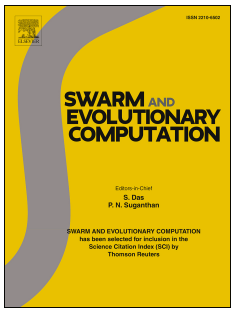



# An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization

Valentino Santucci, Marco Baioletti, Alfredo Milani

PII: S2210-6502(19)30109-9

DOI: https://doi.org/10.1016/j.swevo.2020.100673

Reference: SWEVO 100673

To appear in: *Swarm and Evolutionary Computation BASE DATA*

Received Date: 7 February 2019

Revised Date: 24 February 2020

Accepted Date: 2 March 2020

Please cite this article as: V. Santucci, M. Baioletti, A. Milani, An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization, *Swarm and Evolutionary Computation BASE DATA* (2020), doi: https://doi.org/10.1016/j.swevo.2020.100673.

# An Algebraic Framework for Swarm and Evolutionary Algorithms in Combinatorial Optimization

Valentino Santucci[a,*], Marco Baioletti[b], Alfredo Milani[b]

[a] *Department of Humanities and Social Sciences, University for Foreigners of Perugia, Italy*
[b] *Department of Mathematics and Computer Science, University of Perugia, Italy*

**Abstract**

A popular trend in evolutionary computation is to adapt numerical algorithms to combinatorial optimization problems. For instance, this is the case of a variety of Particle Swarm Optimization and Differential Evolution implementations for both binary and permutation-based optimization problems. In this paper, after highlighting the main drawbacks of the approaches in literature, we provide an algebraic framework which allows to derive fully discrete variants of a large class of numerical evolutionary algorithms to tackle many combinatorial problems. The strong mathematical foundations upon which the framework is built allow to redefine numerical evolutionary operators in such a way that their original movements in the continuous space are simulated in the discrete space. Algebraic implementations of Differential Evolution and Particle Swarm Optimization are then proposed. Experiments have been held to compare the algebraic algorithms to the most popular schemes in literature and to the state-of-the-art results for the tackled problems. Experimental results clearly show that algebraic algorithms outperform the competitors and are competitive with the state-of-the-art results.

*Keywords:* Algebraic Evolutionary Algorithms, Combinatorial search spaces, Algebraic Evolutionary Computation

*Corresponding author
*Email address:* valentino.santucci@unistrapg.it (Valentino Santucci)

## 1. Introduction

Swarm and evolutionary meta-heuristics are widely applied to solve complex optimization problems where traditional techniques are not able to produce good solutions in a reasonable amount of time. By a slight abuse of terminology, in this article the term Evolutionary Algorithms (EAs) is used to refer to meta-heuristics based on both evolutionary and swarm intelligence principles.

Depending on the nature of the solution set, it is possible to distinguish between continuous and combinatorial optimization problems and, as a consequence, between numerical and combinatorial EAs. The former work with solutions represented as real vectors, while the latter tackle problems whose solutions are discrete objects like, for example, bit-strings or permutations.

Though there exist EAs specifically designed to evolve discrete solutions, the wide availability of algorithms for continuous optimization has given rise to a plethora of numerical EAs adapted to solve combinatorial problems. Among them, see for instance the algorithms described in [1, 2, 3, 4, 5]. One possible motivation for this proliferation is that many nature-inspired algorithms have been historically defined for numerical problems. Therefore, in order to solve a combinatorial problem with a natural principle, it is often easier to adapt an existing numerical algorithm than to design a new combinatorial EA.

One of the most used and general technique to adapt a numerical EA to combinatorial problem is to employ decoding procedures which transform numeric vectors into valid discrete solutions for the representation at hand. For instance, the random-key decoder [6] converts a vector in $\mathbb{R}^n$ to a permutation of $n$ integers by sorting the vector indexes according to the corresponding vector values. Decoder-based schemes have been also proposed to transform numeric vectors into bit-strings [7, 2]. However, this approach has several issues:

- often, the reported good results are only obtained by hybridizing the adapted EA with other techniques (local searches, heuristic functions, restart mechanisms, etc.) and, as far as we know, no study is provided to understand if the adapted EA alone is effective or not;

- due to obvious cardinality reasons, a single discrete solution can be encoded by a potentially infinite number of numeric vectors, thus introducing large plateaus in the fitness landscape navigated by the underlying algorithm;
- the intuition of how the EA searches and moves in the continuous space, for which it has been originally designed, is completely lost when the algorithm is integrated with a decoding procedure and its moves are observed in the combinatorial space.

In a previous series of papers [8, 9, 10, 11, 12, 13], we have proposed discrete evolutionary algorithms, based on algebraic properties of the permutation space, which have reached remarkable results on the permutation flowshop scheduling and the linear ordering problem.

The main contribution of this article is the extension of the approach to a general algebraic framework by which it is possible to derive algebraic variants of many of the numerical EAs available in literature in order to tackle a large class of combinatorial optimization problems. Conversely from most of the decoder-based schemes, an algebraic algorithm directly evolves a population of discrete solutions by redefining the evolutionary operators of the numerical EA from which it is derived in such a way that the original movements in the continuous space are simulated in the discrete space.

The proposed method requires that the solution set $X$ forms a finitely generated group. This algebraic structure automatically gives rise to neighborhood relations on $X$, thus the search space can be seen as a graph of interconnected solutions. For binary and permutation problems, the induced search spaces are exactly the same spaces widely considered in the fields of local searches and fitness landscape analysis [14, Ch. 5]. In analogy to what happens in $\mathbb{R}^n$, the algebraic structure also allows to dichotomously interpret discrete solutions both as points and as displacements (i.e., vectors) between points in the space. Hence, it is possible to introduce the operations of addition, subtraction, and scalar multiplication on $X$ with similar properties as the corresponding vector

operations of $\mathbb{R}^n$. Therefore, these operations allow to consistently redefine the move equations of most numerical EAs in combinatorial search spaces.

To show the potential of the framework, in this paper we provide Algebraic EAs (AEAs) derived from differential evolution [15] and particle swarm optimization [16]. Abstract definitions, valid for any finitely generated group, are introduced for both AEAs together with their implementations for the search spaces of bit-strings and permutations. Note anyway that the framework is general enough to allow the discretization of other numerical algorithms such as, for instance, the firefly algorithm [17] or the bacterial foraging optimization [18].

Experiments have been held with the aim of comparing the effectiveness of the proposed algorithms with respect to the numerical decoder-based EAs and the state-of-the-art results. NK landscapes [19] have been considered as binary benchmarks, while, for the permutation search space, the experiments have been held on standard instances of the permutation flowshop scheduling [20] and linear ordering problem [21].

In the first set of experiments, both our algorithms and the competitors' have been implemented in their standalone versions. A second set of experiments has been held in order to compare AEAs with the corresponding numerical decoder-based EAs by incorporating additional techniques such as heuristic functions, local search procedures, restart mechanisms, and self-adaptive strategies. Furthermore, the AEAs results have been compared with the best known solutions in literature.

The rest of the paper is organized as follows. Section 2 describes the most used techniques for adapting continuous EAs to discrete problems, and Section 3 provides a critical analysis of the decoder-based approach. Section 4 provides mathematical background concepts used in Section 5, where the algebraic structure and the vector operations of combinatorial search spaces are introduced. The definitions of the algebraic EAs are then provided in Section 6. The experimental analysis is provided in Section 7. Finally, Section 8 concludes the paper by also providing future research directions.

## 2. Related work

### *2.1. Differential Evolution and Particle Swarm Optimization*

Among the numerical evolutionary algorithms in literature, Differential Evolution and Particle Swarm Optimization are the most studied and effective. Here we briefly describe them.

The Differential Evolution (DE) algorithm has been originally introduced in [15]. Its key operator is the differential mutation that, in the most used variant rand/1, for every population individuals $x_i$, generates a mutant $y_i$ according to the following formula

$$y_i \leftarrow x_{r_0} + F \cdot (x_{r_1} - x_{r_2}), \tag{1}$$

where $F \geq 0$ is the scale factor parameter of DE [22], and $x_{r_0}, x_{r_1}, x_{r_2}$ are three randomly chosen population individuals different from each other and with respect to $x_i$. Then, a crossover operator is applied to $y_i$ and $x_i$ to generate the trial solution $z_i$ that, if fitter than $x_i$, replaces it in the next generation population.

The Particle Swarm Optimization (PSO) algorithm has been originally introduced in [16]. PSO iteratively evolves a population of so-called particles. The $i$-th particle of the population is composed by the current position vector $x_i$, the velocity vector $v_i$, the personal best $p_i$, and the neighborhood best $g_i$. 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. One of the most used is the ring topology, where the particles are statically arranged in a ring such that any particle has a neighbor to its left and one to its right. In PSO, at every generation, the velocity and the current position of every particle $i$ are updated according to:

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

$$x_i \leftarrow x_i + v_i, \tag{3}$$

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. Then, the new position $x_i$ updates the personal best $p_i$ if fitter, while $g_i$ is replaced with the fittest vector among the particles in $\mathcal{N}_i$.

