experiments for algorithms comparison (see Table 1). Every SMAC calibration has been set to perform 2000 executions, while every execution terminates after $1000n^2$ fitness evaluations have been performed. The ranges of the parameters (in input to SMAC), together with the parameters configurations produced by SMAC, are provided in Table 2. Interestingly, the calibrations of APSO-i and APSO-bi produced values of the inertial coefficient w larger than in APSO and APSO-b. This looks to be a first validation of the reasons previously discussed for the introduction of the inertial APSO.

The six algorithms, configured using the SMAC indications, have been compared on the 16 se-

[1]LOLIB instances are available at http://www.optsicom.es/lolib/.

[2]PFSP Taillard instances are available at http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

[3]QAPLIB instances are available at http://anjos.mgi.polymtl.ca/qaplib/.

[4]TSPLIB instances are available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Table 2. Parameters Tuning

Algorithm	Ranges of the Parameters	Tuned Parameters
APSO	$N \in \{20, 60, 100\}$	$N = 20$
	$w \in [0, 0.8]$	$w = 0.04$
	$c_1 \in [0.2, 1.8]$	$c_1 = 1.18$
	$c_2 \in [0.2, 1.8]$	$c_2 = 1.61$
APSO-i	$N \in \{20, 60, 100\}$	$N = 100$
	$w \in [0, 0.8]$	$w = 0.16$
	$c_1 \in [0.2, 1.8]$	$c_1 = 0.25$
	$c_2 \in [0.2, 1.8]$	$c_2 = 1.27$
APSO-b	$N \in \{20, 60, 100\}$	$N = 20$
	$w \in [0, 0.8]$	$w = 0.01$
	$c_1 \in [0.2, 1.8]$	$c_1 = 1.29$
	$c_2 \in [0.2, 1.8]$	$c_2 = 1.61$
APSO-bi	$N \in \{20, 60, 100\}$	$N = 100$
	$w \in [0, 0.8]$	$w = 0.08$
	$c_1 \in [0.2, 1.8]$	$c_1 = 1.78$
	$c_2 \in [0.2, 1.8]$	$c_2 = 0.73$
PSO+RK	$N \in \{20, 60, 100\}$	$N = 100$
	$w \in [0, 0.8]$	$w = 0.80$
	$c_1 \in [0.2, 1.8]$	$c_1 = 1.52$
	$c_2 \in [0.2, 1.8]$	$c_2 = 1.77$
PSO+RKI	$N \in \{20, 60, 100\}$	$N = 100$
	$w \in [0, 0.8]$	$w = 0.74$
	$c_1 \in [0.2, 1.8]$	$c_1 = 0.31$
	$c_2 \in [0.2, 1.8]$	$c_2 = 1.80$

lected test instances reported in Table 1. Every algorithm has been executed 20 times per instance and, again, $1000n^2$ has been used as the budget of fitness evaluations in every run. The final fitness values produced by the runs of every algorithm have been aggregated for each instance using the Average Relative Percentage Deviation (ARPD) measure which is computed according to

$$ARPD_{Inst}^{Alg} = \frac{1}{20} \sum_{i=1}^{20} \frac{\left| Alg_{Inst}^{(i)} - Best_{Inst} \right|}{Best_{Inst}} \times 100 \tag{28}$$

where $Alg_{Inst}^{(i)}$ is the final fitness value produced by the algorithm Alg in its i-th run on the instance $Inst$, while $Best_{Inst}$ is the best result obtained by any algorithm in any run on the given instance. Note that, both for maximization and minimization problems, the ARPD values have to be minimized.

The ARPDs obtained in this experimental session are provided in Table 3 that also shows averaged

ARPD values together with non-parametric measures such as the average ranks among the competitors. Finally, in order to establish the statistical significance of the differences among the performances of the competitor algorithms, as suggested in [28], the non-parametric Friedman statistical test and the Finner post-hoc procedure have been performed on the results. The p-values are provided in the last line of Table 3.

