Algebraic Particle Swarm Optimization for the Permutations Search Space

Marco Baioletti, Alfredo Milani, Valentino Santucci
Department of Mathematics and Computer Science
University of Perugia, Italy
Email: {marco.baioletti, alfredo.milani, valentino.santucci}@unipg.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 problems. As far as we know, the most popular PSO variants that produce 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 the original PSO design in permutations search space. APSO directly represents the particle positions and velocities as permutations. The APSO search scheme is based on a general algebraic framework for combinatorial optimization previously, and successfully, introduced in the context of discrete differential evolution schemes. 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 proposed for both the issues, and two APSO variants are provided. Experiments have been held to compare the performances of the proposed 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.**

I. 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.

Though it has been originally proposed for continuous search spaces, PSO variants for combinatorial optimization problems have increasingly appeared in literature [2], [3]. While the first discrete PSO has been introduced for binary problems [4], a large number of PSO applications to permutation based optimization problems has been proposed. See for instance [5], [6], [7], [8].

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 [9] for genetic algorithms. RK consists in a decoder function that converts a vector in $\mathbb{R}^n$ into a permutation of n integers by sorting the vector components. Using RK, permutation problems can be directly 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 [10], [11], [12], [13], [14], [15], we have proposed very effective algebraic differential evolution schemes for permutation problems. The classical differential evolution, 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, conversely from 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.

The Algebraic PSO (APSO), conversely from the previously proposed algebraic differential evolution, makes use of only algebraic operators, thus representing the first "completely algebraic" evolutionary algorithm proposal. Besides the direct application of the algebraic framework to PSO, a further study on how to simulate the PSO inertial move in the permutations

space has been conducted. Hence, we also provide a second implementation of APSO aiming to preserve the particle inertial trajectory similarly to what happen in its continuous counterpart. Moreover, differently from the past algebraic differential evolution schemes, the non-commutativity of the framework's addition operator highlighted a new design choice on the ordering of the PSO velocity terms.

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 II recall the classical PSO algorithm together with the random key techniques. The algebraic framework for combinatorial optimization is briefly described in Section III, while the APSO scheme is introduced in Section IV. The experimental analysis is provided in Section V, while conclusions are drawn in Section VI, where future lines of research are also depicted.

II. Particle Swarm Optimization and Random Key

The description of a standard continuous PSO is provided in Section II-A. This scheme has been used as the PSO reference implementation throughout the rest of the paper. Moreover, Section II-B recalls the random key approaches available in literature by also connecting them through a simple algebraic consideration.

A. 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 [16].

Particles positions and velocities are randomly initialized. Then, following the most popular PSO update rules [16], 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 update, 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: (i) the inertial tendency to follow the previous search direction, (ii) the cognitive bias towards the best visited position so far, and (iii) 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).

B. Random Key Decoders

The most popular technique that allows to apply PSO to permutation based optimization problems is the use of random key decoders. See for instance [5], [6], [7], [8].

The first random key decoder has been proposed in [9] in the context of genetic algorithms. Formally, by denoting with $\mathcal{S}_n$ the set of permutations of the integers in $\{1, \ldots, n\}$, [9] 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 literature, see for instance [8]. 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 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 $RK(x) = (RKI(x))^{-1}$.

Therefore, 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.

III. Algebraic Framework for Combinatorial Search Spaces

In this section we provide a concise description of the algebraic framework for evolutionary computation previously proposed in [10], together with its extension introduced in [12]. 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.

A. 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, but it is not required;
- $H \subseteq X$ 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 also 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.

B. 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 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 product, 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.

C. 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 properties 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 evolutionary algorithms, 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 out-of-bounds handling techniques of continuous evolutionary algorithms, 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: (i) randomly selecting a shortest path from e to ω passing by x, and (ii) 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 $x^{-1} \star \omega$. 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 take the first $\lceil a \cdot l \rceil - l$ generators in the second sub-path $\langle h_{l+1}, \ldots, h_D \rangle$ and compose them to the right of x.