### *2.2. Combinatorial variants of numerical EAs*

Although DE and PSO are defined for continuous problems, in the literature there are innumerable attempts to use them in combinatorial optimization. Often, they differ from each other in few details, thus we are here interested in taxonomizing the approaches used for the discretization of a numerical EA. With this regard, two main classes of approaches can be distinguished.

In the first class, the evolutionary algorithms are based on the redefinition of the numerical operators of addition, subtraction, and multiplication. In this way, they can directly use discrete objects with formulae similar to (1), (2), and (3). Most algorithms of this class use ad-hoc definitions of the operators, thus preserving little more than the name of the numerical EAs from which they are derived. For example, the discrete operations used in [23, 24, 25, 26] have just a vague resemblance with the corresponding numerical versions.

In other cases, a more principled method of defining the operators is adopted. For instance, Set-based PSO [27] and DE [28] evolve a set representation of the discrete solutions by employing set-theoretical operators. These schemes have been applied to the traveling salesman problem and the multidimensional knapsack problem. Their main difference with respect to our approach is that they require a new set-based representation of the solutions, while our proposal adopts the classical (binary and permutation) representations. Another important difference is that [27, 28] require the concept of dimension (which is not required in our proposal) and may introduce constraints among the dimensions of a solution. For this reason, it is difficult to encode, by using the set-based representation, the solutions of a generic permutation-based problem. Indeed, though in [27] a set-based representation for the traveling salesman problem (TSP) is presented, this one relies on the fact that a TSP solution is a collection

of arcs (with constraints), thus it cannot be generalized to other permutation problems such as the ones tackled in the experimental part of this work.

More general methods based on the operators' redefinition which are somehow in line with our algebraic EAs can be found in [29] and [30]. In these works, the subtraction operation $x - y$ is defined in terms of a sequence of moves which transform $y$ into $x$. As we will see later, our proposal belongs to this class and its main difference with [29, 30] is that, in our approach, solutions and moves between solutions use the same representation, thus allowing additions, subtractions and scalar multiplications without any restriction. Furthermore, our approach is general enough to cover different search spaces.

The second class of approaches is based on decoder procedures. Discrete solutions of combinatorial problems are often represented using a proper subset $X$ of the numeric vectors in $\mathbb{R}^n$, i.e., $X \subset \mathbb{R}^n$. For instance, bit-strings are $0/1$ vectors, while a permutation can be represented as a vector of (all different) integers. However, these vectors are transformed by the move operators to vectors which almost always lie outside the feasible space $X$. To overcome this issue, continuous-to-discrete decoding schemes have been proposed to transform a numeric vector into a valid solution. Practically, decoder procedures can be devised for any representation and incorporated in any numerical algorithm. For this reason, we focus on decoder-based schemes on the experimental comparison provided in Section 7. However, though widely used in literature (see for example [7, 2, 31, 5]), this approach has some drawbacks. The most important one is that the intuition of how the underlying numerical EA searches and moves in the continuous space for which it has been originally designed is totally lost when the same algorithm is integrated with a decoding procedure and its search moves are observed in the combinatorial space. In the following, for the sake of comparison, we briefly describe the most popular combinatorial algorithms based on numerical decoders.

### 2.3. Binary EAs based on Probabilistic Decoders

Binary variants of DE [32] and PSO [7], to which we refer to as BDE and BPSO, aim to optimize an objective function of the form $f : \mathbb{B}^n \to \mathbb{R}$, where $\mathbb{B} = \{0, 1\}$. They evolve a population of bit-strings by mainly using the move operators of their numerical counterparts. Then, as soon as an unfeasible vector is generated, it is transformed back to a valid binary solution by means of a probabilistic decoder. Formally, given the non-binary vector $x \in \mathbb{R}^n$, each of its components $x_i$ is converted to 1 or 0 with probability $S(x_i)$ and $1 - S(x_i)$, respectively. The sigmoid function $S(t) = (1 + e^{-t})^{-1}$ is used to monotonically map any real number $t$ to a probability value $S(t) \in [0, 1]$.

In BDE [32], only the differential mutation of equation (1) can generate a non-binary vector. Hence, the components of a mutant $y \in \mathbb{R}^n$ are converted to bit values according to the probability given by $S\left(\frac{2b(y_i-0.5)}{1+2F}\right)$, where $b > 0$ is a further algorithmic parameter called *bandwidth factor*.

BPSO [7] encodes particle positions as binary strings and velocities as numeric vectors. Hence, a velocity vector $v \in \mathbb{R}^n$ is updated as usual using equation (2). Then, $v$ is used to generate the update probabilities for its corresponding particle position $x$, i.e., the $i$-th bit of $x$ is set to 1 or 0 with probability $S(v_i)$ and $1 - S(v_i)$, respectively.

Applications of BDE and BPSO have been proposed, for instance, in [33, 34, 35, 36]. Finally, there have been proposals to use, in place of the sigmoid function, other mathematical functions which generate probabilities: a review of them can be found in [1].

### 2.4. Angle Modulated EAs

The angle modulation technique, first introduced in [37], allows to transform a four-dimensional numeric vector to an $n$-length binary string, for any dimensionality $n$.

Formally, [37] introduces a decoder function $AM : \mathbb{R}^4 \to \mathbb{B}^n$ which can be used by any numerical algorithm $\mathcal{A}$ to optimize an objective function of the

form $f : \mathbb{B}^n \to \mathbb{R}$. The only modification to $\mathcal{A}$ is to consider $f(AM(x))$ as the fitness value of a generic individual $x \in \mathbb{R}^4$.

Let $x = (a, b, c, d)$ and its corresponding binary string be $y = AM(x)$, then the $i$-th bit of $y$ is computed as

$$y(i) = \begin{cases} 1 & \text{if } g(i-1) > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $g(t) = \sin(2\pi \cdot (t-a) \cdot b \cdot \cos(2\pi \cdot (t-a) \cdot c)) + d$ is called *generating function* in [37].

Angle modulated variants of DE and PSO, i.e., AM-DE and AM-PSO, have been introduced in, respectively, [2] and [37], while further applications are proposed in [38, 39].

### *2.5. Random-Key based EAs*

The Random-Key (RK) technique has been introduced in [6] to tackle permutation optimization problems.

Formally, [6] introduces a decoding function $RK : \mathbb{R}^n \to \mathcal{S}_n$ (where $\mathcal{S}_n$ is the set of permutations of $[n] = \{1, \ldots, n\}$) which can be used by any numerical algorithm $\mathcal{A}$ to optimize an objective function of the form $f : \mathcal{S}_n \to \mathbb{R}$. Also in this case, the only modification to $\mathcal{A}$ is to consider $f(RK(x))$ as the fitness value of a generic individual $x \in \mathbb{R}^n$.

$RK$ transforms $x$ to the permutation $\pi$ such that the sequence $x_{\pi(1)}, \ldots, x_{\pi(n)}$ is increasingly ordered. For example, if $x = (0.46, 0.91, 0.33, 0.75, 0.51)$, the decoded permutation is $\pi = RK(x) = \langle 3, 1, 5, 4, 2 \rangle$. Therefore, $RK$ requires to sort the component indexes of $x$ according to their corresponding values. This can be done in $\Theta(n \log n)$ time.

Also a simple variant of RK has been considered in literature, see for instance [40] and [5]. In this variant, a vector $x$ is decoded to the permutation $\rho$ 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 increasing order. It is easy to see that this decoding scheme can be obtained by inverting the result of $RK$, i.e., $\rho = (RK(x))^{-1}$. Therefore,

we generalize random-key by considering the parametrized decoder $RK^k$ where $k = \pm 1$.

Random-key variants of DE and PSO, i.e., RK-DE and RK-PSO, have been used in many works. See for instance [31, 41, 3, 42, 43].

## 3. Critical analysis of the decoder-based approaches

The decoder-based approaches have two inherent drawbacks: a single discrete solution can be encoded by a potentially infinite number of numeric vectors, and the distance relationships among the discrete objects can be completely upset after the embedding in the continuous space.

The probabilistic decoders described in Section 2.3, though their non-deterministic nature can sometime help to exit search stagnation, introduce a large amount of spatial distortion. Indeed, given a fixed numeric vector, performing multiple decoding steps can result in very different bit-strings. As an extreme example, let consider a BPSO particle with a zero vector as velocity. Since $S(0) = 0.5$, there is equal probability to have a 0 or a 1 for every bit in the decoded bit-string. Hence, the zero vector can be decoded in any one of the $2^n$ bit-strings with uniform probability, thus completely losing its original "identity".

Regarding the angle modulated approach (see Section 2.4), no explicit proof that equation (4) allows to cover the whole space $\mathbb{B}^n$ for any $n$ is provided in the literature. Moreover, by measuring the distances in the numeric and binary spaces by, respectively, the Euclidean and Hamming distance, we have that distant numeric vectors may correspond to close bit-strings and vice versa. For instance, the Euclidean distance between the vectors $x = (-6.94, 8.24, 0.68, 0.08)$ and $y = (-6.89, 8.31, 0.72, 0.11)$ is around 0.1 (a relatively small number with respect to the vector values) but, when $n = 50$, $AM(x)$ differs from $AM(y)$ for 23 bits, i.e., almost half the total number of bits. On the other hand, the vectors $(3.36, -6.6, -2.96, 1.1)$ and $(34.56, 27.14, 10.74, 15.26)$ are very different (their Euclidean distance is around 50), but they encode exactly the same bit-string (again, with $n = 50$).