Table 3. Experimental Results with Fitness Evaluations Budget

Problem	**Instance**	**APSO**	**APSO-i**	**APSO-b**	**APSO-bi**	**PSO+RK**	**PSO+RKI**
LOP	N-be75eec	1.80	0.88	1.66	**0.66**	10.53	2.54
	N-stabu74	3.42	1.88	3.19	**1.53**	10.78	5.62
	N-tiw56n54	2.52	1.31	2.11	**0.58**	12.14	4.73
	N-t69r11xx	1.29	0.59	0.97	**0.31**	8.74	2.76
Avg ARPD on LOP		2.26	1.17	1.98	**0.77**	10.54	3.91
Avg Rank on LOP		4	2	3	**1**	6	5
PFSP	tai20_10_0	1.79	1.18	1.47	**0.84**	5.35	4.35
	tai50_5_0	3.50	2.30	2.44	**1.87**	9.65	5.45
	tai50_20_0	2.13	2.06	2.79	**1.94**	5.89	4.66
	tai100_10_0	1.44	1.39	1.97	**1.33**	5.94	5.57
Avg ARPD on PFSP		2.21	1.73	2.17	**1.49**	6.71	5.01
Avg Rank on PFSP		3.5	2	3.5	**1**	6	5
QAP	tai15a	4.67	4.08	4.15	**3.14**	6.52	7.22
	tai15b	0.37	0.33	0.37	**0.28**	0.72	0.81
	tai40a	1.39	1.42	1.51	**1.09**	4.22	4.76
	tai40b	10.06	**5.18**	9.73	5.96	19.21	19.28
Avg ARPD on QAP		4.12	2.75	3.94	**2.62**	7.67	8.02
Avg Rank on QAP		3.5	2	3.25	**1.25**	5	6
TSP	fri26	15.03	7.23	14.66	**5.88**	29.95	33.69
	bays29	22.16	16.44	20.19	**15.43**	57.36	46.30
	dantzig42	32.45	24.53	31.85	21.24	44.93	**7.54**
	berlin52	18.66	11.91	15.21	**11.14**	50.04	51.99
Avg ARPD on TSP		22.08	15.03	20.48	**13.42**	45.57	34.88
Avg Rank on TSP		4.25	2.25	3.25	**1.25**	5.5	4.5
Avg ARPD on all instances		7.67	5.17	7.14	**4.58**	17.62	12.95
Avg Rank on all instances		3.81	2.06	3.25	**1.13**	5.63	5.13
Friedman+Finner p-value		0	< 0.001	$< 10^{-11}$	**best**	0	0

Table 3 clearly shows that the algebraic algorithms outperform the random key based PSOs. Indeed, all the four APSOs outperformed the best of the two PSO+RK schemes on 15 instances over 16. In particular, the barycentric and inertia-preserving APSO, i.e., APSO-bi, looks to be the best algorithm on 14 instances over 16 (and the second one in the remaining two instances). Also the very small p-values of the statistical test largely validate the superiority of APSO-bi. Therefore, the main

conclusions we draw from this session of experiments are:

- APSO schemes are empirically better than the classical random key based PSOs with high statistical evidence,
- the proposal of the inertia-preserving scheme is validated by the superior performances of APSO-bi and APSO-i with respect to, respectively, APSO-b and APSO.
- analogously as in the previous point, the barycentric scheme obtained better performances than the "normal ones", i.e., APSO-bi and APSO-b outperformed, respectively, APSO-i and APSO,

The only weakness of the algebraic schemes has been registered on the TSP instance "dantzig42" where PSO+RKI outperforms the APSO schemes (that, anyway, are all better than PSO+RK). Finally, two last aspects that emerge from Table 3 are: (i) the larger ARPDs registered on the TSP instances may indicate that all the algorithms lost robustness on that problem, and (ii) PSO+RKI is almost always better than PSO+RK.