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
```

Fig. 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)
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
```

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

D. Vector 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 [12] and [17]). 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 $\binom{n}{2}$.

Since simple transpositions act as 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 [10].

```
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
```

Fig. 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 $\Theta(n^2)$.

IV. Algebraic Particle Swarm Optimization

The Algebraic Particle Swarm Optimization (APSO), analogously to what happen for PSO in the continuous space, iteratively evolves a population of N particles in order to optimize the objective function $f : \mathcal{S}_n \to \mathbb{R}$ of a permutation-based optimization problem.

The i-th particle of the 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 the classical PSO. Importantly, note that both positions and velocities are permutations. Indeed, they have to be intended as, respectively, vertexes and paths on the Cayley graph of permutations.

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

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

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

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

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

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

$$x_i \leftarrow x_i \oplus v_i, \tag{14}$$

where, exactly as in the 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 coefficients.

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 there is no preferred ordering to add them. Additionally, preliminary experiments confirm that no specific ordering 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 is evaluated, i.e., $f(\chi_i)$ is computed, and the personal and neighborhood best are updated as in the classical PSO (see equations (3) and (4)).

The algebraic structure induced by our framework is not exactly a vector space. Therefore, pathological situations like the one depicted in Figure 4 may arise. Let us examine the example. For the sake of clarity, we omit the particle index but we introduce the generation index in the notation. We imagine to be at time step $t + 1$. We know the previous positions χ_{t-1}, χ_t and the previous velocity $\nu_t = \chi_t \ominus \chi_{t-1}$ (from equation (14)). We want to compute ν_{t+1} and χ_{t+1}. Since in the example $c_1 = c_2 = 0$, we have $\nu_{t+1} = \theta_{t+1}^{(I)}$, thus we only consider the computation $\chi_{t+1} = \chi_t \oplus \theta_{t+1}^{(I)}$. 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, 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}. This happens for all the velocities with a decomposition that has as suffix a reversed prefix of itself.

The reason of the pathological case above is that in our framework, differently from the real vector space, $w \odot \nu \neq [(1 + w) \odot \nu] \ominus \nu$.[1] Therefore, to address these situations, we also propose an inertia-preserving variant of APSO, namely APSO-i, which redefines equation (10) by setting the inertial term to be the right side of the inequality above. Formally, APSO-i employs the same move equations (13) and (14) of APSO, except the term $\theta_i^{(I)}$ which is replaced with $\theta_i^{(I*)}$ defined according to

$$\theta_i^{(I*)} = [(1 + w) \odot \nu_i] \ominus \nu_i. \tag{15}$$

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: (i) the operation $(1 + w) \odot \nu_i$ extends the decomposition of ν_i, and (ii) 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.

V. Experiments

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 in 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 I. All the selected instances come from widely adopted benchmark suites: LOLIB[2] for LOP, Taillard benchmark suite[3] for PFSP, QAPLIB[4] for QAP, and TSPLIB[5] for TSP. PFSP has been investigated using the total flowtime as objective criterion [10], while, for TSP, we have adopted the objective function formulation that fixes the last city in the tour [18], thus allowing a one-to-one correspondence between TSP tours and permutations of $n - 1$ cities.

The algorithms investigated are the two APSO implementations here presented, i.e., APSO and APSO-i, together with the two random key based PSOs, i.e., PSO+RK and PSO+RKI. The ring topology has been adopted in all the algorithms which have been implemented without using any additional technique such as local searches, heuristic functions, restart mechanisms, etc. Indeed, the aim of this empirical comparison is not to provide a state-of-the-art algorithm for the problem at hand, but to verify if our proposal, though being more

[1] Actually, this depends from the fact that $\oplus, \ominus, \odot$ induce an algebraic structure more general than a vector space, where the distributivity of the scalar multiplication with respect to the field addition does not worth.