Similar scenarios happen with the random-key approaches (see Section 2.5). Here we only consider $RK^{-1}$, though the space distortion induced by $RK^{+1}$ is even more pronounced. For example, the two vectors $x = (0.46, 0.91, 0.33, 0.75, 0.51)$ and $y = (0.4, 0.9, 0.3, 0.7, 0.5)$ correspond to the same permutation, i.e., $RK^{-1}(x) = RK^{-1}(y) = \langle 3, 1, 5, 4, 2\rangle$. The vectors $x$ and $y$ have a small Euclidean distance, but the same problem can also happens for vector pairs whose distance is arbitrarily large. As an example, let consider the family of vectors $y_\Delta = (0.4, 0.9 + \Delta, 0.3, 0.7, 0.5)$ which encode the same permutation $\langle 3, 1, 5, 4, 2\rangle$ for every choice of $\Delta \geq 0$. At the same time, given an $\epsilon > 0$, we can find, for every possible permutation $\pi \in \mathcal{S}_n$, a vector $x \in \mathbb{R}^n$ such that its Euclidean distance from the zero vector is $\epsilon$ and $RK^{-1}(x) = \pi$.

Finally, we have conducted an experiment in order to show the weak correlation between the distances on the continuous and on the permutation space. We have considered the classical distance functions: Euclidean distance for the continuous space, and Kendall's-$\tau$ distance for permutations (i.e., the number of pairwise disagreements between two permutations). Given $\sigma > 0$, 10 000 pairs of vectors $x_i, y_i \in \mathbb{R}^{10}$ have been generated such that $d(x_i, y_i) = \sigma$. The Kendall's-$\tau$ distance between $RK^{-1}(x)$ and $RK^{-1}(y)$ has been computed as well. Different values of $\sigma$ in the range $[0.1, 3]$ have been considered. The graph in Figure 1 clearly shows that, though in average both distances have a similar behavior, the variability in terms of Kendall's-$\tau$ distance explodes when the Euclidean distance $\sigma$ increases. Therefore, a large number of nearby vectors correspond to far away permutations and vice versa.

## 4. Algebraic Background

In many combinatorial optimization problems, the set of discrete solutions $X$ is naturally endowed with a composition operator, i.e., there exists a binary operator $\star$ such that, given two solutions $x, y \in X$, then $x \star y$ is again a solution. Often, $X$ and $\star$ satisfy the group properties [44]. As we will see later, this is the case of the binary and permutation representations that are quite ubiquitous in

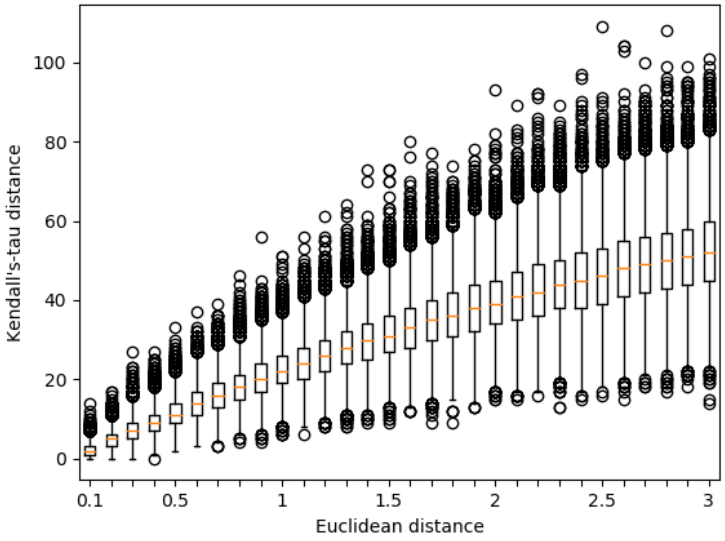



Figure 1: Boxplot chart showing the correlation between Euclidean distance and Kendall's-$\tau$ distance.

combinatorial problems: for example in the binary knapsack problem or in the permutation flowshop scheduling problem.

The algebraic structure of the seach spaces allows to characterize the geometry of the search space and to describe the search moves. Usually, the search algorithms do not explicitly exploit the algebraic properties of the search spaces. Therefore, the aim of the paper is to show how groups can be used in evolutionary algorithms, while the rest of this section is devoted to introduce the mathematical concepts used later on.

### *4.1. Groups*

A group [44] is an algebraic structure $(X, \star)$ where $X$ is a set and $\star$ is a binary operation on $X$ which fulfills the following properties:

- for all $x, y, z \in X$, $x \star (y \star z) = (x \star y) \star z$ (associativity);
- there exists a unique element $e \in X$ such that, for all $x \in X$, $x \star e =$

$e \star x = x$ (neutral or identity element);

- for every $x \in X$, there exists a unique element $x^{-1} \in X$ such that $x \star x^{-1} = x^{-1} \star x = e$ (inverse element).

If $\star$ is commutative, i.e., for all $x, y \in X$, $x \star y = y \star x$, then the group is called Abelian.

A group $(X, \star)$ is finitely generated if there exists a finite subset $H \subseteq X$ such that any $x \in X$ can be written as a composition of elements in $H$, i.e., $x = h_1 \star h_2 \star \cdots \star h_l$ for some $h_1, h_2, \ldots, h_l \in H$. $H$ and its elements are called, respectively, the generating set and the generators of $X$, while the sequence $\langle h_1, h_2, \ldots, h_l \rangle$ is a decomposition of $x$. Usually, though not strictly necessary, $H$ is closed with respect to inversion, i.e., for all $h \in H$ also $h^{-1} \in H$.

Generally, every $x \in X$ has many decompositions with possibly different lengths. Hence, a useful concept is that of minimal decomposition. A decomposition $\langle h_1, h_2, \ldots, h_l \rangle$ of a given $x \in X$ is minimal if, for any other decomposition $\langle h'_1, h'_2, \ldots, h'_m \rangle$ of $x$, we have $l \leq m$. Even minimal decompositions are not unique in general, but it is possible to define the weight $|x|$ as the length of the minimal decompositions of $x$.

Minimal decompositions allow also to define a partial order on $X$. Given $x, y \in X$, we write $x \sqsubseteq y$ if, for each minimal decomposition $s_x$ of $x$, there exists a minimal decomposition $s_y$ of $y$ such that $s_x$ is a prefix of $s_y$.

Furthermore, if $X$ is finite, there exists at least one maximal weight element. For the sake of presentation, here we focus on groups with a unique maximal weight element $\omega$ such that $x \sqsubseteq \omega$ for all $x \in X$.

### *4.2. Cayley Graphs*

Every group $(X, \star)$, finitely generated by $H$, geometrically corresponds to the Cayley graph $\mathcal{C}(X, \star, H)$, i.e., the labeled digraph whose vertexes are the elements of $X$ and there is an arc from $x$ to $y$ labeled by $h \in H$ if and only if $y = x \star h$. The graph $\mathcal{C}(X, \star, H)$ is:

- strongly connected, i.e., for all $x, y \in X$ there is a directed path from $x$ to $y$ and from $y$ to $x$;
- regular, i.e., every vertex has the same number of incoming and outgoing arcs;
- vertex-transitive, i.e., every vertex has the same set of (incoming and outgoing) arc labels.

These properties guarantee that, for any possible sequence of generators $s$ and for any element $x \in X$, $\mathcal{C}(X, \star, H)$ has exactly one path which starts from the vertex $x$ and whose arcs are labeled according to $s$. In the Cayley graph, for all $x \in X$, each directed path from the group identity $e$ to $x$ corresponds to a decomposition of $x$: if the arc 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 $x$. Hence, if $e \xrightarrow{h_1} x_1 \xrightarrow{h_2} x_2 \xrightarrow{h_3} \cdots \xrightarrow{h_l} x_l$ is a shortest path in $\mathcal{C}(X, \star, H)$, then, for any integer $i \in [1, l]$, $\langle h_1, \ldots, h_i \rangle$ is a minimal decomposition of $x_i$, and $|x_i| = i$. Moreover, given $x, y \in X$, $x \sqsubseteq y$ if and only if there exists at least one shortest path from $e$ to $y$ passing by $x$.

More generally, for all $x, y \in X$, any path from $x$ to $y$ in $\mathcal{C}(X, \star, H)$ has an algebraic interpretation. If the arc labels in the path are $\langle h_1, h_2, \ldots, h_l \rangle$, then $x \star (h_1 \star h_2 \star \cdots \star h_l) = y$. Hence, $\langle h_1, h_2, \ldots, h_l \rangle$ is a decomposition of $x^{-1} \star y$. In particular, shortest paths correspond to minimal decompositions. Starting from this observation, it is possible to define a distance $d$ on the group $X$ generated by $H$. For all $x, y \in X$, $d(x, y)$ is the length of a shortest path from $x$ to $y$ in $\mathcal{C}(X, \star, H)$ or, equivalently, $d(x, y) = |x^{-1} \star y|$. If $H$ is closed with respect to inversion, the neighborhoods of $\mathcal{C}(X, \star, H)$ are symmetric and $d$ is a metric distance. Finally, the diameter $D$ of a Cayley graph is equal to the maximal weight, i.e., $D = |\omega|$.