Due to the different time complexities — $\Theta(Nn^2)$ and $\Theta(Nn \log n)$ per generation, respectively, for APSOs and PSO+RKs —, the random key based algorithms reach the evaluations cap faster than the algebraic PSOs. Therefore, a further comparison has been performed using a termination criterion based on the computational time, thus allowing random key based PSOs to perform more fitness evaluations than APSOs, but both using an equal budget of time and the same parameters of Table 2. Therefore, in this experimental session, every algorithm execution terminates after n^2 seconds regardless of the evaluations performed. The ARPDs, the ranks and the statistical results are presented in Table 4 using the same layout of Table 3.

Table 4 largely confirms the indications of the previous comparison. Indeed, APSO-bi is again the best algorithm with high statistical evidence, and the worst algebraic scheme is better than the best random key PSO on 15 instances over 16. Again, the deceptive instance is "dantzig42". Other differences with respect to the results provided in Table 3 are: (i) APSO-i obtained comparable results with respect to APSO-bi on QAP instances, and (ii) PSO+RK obtains the same overall rank of PSO+RKI.

Therefore, either considering an equal number of fitness evaluations or an equal amount of computational time, algebraic algorithms clearly outperform the random key based PSOs. Furthermore, both the inertia-preserving and barycentric mechanisms improved the performance of the basic APSO scheme.

For the sake of completeness, in Table 5 we provide a comparison of the best objective values reached by APSO schemes with respect to the best known solutions available in literature for the selected instances.

Before analyzing these results it is important to remember that: (i) experiments were not aimed to obtain state-of-the-art solutions, (ii) we are tackling four NP-hard combinatorial problems quite different among them, (iii) exact algorithms have to be heavily or completely rethought to tackle all the problems, and (iv) in literature there is no general meta-heuristic that has been able to obtain top performances on all the different problems. In light of these considerations, it is possible to appreciate the comparison of Table 5, where APSO schemes were able to match 7 best known solutions. Finally, Table 5 clearly indicates that TSP looks to be the more difficult problem for APSO.

7. Conclusion and Future Work

In this paper, an Algebraic Particle Swarm Optimization (APSO) scheme for permutation-based optimization problems has been introduced.

The design of APSO uses an algebraic framework for combinatorial optimization based on strong mathematical foundations. The algebraic operators allow APSO to meaningfully brings the swarm intelligence dynamics of PSO in the discrete space of permutations. In particular, both particles positions and velocities are directly represented as permutations. Indeed, the underlying algebraic framework allows to interpret a permutation, not only as candidate solution, but also as a displacement ("vector") between solutions.

First, a basic APSO scheme has been proposed by directly discretizing the classical PSO move equations by means of the algebraic framework. However, an anomalous behavior of this approach has been illustrated, both analytically and experimentally, thus an inertia-preserving mechanism has been introduced in order to fix this issue. Furthermore, a barycentric APSO has been introduced by exploiting an alternative interpretation of classical PSO dynamics and a meaningful scheme to implement convex combinations in the permutations space. In total, four APSO schemes have been introduced.

Experiments have been held to compare the performances of the proposed APSO algorithms with the other PSO schemes for permutation problems in literature. In particular, APSOs have been compared with the popular random key based PSO schemes. Commonly adopted benchmark instances from four permutation problems have been considered. The experimental results clearly show, with high statistical evidence, that APSO outperforms the competitor algorithms. Therefore, a clear indication emerges for the researchers or practitioners that wish to apply PSO to permutation problems: before considering to build an algorithm basing on the random key technique, give a try to APSO.

Possible future lines of research include: the introduction of other generating sets (based on exchange and insertion moves), the design of a binary APSO, the hybridization of APSO with other improving techniques, and the applications to other problems [29].

References

[1] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. of IEEE Intern. Conf. on Neural Networks, volume 4. 1995 pp. 1942–1948.

[2] Milani A, Santucci V. Community of scientist optimization: An autonomy oriented approach to distributed optimization. *AI Communications*, 2012. **25**(2):157–172.

[3] Poli R, Kennedy J, Blackwell T. Particle swarm optimization: An overview. *Swarm Intelligence*, 2007. **1**(1):33–57.

[4] del Valle Y, Venayagamoorthy GK, Mohagheghi S, Hernandez JC, Harley RG. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. *IEEE Transactions on Evolutionary Computation*, 2008. **12**(2):171–195.