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

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

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

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

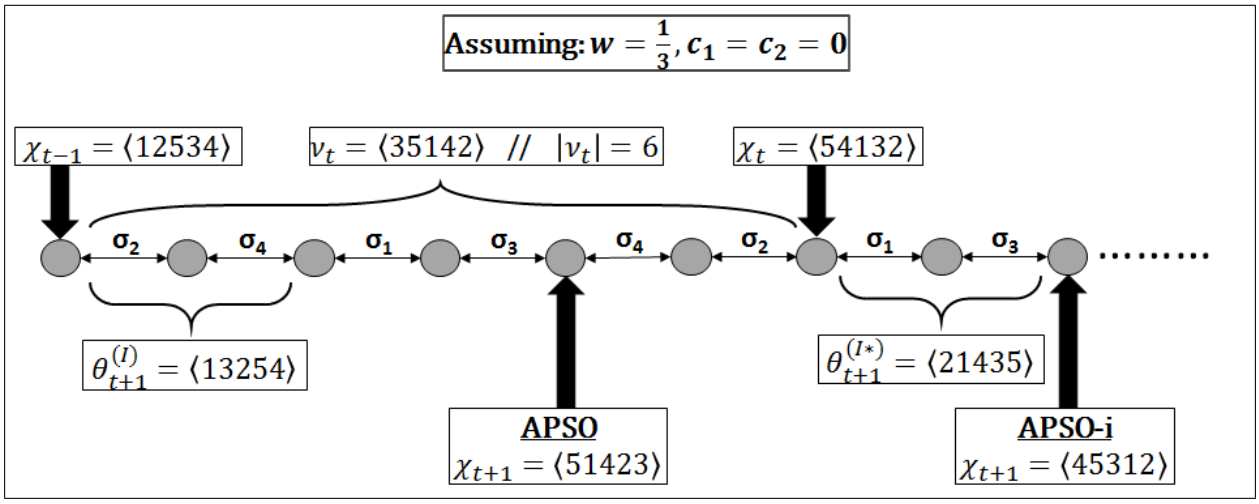

Fig. 4. Pathological example of the inertial move

TABLE I
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

close in some sense to the search philosophy at the basis of the continuous PSO, is also competitive with respect to the standalone PSO+RK schemes.

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 four algorithms have been separately tuned using SMAC [19], i.e., a popular software tool for automatic algorithm configuration based on statistical and machine learning techniques. To avoid the over-tuning phenomenon [20], SMAC calibrations have been run using a separate set of instances with respect to those used in the experiments for algorithms comparison (see Table I). 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 II. Interestingly, the calibrated value of the inertial coefficient w is four times bigger in APSO-i with respect to APSO where w is almost null. This looks to be a first validation of the reasons previously discussed for the introduction of the inertial APSO.

The four algorithms, configured using the SMAC indications, have been compared on the 16 selected test instances

TABLE II
PARAMETERS TUNING

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

reported in Table I. 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

$$ARPD_{Inst}^{Alg} = \frac{1}{20}\sum_{i=1}^{20}\frac{\left|Alg_{Inst}^{(i)} - Best_{Inst}\right| \times 100}{Best_{Inst}} \quad (16)$$

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 III 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 [21], 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 III.