### *4.3. Bit-String Group*

The set $\mathbb{B}^n = \{0, 1\}^n$ of the $n$-length bit-strings forms a group with respect to the bitwise XOR, denoted by $\veebar$. The identity is the "all 0s string" $\mathbf{0}$. Moreover,

$\veebar$ is commutative and $x^{-1} = x$, for all $x \in \mathbb{B}^n$.

The most natural and elementary generating set of $\mathbb{B}^n$ is the subset of the $n$ bit-strings with exactly one 1-bit, i.e., the set $U = \{u_i \in \mathbb{B}^n : u_i(i) = 1 \text{ and } u_i(j) = 0 \text{ for } j \neq i\}$, where $u_i(k)$ is the $k$-th bit of the string $u_i$. It is important to note that $x \veebar u_i$ corresponds to flipping the $i$-th bit of $x$.

The maximal weight element of $\mathbb{B}^n$ with respect to $U$ is the "all 1s string" $\mathbf{1}$. Hence, the Cayley graph diameter is $n$.

Furthermore, the group weight is the Hamming weight, the group distance is the Hamming distance, and the Cayley graph is the binary hypercube.

### *4.4. Permutation Group*

The set $\mathcal{S}_n$ of the permutations of $[n] = \{1, 2, \ldots, n\}$ forms a group, called the *symmetric group*, with respect to the composition operator $\circ$. Given $\pi, \rho \in \mathcal{S}_n$, $\pi \circ \rho$ is defined as the permutation $(\pi \circ \rho)(i) = \pi(\rho(i))$ for all the indices $i \in [n]$. $\mathcal{S}_n$ is not Abelian and its identity is the permutation $e$ such that $e(i) = i$ for all $i \in [n]$.

Different generating sets are possible in $\mathcal{S}_n$ (see [9, 45, 46]). One of the most elementary is the subset of the $n - 1$ simple transpositions, i.e., the set $ST = \{\sigma_i \in \mathcal{S}_n : 1 \leq i < n\}$, where $\sigma_i$ is defined as: $\sigma_i(i) = i + 1$, $\sigma_i(i + 1) = i$, and $\sigma_i(j) = j$ for $j \in [n] \setminus \{i, i + 1\}$. Since the inverse of a simple transposition is itself, $ST$ is closed with respect to inversion. The maximal weight element of $\mathcal{S}_n$, with respect to $ST$, is the permutation $r$ such that $r(i) = n + 1 - i$ for all $i \in [n]$. Its weight, thus the Cayley graph diameter, is $\binom{n}{2}$.

For all $\pi \in \mathcal{S}_n$, the composition $\pi \circ \sigma_i$ corresponds to swap the adjacent items at positions $i$ and $i+1$ in $\pi$. The weight $|\pi|$ is equivalent to the number of inversions in $\pi$, i.e., the pairs of indexes $i, j \in [n]$ such that $i < j$ and $\pi(i) > \pi(j)$. Finally, the distance $d(\pi, \rho)$ between $\pi, \rho \in \mathcal{S}_n$ is known as bubble-sort distance and counts the minimum number of adjacent swaps required to transform $\pi$ in $\rho$ (or vice versa) [45].

## 5. Algebra of Combinatorial Search Spaces

In all the combinatorial optimization problems, the search space is usually described by the set $X$ of discrete solutions and the set $O$ of operators, such that any $o \in O$ can be applied to any solution $x \in X$ to obtain a (neighbor) solution $o(x) \in X$. Therefore, the neighborhood of any $x \in X$ is $\mathcal{N}(x) = \{y \in X : \exists o \in O \text{ s.t. } y = o(x)\}$. Usually, $O$ contains the most elementary variations of the solution representation adopted, e.g., bit-flips for binary strings or adjacent swaps for permutations.

It is important to note that, in many cases, the search space $(X, O)$ has an algebraic structure: it can be represented by a finitely generated group. This happens when there exist:

1. a binary operation $\star$ satisfying the group properties on $X$,
2. a finite subset $H \subseteq X$ that generates the group $(X, \star)$, and
3. a one-to-one correspondence which associates to every $o \in O$ a generator $h \in H$ such that $o(x) = x \star h$.

This algebraic structure gives rise to a Cayley graph that geometrically represents the search space and the same neighborhood relations induced by $O$. For instance, the previously described $\mathbb{B}^n$ and $\mathcal{S}_n$ are search spaces representable by finitely generated groups. Moreover, though not studied in this paper, also other search spaces satisfy the properties (1–3) like, for instance, the space of integer vectors.

In the rest of this section we show how, in a combinatorial search space represented by a group $(X, \star)$ finitely generated by $H$, it is possible to naturally introduce the operations of addition $\oplus$, subtraction $\ominus$, and scalar multiplication $\odot$ in a meaningful way and with similar properties as the analogous operations of the Euclidean vector space. This, in turn, will allow to consistently redefine the move equations of numerical EAs (like differential evolution or particle swarm optimization) for combinatorial search spaces.

The key observation is the dichotomous interpretation of an element of $X$. From Section 4, any element $x \in X$ can be decomposed and seen as a sequence

of generators, hence $x$ corresponds to a sequence of arc labels in several paths of $\mathcal{C}(X, \star, H)$. This observation is crucial, because the elements of $X$ can be seen both as *points*, i.e., vertices in the Cayley graph, and as *vectors*[1], i.e., sequences of generators in shortest paths of the Cayley graph.

### *5.1. Abstract 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 arc 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 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 with each other: $x \oplus (y \ominus x) = y$ for all $x, y \in X$.

### *5.2. Abstract Scalar Multiplication*

Again, as in the Euclidean space, it is possible to multiply a vector by a non-negative scalar in order to stretch its length.

Given $a \geq 0$ and $x \in X$, we denote their multiplication with $a \odot x$ and we first identify the conditions that $a \odot x$ has to verify in order to simulate, as much as possible, the scalar multiplication of vector spaces:

[1] Here, with "vector" we intend "free vector", i.e., a vector without point of application.

**(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 a 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 minimal decompositions. Therefore, different strategies can be devised to compute $F \odot x$. Nevertheless, since our aim is to apply the operation in evolutionary algorithms, 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 $a$. For any $x \in X$, let $\overline{a}_x = \frac{D}{|x|}$, if $a > \overline{a}_x$, (C1) would imply $|a \odot x| > D$, which is impossible. Therefore, we define

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

For the sake of clarity, we separately define the operation $a \odot x$ for the two cases $a \in [0, 1]$ and $a > 1$.

Both cases employ an abstract procedure which returns a randomly selected minimal decomposition of the element in input.

When $a \in [0, 1]$, let $l = |x|$ and consider a random shortest path from $e$ to $x$ such as $e \xrightarrow{h_1} \cdots \xrightarrow{h_k} x_k \cdots \xrightarrow{h_l} x_l$, where $x_l = x$. For the Cayley graph properties, (C1) and (C2) are satisfied by setting $a \odot x = x_k$, with $k = \lceil a \cdot l \rceil$. Moreover, when $a = 1$, $a \odot x = x$ and this satisfies both (C2) and (C3).

When $a > 1$, let $l = |x|$ and consider a random shortest path from $e$ to $\omega$ passing by $x$ such as $e \xrightarrow{h_1} \cdots \xrightarrow{h_l} x_l \xrightarrow{h_{l+1}} \cdots \xrightarrow{h_k} x_k \xrightarrow{h_{k+1}} \cdots \xrightarrow{h_D} \omega$, where $x_l = x$. For the Cayley graph properties, (C1) and (C3) are satisfied by setting $a \odot x = x_k$, with $k = \lceil a \cdot l \rceil$.

Anyway, it is possible to make the computation more efficient by exploiting that $a \odot x = (h_1 \star \cdots \star h_l) \star (h_{l+1} \star \cdots \star h_k) = x \star (h_{l+1} \star \cdots \star h_k)$, thus only the sub-path from $x$ to $\omega$, which forms a minimal decomposition of $\omega \ominus x = x^{-1} \star \omega$, needs to be known. Since $|x| + |x^{-1} \star \omega| = D$, thus $|x| = D - |x^{-1} \star \omega|$, then $a \odot x$ can be computed by taking a minimal decomposition of $x^{-1} \star \omega$, truncating it after $\lceil a \cdot |x| \rceil - |x|$ generators, and right-composing the truncated sequence with $x$.

These two abstract methods are valid for any finitely generated group. Their implementations mainly require a concrete randomized decomposition algorithm for the group at hand.