[5] Marini F, Walczak B. Particle swarm optimization (PSO). A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 2015. **149**:153 – 165. doi:https://doi.org/10.1016/j.chemolab.2015.08.020.

[6] Kennedy J, Eberhart RC. A discrete binary version of the particle swarm algorithm. In: Proc. of IEEE International Conference on Systems, Man, and Cybernetics, volume 5. 1997 pp. 4104–4108.

[7] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 2007. **177**(3):1930–1947.

[8] Ai TJ, Kachitvichyanukul V. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 2009. **36**(5):1693–1702.

[9] Koulinas G, Kotsikas L, Anagnostopoulos K. A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. *Information Sciences*, 2014. **277**:680–693.

[10] Gao H, Kwong S, Fan B, Wang R. A Hybrid Particle-Swarm Tabu Search Algorithm for Solving Job Shop Scheduling Problems. *IEEE Transactions on Industrial Informatics*, 2014. **10**(4):2044–2054.

[11] Bean JC. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 1994. **6**(2):154–160.

[12] Santucci V, Baioletti M, Milani A. Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem With Total Flowtime Criterion. *IEEE Transactions on Evolutionary Computation*, 2016. **20**(5):682–694.

[13] Santucci V, Baioletti M, Milani A. Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm. *AI Communications*, 2016. **29**(2):269–286.

[14] Baioletti M, Milani A, Santucci V. An Extension of Algebraic Differential Evolution for the Linear Ordering Problem with Cumulative Costs. In: Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings. 2016 pp. 123–133.

[15] Santucci V, Baioletti M, Milani A. A Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem with Total Flow Time Criterion. In: Parallel Problem Solving from Nature – PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings. Springer International Publishing, 2014 pp. 161–170.

[16] Baioletti M, Milani A, Santucci V. Linear Ordering Optimization with a Combinatorial Differential Evolution. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics. 2015 pp. 2135–2140.

[17] Baioletti M, Milani A, Santucci V. A discrete differential evolution algorithm for multi-objective permutation flowshop scheduling. *Intelligenza Artificiale*, 2016. **10**(2):81–95.

[18] Santucci V, Baioletti M, Milani A. An Algebraic Differential Evolution for the Linear Ordering Problem. In: Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings. 2015 pp. 1479–1480.

[19] Santucci V, Milani A, Vella F. A study on the synchronization behaviour of differential evolution and a self-adaptive extension. *Journal of Artificial Intelligence and Soft Computing Research*, 2012. **2**(4):279–301.

[20] Baioletti M, Milani A, Santucci V. Algebraic Particle Swarm Optimization for the permutations search space. In: 2017 IEEE Congress on Evolutionary Computation (CEC). 2017 pp. 1587–1594. doi:10.1109/CEC.2017.7969492.

[21] Bratton D, Kennedy J. Defining a Standard for Particle Swarm Optimization. In: Proc. of IEEE Swarm Intelligence Symposium. 2007 pp. 120–127.

[22] Ayodele M, McCall JAW, Regnier-Coudert O. RK-EDA: A Novel Random Key Based Estimation of Distribution Algorithm. In: Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings. 2016 pp. 849–858.

[23] van den Bergh F, Engelbrecht A. A study of particle swarm optimization particle trajectories. *Information Sciences*, 2006. **176**(8):937 – 971. doi:https://doi.org/10.1016/j.ins.2005.02.003.

[24] Schiavinotto T, Stützle T. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 2007. **34**(10):3143–3153.

[25] Gopal R, Rosmaita B, Van Gucht D. Genetic algorithms for the traveling salesman problem. In: Proc. of 1st International Conference on Genetic Algorithms and their Applications. 1985 pp. 160–165.

[26] Hutter F, Hoos HH, Leyton-Brown K. Sequential Model-Based Optimization for General Algorithm Configuration. In: Proc. of LION-5 (Learning and Intelligent Optimization Conference). 2011 pp. 507–523.