TABLE III
EXPERIMENTAL RESULTS WITH FITNESS EVALUATIONS BUDGET

Problem	Instance	APSO	APSO-i	PSO+RK	PSO+RKI
LOP	N-be75eec	1.26	**0.32**	9.99	1.96
	N-stabu74	2.44	**1.01**	9.87	4.67
	N-tiw56n54	2.15	**0.96**	11.81	4.37
	N-t69r11xx	1.20	**0.50**	8.65	2.67
Avg ARPD on LOP		1.76	**0.70**	10.08	3.42
Avg Rank on LOP		2	**1**	4	3
PFSP	tai20_10_0	1.34	**0.76**	4.89	3.89
	tai50_5_0	3.49	**2.30**	9.65	5.45
	tai50_20_0	2.13	**2.06**	5.89	4.66
	tai100_10_0	1.16	**1.12**	5.65	5.28
Avg ARPD on PFSP		2.03	**1.56**	6.52	3.73
Avg Rank on PFSP		2	**1**	4	3
QAP	tai15a	3.47	**2.88**	5.30	5.99
	tai15b	0.35	**0.31**	0.71	0.80
	tai40a	**1.12**	1.15	3.94	4.48
	tai40b	10.06	**5.18**	19.21	19.28
Avg ARPD on QAP		3.75	**2.38**	7.29	7.64
Avg Rank on QAP		1.75	**1.25**	3	4
TSP	fri26	15.03	**7.23**	29.95	33.69
	bays29	22.16	**16.44**	57.36	46.30
	dantzig42	32.45	24.53	44.93	**7.54**
	berlin52	18.66	**11.91**	50.04	51.99
Avg ARPD on TSP		22.08	**15.03**	45.57	34.88
Avg Rank on TSP		2.25	**1.25**	3.5	3
Avg ARPD on all instances		7.40	**4.92**	17.36	12.69
Avg Rank on all instances		2	**1.12**	3.62	3.25
Friedman+Finner p-value		$< 10^{-3}$	**best**	$< 10^{-14}$	$< 10^{-12}$

Table III clearly shows that the algebraic algorithms outperform the random key based PSOs. Indeed, both APSO and APSO-i outperformed the best of the two PSO+RK schemes on 15 instances over 16. In particular the inertial variant of APSO, i.e., APSO-i, 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-i. Therefore, the two main conclusions we draw from this session of experiments are: (i) APSO schemes are empirically better than the classical random key based PSOs with an high statistical evidence, and (ii) the proposal of the inertia-preserving variant APSO-i is validated by its results with respect to the "normal" APSO. The only weakness of the algebraic schemes has been registered on the TSP instance "dantzig42" where PSO+RKI outperforms both APSO-i and APSO (that, anyway, are better than PSO+RK). Anyway, the reasons of why this instance is deceptive for all the competitors except PSO+RKI deserves further investigation. Finally, two last aspects that emerge from Table III are: (i) the larger ARPDs registered on the TSP instances may indicate that all the algorithms loss 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 by 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 the equal budget of time and the same parameters of Table II. Therefore, in this experimental session, every algorithm execution terminates after n^2 seconds. The ARPDs, the ranks and the statistical results are presented in Table IV using the same layout of Table III.

TABLE IV
EXPERIMENTAL RESULTS WITH TIME BUDGET

Problem	Instance	APSO	APSO-i	PSO+RK	PSO+RKI
LOP	N-be75eec	0.63	**0.34**	5.58	3.06
	N-stabu74	1.39	**0.60**	6.57	4.46
	N-tiw56n54	1.29	**0.43**	8.91	4.81
	N-t69r11xx	1.01	**0.17**	8.25	2.63
Avg ARPD on LOP		1.08	**0.39**	7.33	3.74
Avg Rank on LOP		2	**1**	4	3
PFSP	tai20_10_0	0.34	**0.12**	3.84	3.96
	tai50_5_0	2.29	**1.63**	8.16	6.96
	tai50_20_0	1.28	**0.78**	3.09	4.42
	tai100_10_0	1.10	**1.05**	4.52	4.80
Avg ARPD on PFSP		1.25	**0.89**	4.90	5.04
Avg Rank on PFSP		2	**1**	3.25	3.75
QAP	tai15a	1.60	**0.95**	3.65	6.67
	tai15b	0.14	**0.06**	0.55	0.71
	tai40a	1.20	**0.37**	2.59	5.41
	tai40b	9.11	**4.95**	12.47	22.33
Avg ARPD on QAP		3.01	**1.58**	4.81	8.78
Avg Rank on QAP		2	**1**	3	4
TSP	fri26	11.24	**8.42**	30.45	34.96
	bays29	**7.64**	11.69	36.93	41.87
	dantzig42	29.32	17.04	34.92	**6.66**
	berlin52	**4.29**	8.77	43.22	39.72
Avg ARPD on TSP		13.12	**11.48**	36.38	30.80
Avg Rank on TSP		**1.75**	2.25	3.5	3
Avg ARPD on all instances		4.62	**3.59**	13.36	12.09
Avg Rank on all instances		1.94	**1.19**	3.44	3.44
Friedman+Finner p-value		$< 10^{-2}$	**best**	$< 10^{-11}$	$< 10^{-11}$