### *5.3. Vector Operations for Bit-Strings*

The bit-string representation can be used in a very large number of problems such as, for example, NK landscape optimization, binary knapsack problems, number partitioning, any subset selection problem, etc. As described in Section 4.3, for the search space of bit-strings $\mathbb{B}^n$, we consider the bitwise XOR operator $\veebar$ and the generating set $U$ representing the bit-flip moves.

Thanks to properties of this group, $\oplus$ and $\ominus$ coincide and are defined as

$$x \oplus y = x \ominus y := x \veebar y. \tag{8}$$

These operations are both commutative, and the time complexity to compute them is $\Theta(n)$.

The operation $\odot$ is implemented by considering that the search space diameter is $n$ and the maximal weight bit-string is the "all 1s string" $\mathbf{1}$. The randomized decomposition algorithm is RandBits, presented in Algorithm 1.

RandBits produces a random minimal decomposition of $x$ by returning a random permutation of the generators corresponding to the 1-bits of $x$. RandBits and $\odot$ can be computed in time $\Theta(n)$.

For the sake of clarity, we provide an illustrative example. Let consider the two bit-strings of $n = 5$ bits $x = (10101)$ and $y = (01100)$. We compute the difference $z = x \ominus y = x \veebar y = (11001)$ which, as expected, has a 1-bit in the

**Algorithm 1** Randomized decomposition algorithms for bit-strings

```
1: function RandBits(x)
2:     s ← ⟨ ⟩                                  ▷ s is a sequence of generators
3:     for i ← 1 to n do
4:         if x(i) = 1 then                     ▷ x(i) is the i-th bit of x
5:             s ← Concatenate(⟨u_i⟩, s)        ▷ u_i is a generator
6:     s ← Shuffle(s)                           ▷ Uniform random shuffle of s
7:     return s                                 ▷ s is now a min. decomposition of x
8: end function
```

positions where $x$ and $y$ differ. Adding $z$ to $y$, as expected, we get back to $x$, i.e., $y \oplus z = y \veebar z = (10101) = x$. Now, let $a = 0.66$ and let analyze the scalar multiplication $a \odot z$. First, a (random) minimal decomposition of $z$ is obtained as follows: $\mathit{RandBits}(z) = \langle u_5, u_1, u_2 \rangle$, where the $u_i$ are the generator bit-strings with exactly one 1-bit (see Section 4.3). From the decomposition it is easy to see that $|z| = 3$, thus the computation of $0.66 \odot z$ simply requires to compose the first $\lceil 0.66 \cdot 3 \rceil = 2$ generators of $z$, i.e., $0.66 \odot z = u_5 \veebar u_1 = (00001) \veebar (10000) = (10001)$. The case $a > 1$ is slightly different. Let analyze the computation of $1.33 \odot z$. First, we need the maximum-weight bit-string $\mathbf{1} = (11111)$ (see Section 4.3) and a (random) minimal decomposition of the difference $\mathbf{1} \ominus z = (00110)$, i.e., $\mathit{RandBits}(\mathbf{1} \ominus z) = \langle u_4, u_3 \rangle$. Exploiting the equivalences $D = n = 5$ and $|z| + |\mathbf{1} \ominus z| = D$, we compute, as expected, $|z| = 3$. Therefore, $1.33 \odot z$ simply requires to compose $z$ with the first $\lceil 1.33 \cdot 3 \rceil - 3 = 1$ generator of $\mathbf{1} \ominus z$, i.e., $1.33 \odot z = z \veebar u_4 = (11011)$.

### *5.4. Vector Operations for Permutations*

The permutation representation can be used in many problems such as, for example, the linear ordering problem, the permutation flowshop scheduling, the quadratic assignment problem, etc. As described in Section 4.4, for the permutation space $\mathcal{S}_n$, we consider the composition operator $\circ$ and the generating set $ST$ representing the adjacent swap moves.

The operations $\oplus$ and $\ominus$ are defined as:

$$x \oplus y := x \circ y, \tag{9}$$

$$y \ominus x := x^{-1} \circ y. \tag{10}$$

Both are non-commutative and can be computed in time $\Theta(n)$.

The operation $\odot$ is implemented by considering that the search space diameter is $\binom{n}{2}$ and the maximal weight permutation is the reversed identity $r$. The randomized decomposition algorithm is RandBS, presented in Algorithm 2.

RandBS iteratively sorts $x$ in increasing order (hence obtaining $e$) by iteratively choosing a random adjacent swap moves from the set of adjacent inversions $A$. Then, $A$ is efficiently updated by considering that the adjacent swap $\sigma_i$ can only affect three adjacent inversions $(i-1,i),(i,i+1),(i+1,i+2)$. As also highlighted in [8], RandBS and $\odot$ are computed in time $\Theta(n^2)$.

**Algorithm 2** Randomized decomposition algorithms for permutations

```
1: function RandBS(x)
2:     s ← ⟨⟩                                   ▷ s is a sequence of generators
3:     A ← {σ_i ∈ ST : i < i+1 and x(i) > x(i+1)}
4:     while A ≠ ∅ do                           ▷ A ≠∅ ⟺ x ≠ e
5:         σ ← select an element from A uniformly at random
6:         x ← x ∘ σ                            ▷ σ is a generator
7:         s ← Concatenate(⟨σ⟩, s)
8:         A ← Update(A, σ)                     ▷ Θ(1) complexity
9:     return s                                 ▷ s is now a min. decomposition of x
10: end function
```

For the sake of clarity, we provide an illustrative example. Let consider the two permutations of $n = 5$ items $x = \langle 12534 \rangle$ and $y = \langle 41532 \rangle$. In order to compute the difference $x \ominus y$, we need the inverse of $y$, i.e., $y^{-1} = \langle 25413 \rangle$. Then, $z = x \ominus y = y^{-1} \circ x = \langle 25341 \rangle$. Adding $z$ to the right of $y$, as expected, we obtain again $x$, i.e., $y \oplus z = y \circ z = \langle 12534 \rangle = x$. Now, let $a = 0.33$ and let analyze the computation of $a \odot z$. First, a (random) minimal decomposition of $z$

is obtained as follows: $\mathit{RandBS}(z) = \langle \sigma_1, \sigma_2, \sigma_4, \sigma_3, \sigma_4, \sigma_2 \rangle$, where the $\sigma_i$ are the generators defined as in Section 4.4. From the decomposition it is easy to see that $|z| = 6$, thus the computation of $0.33 \odot z$ simply requires to compose the first $\lceil 0.33 \cdot 6 \rceil = 2$ generators of $z$, i.e., $0.33 \odot z = \sigma_1 \circ \sigma_2 = \langle 21345 \rangle \circ \langle 13245 \rangle = \langle 23145 \rangle$. The case $a > 1$ is slightly different. Let analyze the computation of $1.5 \odot z$. First, we need the maximum-weight permutation $r = \langle 54321 \rangle$ (see Section 4.4) and a (random) minimal decomposition of the difference $r \ominus z = \langle 24315 \rangle$, i.e., $\mathit{RandBS}(r \ominus z) = \langle \sigma_1, \sigma_2, \sigma_3, \sigma_2 \rangle$. Exploiting the equivalences $D = \binom{5}{2} = 10$ and $|z| + |r \ominus z| = D$, we compute, as expected, $|z| = 6$. Therefore, $1.5 \odot z$ simply requires to compose $z$ with the first $\lceil 1.5 \cdot 6 \rceil - 6 = 3$ generators of $r \ominus z$, i.e., $1.5 \odot z = z \circ \sigma_1 \circ \sigma_2 \circ \sigma_3 = \langle 53421 \rangle$.

### *5.5. Algebraic Properties*

It is easy to prove that the operations $\oplus, \ominus, \odot$ satisfy the following properties:

(i) $\oplus$ is associative;

(ii) $\oplus$ is commutative, if $\star$ is commutative;

(iii) $e$ is the neutral element for $\oplus$;

(iv) $x \oplus x^{-1} = x^{-1} \oplus x = e$ for each $x \in X$;

(v) $1 \odot x = x$ for each $x \in X$;

(vi) $a \odot (b \odot x) = (ab) \odot x$ for each $x \in X$ and $a, b \geq 0$;

(vii) $0 \odot x = e$ for each $x \in X$;

(viii) $x \oplus (y \ominus x) = y$ for each $x, y \in X$.

The properties (i–viii) make $X$ similar to a vector space over $\mathbb{R}$. Besides property (ii), the two vector space's properties that do not hold in general are the distributivity laws of $\odot$ with respect to $\oplus$ and to $+$.

## 6. Algebraic Evolutionary Algorithms

In this section we describe an algebraic method to adapt an evolutionary algorithm $\mathcal{A}$, originally designed for continuous optimization, to solve a combinatorial optimization problem $\mathcal{P}$. The adaptation is possible under the following

two conditions: 1) the search space $X$ of $\mathcal{P}$ is representable by a finitely generated group, and 2) the evolutionary operators of $\mathcal{A}$ are linear combinations of population individuals.

When both conditions are met, the original evolutionary operators, expressed with the usual numerical vector operations, can be rewritten using $\oplus$, $\ominus$, and $\odot$. The adapted algorithms, called *algebraic*, directly navigate the combinatorial search space by means of the operations defined on the underlying group. Therefore, algebraic evolutionary algorithms (AEAs) simulate the search moves of their numerical counterparts in the combinatorial search space at hand.