[27] Birattari M. Tuning Metaheuristics: A Machine Learning Perspective. Springer, 2009.

[28] Derrac J, Garca S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 2011. **1**(1):3–18.

[29] Mancini L, Milani A, Poggioni V, Chiancone A. Self regulating mechanisms for network immunization. *AI Communications*, 2016. **29**(2):301–317.

[30] Cagnina L, Esquivel S, Gallard R. Particle swarm optimization for sequencing problems: a case study. In: Proc. of Congress on Evolutionary Computation, volume 1. 2004 pp. 536–541.

Table 4. Experimental Results with Time Budget

Problem	**Instance**	**APSO**	**APSO-i**	**APSO-b**	**APSO-bi**	**PSO+RK**	**PSO+RKI**
LOP	N-be75eec	1.04	0.67	1.29	**0.47**	5.89	3.21
	N-stabu74	2.13	1.47	2.93	**0.81**	7.11	4.82
	N-tiw56n54	1.95	0.80	1.56	**0.48**	9.41	5.30
	N-t69r11xx	1.02	**0.20**	0.58	0.21	8.10	2.56
Avg ARPD on LOP		1.54	0.78	1.59	**0.49**	7.63	3.97
Avg Rank on LOP		3.5	1.75	3.5	**1.25**	6	5
PFSP	tai20_10_0	0.81	0.38	0.70	**0.22**	4.36	4.39
	tai50_5_0	2.95	**1.79**	2.59	1.85	8.73	7.61
	tai50_20_0	2.28	2.21	2.83	**2.10**	4.23	5.72
	tai100_10_0	2.69	2.32	2.26	**2.22**	5.61	6.06
Avg ARPD on PFSP		2.18	1.67	2.09	**1.60**	5.73	5.95
Avg Rank on PFSP		3.75	2	3	**1.25**	5.25	5.75
QAP	tai15a	1.59	**0.81**	2.14	0.89	3.79	6.90
	tai15b	0.16	0.07	0.20	**0.06**	0.56	0.66
	tai40a	1.09	**0.47**	0.90	0.54	2.21	5.50
	tai40b	10.58	7.31	11.24	**5.22**	15.26	25.03
Avg ARPD on QAP		3.35	2.16	3.62	**1.68**	5.45	9.52
Avg Rank on QAP		3.25	**1.5**	3.75	**1.5**	5	6
TSP	fri26	15.94	9.41	13.32	**8.35**	37.15	40.46
	bays29	9.35	11.56	11.86	**6.56**	39.35	40.63
	dantzig42	28.90	15.02	27.99	14.26	33.95	**6.12**
	berlin52	15.42	16.98	14.15	**10.70**	54.40	54.86
Avg ARPD on TSP		17.40	13.24	16.83	**9.97**	41.21	35.52
Avg Rank on TSP		3.5	3	3.25	**1.25**	5.25	4.75
Avg ARPD on all instances		6.12	4.47	6.03	**3.43**	15.01	13.74
Avg Rank on all instances		3.50	2.06	3.38	**1.31**	5.38	5.38
Friedman+Finner p-value		$< 10^{-8}$	0.018	$< 10^{-8}$	**best**	0	0

Table 5. Comparison with best known solutions

Problem	Instance	APSO b.s.	B.k.s.	Rel. Dev.
LOP	N-be75eec	236464	236464	**0%**
	N-stabu74	541393	541393	**0%**
	N-tiw56n54	91554	91554	**0%**
	N-t69r11xx	771149	771149	**0%**
PFSP	tai20_10_0	20911	20911	**0%**
	tai50_5_0	66049	64803	+1.92%
	tai50_20_0	130492	125831	+3.70%
	tai100_10_0	320410	299101	+7.12%
QAP	tai15a	388214	388214	**0%**
	tai15b	51765268	51765268	**0%**
	tai40a	3339846	3139370	+6.39%
	tai40b	654768846	637250948	+2.75%
TSP	fri26	958	937	+2.24%
	bays29	2159	2020	+6.88%
	dantzig42	925	699	+32.33%
	berlin52	11860	7542	+57.25%