Table IV largely confirms the indications of the previous comparison. Indeed, APSO-i 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 III are: (i) APSO obtains a better average rank with respect to APSO-i on the TSP problem, and (ii) PSO+RK obtains the same overall rank

of PSO+RKI.

In conclusion, either considering an equal number of fitness evaluations or an equal amount of computational time, algebraic algorithms clearly outperform the random key based PSOs, and, in particular, the inertia-preserving APSO-i seems to be the reference PSO scheme for permutation-based combinatorial optimization problems.

VI. 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 is based on an algebraic framework for combinatorial optimization with strong mathematical foundations. The algebraic operators allow APSO to simulate the dynamics of continuous PSO in the search space of permutations. In particular, in APSO both the positions and velocities of the particles 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.

Two variants of APSO have been proposed. The first one directly introduces the algebraic framework in the classical PSO move equations. However, pathological situations of this approach have been illustrated, and a second variant of APSO, i.e., APSO-i, has been proposed to address these cases. In APSO-i, the inertial term of the velocity update rule is handled analogously as in the continuous PSO.

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 random key based PSO schemes. Commonly adopted benchmark instances from four popular permutation problems have been considered. The experimental results clearly show, with high statistical evidence, that APSO outperforms the competitor algorithms.

Possible future lines of research include: a further investigation of the non-commutativity of the velocity terms, the introduction of other generating sets in APSO, the design of a binary APSO scheme, and the hybridization of APSO with other improving techniques.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[2] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[3] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, "Particle swarm optimization: Basic concepts, variants and applications in power systems," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 171–195, 2008.

[4] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 1997, pp. 4104–4108.

[5] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930–1947, 2007.

[6] T. J. Ai and V. Kachitvichyanukul, "A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery," *Computers & Operations Research*, vol. 36, no. 5, pp. 1693–1702, 2009.

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

[8] H. Gao, S. Kwong, B. Fan, and R. Wang, "A hybrid particle-swarm tabu search algorithm for solving job shop scheduling problems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2044–2054, 2014.

[9] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[10] V. Santucci, M. Baioletti, and A. Milani, "Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 682–694, 2016.

[11] ——, "Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm," *AI Communications*, vol. 29, no. 2, pp. 269–286, 2016.

[12] M. Baioletti, A. Milani, and V. Santucci, "An extension of algebraic differential evolution for the linear ordering problem with cumulative costs," in *Proc. of 14th International Conference on Parallel Problem Solving from Nature*, 2016, pp. 123–133.

[13] V. Santucci, M. Baioletti, and A. Milani, "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.

[14] M. Baioletti, A. Milani, and V. Santucci, "Linear ordering optimization with a combinatorial differential evolution," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015, pp. 2135–2140.

[15] ——, "A discrete differential evolution algorithm for multi-objective permutation flowshop scheduling," *Intelligenza Artificiale*, vol. 10, no. 2, pp. 81–95, 2016.

[16] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. of IEEE Swarm Intelligence Symposium*, 2007, pp. 120–127.

[17] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Computers & Operations Research*, vol. 34, no. 10, pp. 3143–3153, 2007.

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

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. of LION-5 (Learning and Intelligent Optimization Conference)*, 2011, pp. 507–523.

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

[21] J. Derrac, S. Garca, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.