AEAs can be defined for the most popular and effective numerical evolutionary algorithms in literature. Here, we consider the most used ones: Differential Evolution (DE) [15] and Particle Swarm Optimization (PSO) [16]. Anyway, the same technique can be applied also to other evolutionary algorithms like, for instance, Firefly Algorithm [17], Bacterial Foraging Optimization Algorithm [18], and Artificial Bee Colony [47], and simple univariate Evolution Strategies [48].

In the following we introduce the abstract forms of the algebraic variants of DE and PSO, namely, ADE and APSO. By replacing the abstract $\oplus$, $\ominus$, $\odot$ with their bit-string or permutation implementations, we have AEAs for, respectively, binary or permutation problems. The precedences among the algebraic operations are assumed to be as in their numerical counterparts.

### *6.1. Algebraic Differential Evolution*

In the Algebraic DE (ADE), previously introduced in [8] for permutation problems, the differential mutation is defined as

$$y_i \leftarrow x_{r_0} \oplus F \odot (x_{r_1} \ominus x_{r_2}), \tag{11}$$

which is the algebraic version of equation (1). The recombination between $y_i$ and $x_i$ is performed using one of the discrete crossover operators available in literature (both for bit-strings and permutations). Then, the same one-to-one selection of numerical DE decides which solution enters the next-generation population.

Finally, it is worth to note that, though we are limiting our attention to rand/1, all the known differential mutation variants (rand/2, best/1, current-to-best/1, etc.) [49] can be replaced with their algebraic counterparts.

### *6.2. Algebraic Particle Swarm Optimization*

In the Algebraic PSO (APSO), preliminary proposed in [10, 50] for permutation problems, the velocity and position update rules of the $i$-th particle are defined as:

$$v_i \leftarrow [w \odot v_i] \oplus [(c_1 \cdot r_{1i}) \odot (p_i \ominus x_i)] \oplus [(c_2 \cdot r_{2i}) \odot (g_i \ominus x_i)], \tag{12}$$

$$x_i \leftarrow x_i \oplus v_i, \tag{13}$$

which are the algebraic versions of, respectively, equations (2) and (3). The personal and social best $p_i$ and $g_i$ are then updated as in the numerical PSO.

Replacing the velocity $v_i$ in equation (13) with the right side of equation (12), we have that, differently from ADE, the solution $x_i$ is updated by adding three terms to it. Though $x_i$ can be reasonably interpreted as a point in the space, the three terms composing $v_i$ have natural interpretations as vectors. This generates an ambiguity when $\oplus$ is not commutative (for instance, in the permutations space). Hence, in these cases, the three terms of equation (12) are randomly arranged in one of the $3! = 6$ possible orders.

### *6.3. Search Characteristics of the AEAs*

Thanks to the properties of the algebraic framework from which ADE and APSO are derived, their search moves are geometrically similar with respect to the movements performed by their numerical counterparts. Anyway, the discrete nature of the combinatorial spaces inevitably introduces some differences with the numerical algorithms.

The main one is the lack of continuity. Indeed, since discrete spaces are finite (or at most countable) an individual cannot get infinitesimally closer to a given solution. Another way of looking at the same issue is that the discrete

spaces are graphs (Cayley graphs in our cases), hence the distance between two solutions is integer-valued. The practical consequence in AEAs, but also in other combinatorial algorithms, is that loss of population diversity is more drastic than in continuous space.

Another difference with numerical algorithms is that when the underlying group is not Abelian, the $\oplus$ operation is not commutative. This issue has been considered in APSO for the permutations space (see Section 6.2).

Furthermore, in the case of bit-strings, both $\oplus$ and $\ominus$ are actually the bit-wise XOR operator, hence $x \oplus x = x \ominus x = e$ for all $x \in \mathbb{B}^n$. This implies that, when the population reaches consensus on a bit (i.e., when all individuals have their $i$-th bit set to the same value), it is impossible to flip that bit in the future generations by using movement equations that only consider linear combinations of population individuals and multiplication by scalars not larger than 1. Anyway, note that this aspect is counter-balanced by the fact that the binary underlying group is Abelian. Indeed, when the composition is commutative, the number of possible minimal decompositions of a generic group element is intuitively large. Just look at RandBits of Algorithm 1 and observe that any possible permutation of the 1-bits of the input string corresponds to a minimal decomposition sequence. This implies that there are intuitively many ways of truncating a decomposition sequence, that in turn likely slows down the convergence to population consensus on a bit.

## 7. Experimental Analysis

The proposed algebraic EAs (AEAs) have been experimentally analyzed on both binary and permutation problems. The benchmarks and the experimental setup adopted are described in, respectively, Sections 7.1 and 7.2. Algorithm parameters are tuned as reported in Section 7.3, while different experimental scenarios have been considered. A first set of experiments is described in Section 7.4, where standalone versions of the AEAs are compared with the standalone decoder-based numerical EAs described in Sections 2.3, 2.4, and 2.5. Then,

Section 7.5 describes a second set of experiments designed to compare enhanced versions of both classes of algorithms. Lastly, in Section 7.6, the best AEAs results are compared to the state-of-the-art results in literature for the tackled problems.

### *7.1. Benchmarks*

Benchmark problems and instances have been selected for both the binary and permutation cases. Due to their generality, NK Landscapes (NKL) [19] have been considered as binary benchmarks, while, for permutation problems, the experiments have been held on the Permutation Flowshop Scheduling Problem (PFSP) with the total flowtime as objective function [20] and on the Linear Ordering Problem (LOP) [21].

An NKL instance of $n$ bits and epistasis $K$ is provided as $n$ tabulated sub-functions $f_i : \mathbb{B}^{K+1} \to \mathbb{R}$, with $i \in \{1, \ldots, n\}$, each one defined on $K+1$ bits of an $n$-length bit-string including its $i$-th bit. The objective is to maximize the sum of the sub-function values.

A PFSP instance consists of $n$ jobs, $m$ machines, and a processing time $p_{i,j}$ for every job $i \in \{1, \ldots, n\}$ in every machine $j \in \{1, \ldots, m\}$. Given a permutation of jobs $\pi \in \mathcal{S}_n$, the completion time of job $\pi(i)$ in machine $j$ is recursively computed as $c_{\pi(i),j} = p_{\pi(i),j} + \max\{c_{\pi(i-1),j}, c_{\pi(i),j-1}\}$ by also considering the terminal cases $c_{i,0} = c_{0,j} = 0$. The total flowtime objective requires to minimize $f(\pi) = \sum_{i=1}^{n} c_{\pi(i),m}$.

A LOP instance is provided as an $n \times n$ matrix $H$ and the objective is to find a permutation $\pi \in \mathcal{S}_n$ which maximizes $f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} H_{\pi(i),\pi(j)}$.

Sixty instances for each one of the three problems have been selected from widely adopted benchmark suites as follows.

- NKL instances come from the combinatorial black-box optimization contest organized at GECCO 2015[2], where they have been randomly generated using different dimensionalities and epistasis values.

[2] http://web.mst.edu/~tauritzd/CBBOC/GECCO2015.

- PFSP instances are taken from the Taillard benchmark suite[3] and, as explained in [51], they are the most difficult instances (basing on few algorithms chosen by the author) among a randomly generated set of instances.
- LOP instances are from the benchmark sets LOLIB, SGB and MB[4]: LOLIB is a real-world dataset of input-output tables used in economics, while SGB and MB are randomly generated instances with different dimensionalities and levels of sparsity.

Finally, note that the subset of $60 \times 3 = 180$ instances used in our work has been selected from these datasets by uniformly covering their inner characteristics. For the sake of space, the names of the selected instances are provided as supplementary material on the web [52].

### *7.2. Setup of the experiments*

ADE and APSO have, each one, two implementations: for the binary and permutation space, thus we have a total of four algebraic algorithms. Both in the standalone and enhanced comparison scenarios (see Sections 7.4 and 7.5), ADE and APSO have been compared with the decoder-based numerical EAs described in Sections 2.3, 2.4 and 2.5. Therefore, six binary algorithms have been compared on the NKL instances, while four algorithms for permutation problems have been analyzed on PFSP and LOP instances. We use the same acronyms defined in Section 2 and, for the sake of presentation, we report them in Table 1.

All the PSO-based schemes adopt the ring topology. With regards to the DE-based algorithms, the standard binomial crossover of classical DE is used without any modification on the binary problems, while the OBX crossover [9] is adopted for the permutation problems. Indeed, OBX can be regarded as a simple feasible variant of the binomial crossover for the permutation representation.

[3] http://mistic.heig-vd.ch/taillard.
[4] http://www.optsicom.es/lolib.

Table 1: Acronyms of the algorithms compared in the experimentation

| **Search space** | **Problems** | **Algorithms Acronyms** | **Algorithms Descriptions** |
|---|---|---|---|
| Binary | NKL | ADE | Algebraic DE for the binary space |
| | | APSO | Algebraic PSO for the binary space |
| | | BDE [32] | Binary DE (see Section 2.3) |
| | | BPSO [7] | Binary PSO (see Section 2.3) |
| | | AM-DE [2] | Angle Modulated DE (see Section 2.4) |
| | | AM-PSO [37] | Angle Modulated PSO (see Section 2.4) |
| Permutation | PFSP | ADE | Algebraic DE for the permutation space |
| | LOP | APSO | Algebraic PSO for the permutation space |
| | | RK-DE [31] | Random-key DE (see Section 2.5) |
| | | RK-PSO [3] | Random-key PSO (see Section 2.5) |

For every instance, all the algorithms have been executed 20 times using, as termination criteria, both a given budget of fitness evaluations and of computational time. The final fitness values produced by the executions of every algorithm have been aggregated for each instance using the Average Relative Percentage Deviation (ARPD) measure, which is computed according to

$$\mathrm{ARPD}_{\mathrm{Inst}}^{\mathrm{Alg}} = \frac{1}{20}\sum_{i=1}^{20}\frac{\left|\mathrm{Alg}_{\mathrm{Inst}}^{(i)} - \mathrm{Best}_{\mathrm{Inst}}\right|}{\mathrm{Best}_{\mathrm{Inst}}} \times 100, \tag{14}$$

where $\mathrm{Alg}_{\mathrm{Inst}}^{(i)}$ is the final fitness value produced by the algorithm Alg in its $i$-th run on the instance Inst, while $\mathrm{Best}_{\mathrm{Inst}}$ is the best result obtained on the given instance by any algorithm in any run of the considered experiment. The ARPDs have to be minimized both for maximization and minimization problems. Statistical analyses have been conducted by means of the well known Wilcoxon signed-rank test [53].

Finally, a state-of-the-art comparison has been carried out by comparing the best objective values obtained by the algebraic algorithms with the best known solutions so far of every benchmark instance (see Section 7.6).

### 7.3. Tuning of the parameters

Due to the different characteristics of the search spaces navigated by the algebraic and decoder-based algorithms, in order to perform a fair comparison, the parameters of the algorithms have been separately tuned using SMAC [54], i.e., a popular tool for automatic algorithm configuration based on statistical and machine learning techniques. To avoid the over-tuning phenomenon [55], SMAC calibrations have been run using a separate set of instances with respect to those used for algorithm comparisons. The list of the 30 selected calibration instances for each problem class is provided as supplementary material on the web [52]. Every SMAC calibration has been set to run for 72 hours of computational time, while every algorithm execution terminates after $100n^2$ fitness evaluations have been performed. The ranges of the parameters in input to SMAC are as follows: $N \in \{20, 60, 100\}$; $F, CR \in [0, 1]$ for DE schemes; $w \in [0, 1]$ and $c_1, c_2 \in [0, 2]$ for PSO schemes; $b \in [20, 100]$ for BDE; $k \in \{-1, 1\}$ for the random-key schemes. The tuned parameter settings resulting from the SMAC executions are provided in Table 2.

As expected, the different morphologies of the search landscapes are reflected on the noticeable differences among the calibrated parameter values.

### 7.4. Standalone Algorithms Comparison

The aim of this section is to verify if our proposals are competitive with respect to the decoder-based variants of DE and PSO. Therefore, in this set of experiments all the algorithms have been implemented in their basic versions as depicted by Sections 2 and 6.

The algorithms comparison has been performed by using the calibrated settings of Table 2 and considering two termination criteria: $100n^2$ fitness evaluations (as in calibration), and $50n^2$ milliseconds of computational time. The second criterion is motivated by the inherently different computational complexities of the algebraic and decoder-based algorithms.

The results of the executions are presented in aggregated form in Tables 3, 4, and 5, respectively, for NKL, PFSP, and LOP instances. For each algorithm

Table 2: Calibred settings obtained with SMAC

| Problem | Algorithm | Parameter Settings |
|---|---|---|
| NKL | ADE | $N = 100, F = 0.22, CR = 0.22$ |
| | BDE [32] | $N = 100, F = 0.78, CR = 0.65, b = 98.77$ |
| | AM-DE [2] | $N = 60, F = 0.12, CR = 0.32$ |
| | APSO | $N = 100, w = 0.04, c_1 = 0.58, c_2 = 1.7$ |
| | BPSO [7] | $N = 20, w = 0.79, c_1 = 1.64, c_2 = 1.33$ |
| | AM-PSO [37] | $N = 20, w = 0.73, c_1 = 1.77, c_2 = 1.44$ |
| PFSP | ADE | $N = 100, F = 0.13, CR = 0.65$ |
| | RK-DE [31] | $N = 100, F = 0.4, CR = 0.95, k = 1$ |
| | APSO | $N = 20, w = 4 \cdot 10^{-5}, c_1 = 1.4, c_2 = 1.05$ |
| | RK-PSO [3] | $N = 100, w = 0.8, c_1 = 1.78, c_2 = 1.76, k = -1$ |
| LOP | ADE | $N = 100, F = 0.05, CR = 0.42$ |
| | RK-DE [31] | $N = 60, F = 0.9, CR = 0.95, k = 1$ |
| | APSO | $N = 60, w = 0.003, c_1 = 1.74, c_2 = 1.09$ |
| | RK-PSO [3] | $N = 100, w = 0.8, c_1 = 1.73, c_2 = 1.64, k = -1$ |

and for both termination criteria, any of such tables provides four performance measures: 1) the overall ARPD, 2) the non-parametric average rank (of the algorithm's ARPDs), 3) the intra-class success rate defined as the number of instances where the algorithm obtained the best ARPD compared to the same-class competitors[5], and 4) the p-value of the Wilcoxon test with respect to the best performing competitor in the same algorithmic class.

The results clearly show that the algebraic algorithms outperform the decoder-based approaches with very high statistical evidence in every problem and considering both termination criteria. In general, the algebraic DE is the best scheme in every problem. Indeed, ADE obtained 1 as average rank both for PFSP and LOP experiments, meaning that ADE has obtained the best ARPD

[5]This is not the standard definition of success rate, i.e., the frequency with which the best known solution has been obtained in $n$ trials. Note anyway that, since our intra-class success rate consider the ARPD values, it takes implicitly into account also the multiple trials performed by the algorithm.

Table 3: Experimental results of standalone algorithms on NKL

| **Termination Criterion** | **Performance Measure** | **ADE** | **BDE** [32] | **AM-DE** [2] | **APSO** | **BPSO** [7] | **AM-PSO** [37] |
|---|---|---|---|---|---|---|---|
| | Overall ARPD | **0.29** | 12.32 | 12.83 | 0.79 | 6.85 | 12.31 |
| Evaluations | Avg Rank | **1.25** | 4.59 | 5.88 | 1.84 | 2.91 | 4.53 |
| Budget | Success Rate | **60/60** | 0/60 | 0/60 | **56/60** | 4/60 | 0/60 |
| | p-value | **best** | $< 10^{-10}$ | $< 10^{-10}$ | **best** | $< 10^{-10}$ | $< 10^{-10}$ |
| | Overall ARPD | **0.25** | 12.40 | 12.90 | 0.43 | 6.93 | 12.39 |
| Time | Avg Rank | **1.39** | 4.58 | 5.88 | 1.62 | 2.99 | 4.54 |
| Budget | Success Rate | **60/60** | 0/60 | 0/60 | **60/60** | 0/60 | 0/60 |
| | p-value | **best** | $< 10^{-10}$ | $< 10^{-10}$ | **best** | $< 10^{-10}$ | $< 10^{-10}$ |

Table 4: Experimental results of standalone algorithms on PFSP

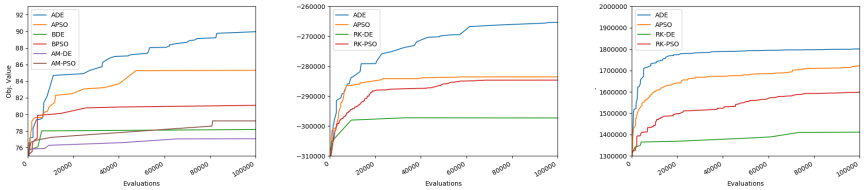
| **Termination Criterion** | **Performance Measure** | **ADE** | **RK-DE** [31] | **APSO** | **RK-PSO** [3] |
|---|---|---|---|---|---|
| | Overall ARPD | **0.62** | 13.72 | 5.78 | 6.58 |
| Evaluations | Avg Rank | **1.00** | 4.00 | 2.10 | 2.90 |
| Budget | Success Rate | **60/60** | 0/60 | **54/60** | 6/60 |
| | p-value | **best** | $< 10^{-10}$ | **best** | $< 10^{-8}$ |
| | Overall ARPD | **0.38** | 11.72 | 4.59 | 4.83 |
| Time | Avg Rank | **1.00** | 4.00 | 2.31 | 2.69 |
| Budget | Success Rate | **60/60** | 0/60 | **42/60** | 18/60 |
| | p-value | **best** | $< 10^{-10}$ | **best** | 0.006 |

Table 5: Experimental results of standalone algorithms on LOP

| **Termination Condition** | **Performance Measure** | **ADE** | **RK-DE** [31] | **APSO** | **RK-PSO** [3] |
|---|---|---|---|---|---|
| | Overall ARPD | **0.14** | 19.40 | 3.52 | 4.11 |
| Evaluations | Avg Rank | **1.00** | 4.00 | 2.17 | 2.83 |
| Budget | Success Rate | **60/60** | 0/60 | **50/60** | 10/60 |
| | p-value | **best** | $< 10^{-10}$ | **best** | $< 10^{-9}$ |
| | Overall ARPD | **0.08** | 16.93 | 2.95 | 3.12 |
| Time | Avg Rank | **1.00** | 4.00 | 2.28 | 2.72 |
| Budget | Success Rate | **60/60** | 0/60 | **42/60** | 18/60 |
| | p-value | **best** | $< 10^{-10}$ | **best** | $< 10^{-3}$ |

on every single PFSP and LOP instance. Moreover, in the binary NKL instances, though ADE is anyway the best algorithm in average, APSO reached quite comparable results.

In order to analyze the convergence behaviors, Figure 2 presents the convergence graphs obtained by the median execution of every algorithm in three benchmark instances, one per problem. The data provided are typical data which have been observed also in other executions and benchmark instances. The graphs show behavior of the objective value – of the best solution in the population – with respect to to the number of evaluations performed so far. Since the PFSP is the only minimization problem, for the sake of homogeneity, we present the opposite objective values.




(a) NKL (p2/00000.txt) (b) PFSP (tai_100_5_0) (c) LOP (IO/N-usa79)

Figure 2: Convergence graphs in three selected instances of the algorithms ADE, APSO, BDE [32], BPSO [7], AM-DE [2], AM-PSO [37], RK-DE [31], RK-PSO [3].

The most interesting observation which follows from these graphs is that, in general, most of the decoder-based schemes seem to converge earlier than algebraic algorithms. One only exception looks to be RK-PSO, in particular in the LOP benchmark. This premature convergence looks to be a plausible explanation of the different performances among the schemes.

Finally, we refer the interested reader to the supplementary material [52] for the full details of the experimental results.

### *7.5. Enhanced Algorithms Comparison*

In order to provide a more complete analysis, additional experiments have been held with the aim of comparing enhanced versions of the AEAs and

decoder-based schemes. The algorithms from both classes have been equipped with four additional components commonly employed in literature to improve the performance of an evolutionary algorithm: a heuristic initialization, a restart mechanism, a local search procedure, and a self-adaptive strategy for the algorithms' parameters.

The heuristic initialization works by generating one initial solution by means of a heuristic procedure purposely defined for the problem at hand, while the remaining $N-1$ individuals are randomly generated. For PFSP and LOP, we have adopted very popular and effective constructive procedures: respectively, the Liu-Reeves [56] and best-insertion [57] heuristics. For NK landscapes, as far as we know, there is no heuristic available in literature, thus we have designed a simple procedure: randomly generate $n(K+1)$ bit-strings and select the best one. Though being very simple, we have experimentally observed that it guarantees a good starting point for the algorithms.

Restarts are triggered when the best population solution has not been improved during the last $T$ generations. After some preliminary experiments, $T$ has been set to 1000. In the DE-based schemes, as done for example in [8], the restart randomizes all the solutions except the best one, while in the PSO-based schemes we have adopted the restart strategy described in [58], i.e., all the personal best solutions are randomized except for the global best particle that randomizes its velocity.

As suggested in [59], in every generation, local search is applied to every solution with probability $1/N$. Hence, in average, one solution per generation undergoes local search. Additionally, in order to refine the search as much as possible, we apply the local search also to the best solution after every restart and at the end of the evolution. A complete best-improvement local search procedure has been employed. The bit-flip neighborhood has been used in NKL instances, while for PFSP and LOP, the commonly used neighborhood based on item's insertion moves has been considered [45].

In order to automatically calibrate the parameters during the evolution we have used two popular self-adaptive mechanisms: the jDE scheme for the DE-

based algorithms [60], and the self-adaptive PSO scheme introduced in [61]. Both schemes extend each population individual with a personal copy of the parameters which are evolved similarly to the genotype solution. Therefore, no parameter setting is required except for the population size $N$ and the random-key parameter $k$ that have been set as in Table 2.

The enhanced algorithms are denoted as: ADE$^+$, APSO$^+$, BDE$^+$, BPSO$^+$, AM-DE$^+$, AM-PSO$^+$, RK-DE$^+$, and RK-PSO$^+$. Note also that, both in the heuristic initialization and after any application of the local search, the random-key and the angle modulated schemes require to encode the obtained discrete solution to a numerical vector. For random-key schemes, this is done by normalizing the permutation to a numerical vector such that any dimension is within the allowed numerical bounds. However, as far as we know, there is no known method to invert the AM decoder of equation (4), therefore, AM-DE$^+$ and AM-PSO$^+$ actually do not use heuristic initialization and adopt a Baldwinian local search, i.e., the (hopefully) improved solution is recorded but does not enter the population. Moreover, BPSO$^+$ has been further improved by setting $v_{\max} = \ln(n-1)$, as suggested in [62].

Experiments have been held using the set of selected instances, every algorithm has been executed 20 times per instance, and each execution terminates after 10 consecutive restarts without improvement of the best solution or after 10 minutes of computational time. The aggregated experimental results are presented in Tables 6, 7, and 8 for, respectively, the NKL, PFSP and LOP benchmarks. The same layout described in Section 7.4 is used.

Table 6: Experimental results of enhanced algorithms on NKL

| **Performance Measure** | **ADE$^+$** | **BDE$^+$** [32] | **AM-DE$^+$** [2] | **APSO$^+$** | **BPSO$^+$** [7] | **AM-PSO$^+$** [37] |
|---|---|---|---|---|---|---|
| Overall ARPD | **0.16** | 12.85 | 13.68 | 0.25 | 1.29 | 4.15 |
| Avg Rank | **1.32** | 6.00 | 4.00 | 1.78 | 5.00 | 2.90 |
| Success Rate | **60/60** | 0/60 | 0/60 | **60/60** | 3/60 | 0/60 |
| p-value | **best** | $< 10^{-10}$ | $< 10^{-10}$ | **best** | $< 10^{-10}$ | $< 10^{-10}$ |

Table 7: Experimental results of enhanced algorithms on PFSP

| **Performance Measure** | **ADE$^+$** | **RK-DE$^+$** [31] | **APSO$^+$** | **RK-PSO$^+$** [3] |
|---|---|---|---|---|
| Overall ARPD | **0.42** | 1.06 | 0.59 | 0.92 |
| Avg Rank | **1.48** | 3.13 | 2.45 | 2.93 |
| Success Rate | **49/60** | 11/60 | **41/60** | 19/60 |
| p-value | **best** | $< 10^{-7}$ | **best** | $< 10^{-5}$ |

Table 8: Experimental results of enhanced algorithms on LOP

| **Performance Measure** | **ADE$^+$** | **RK-DE$^+$** [31] | **APSO$^+$** | **RK-PSO$^+$** [3] |
|---|---|---|---|---|
| Overall ARPD | **0.01** | 0.29 | 0.20 | 0.24 |
| Avg Rank | **1.22** | 3.10 | 2.58 | 3.10 |
| Success Rate | **57/60** | 4/60 | **41/60** | 20/60 |
| p-value | **best** | $< 10^{-9}$ | **best** | 0.018 |

The results clearly show that the AEAs outperform the decoder-based approaches in every problem also when both classes of algorithms are enhanced with additional algorithmic components. Indeed, as in the previous experiments, the reported p-values largely indicate that all the comparisons have very high significance, thus definitely validating our proposals with respect to the decoder-based competitors.

In general, all algorithms, apart few exceptions (BDE$^+$, AM-DE$^+$, AM-PSO$^+$ in NKL instances), decrease their overall ARPDs with respect to the experiments of Section 7.4. This is a consequence of the additional algorithmic components which tend to reduce performance differences among multiple runs of the same algorithms. In a few instances, the decoder-based approaches have been able to reach the ARPDs of the algebraic algorithms. This is shown by the non-zero success rates obtained by BPSO$^+$ in the NKL instances, and by RK-DE$^+$ and RK-PSO$^+$ in both the PFSP and LOP instances. However, also the AEAs have consistently improved their overall ARPDs: the best one is 0.01%

(ADE$^+$ in the LOP problem), while the worst one is only 0.59% (APSO$^+$ in the PFSP problem). Finally, it is worth to note that, as in the previous set of experiments, ADE$^+$ is clearly the best algorithm, though APSO$^+$ reaches a quite comparable average rank on NKL instances.

Finally, we refer the interested reader to the supplementary material [52] for the full details of the experimental results.

### *7.6. Comparison with State-of-the-art Results*

Though the algebraic algorithms clearly outperform the decoder-based approaches, the previous experiments say nothing about the general performances of AEAs with respect to pure combinatorial algorithms. Therefore, in this section we compare the AEAs performances with respect to the state-of-the-art results on the tackled problems.

The analysis has been conducted by comparing, on every instance, the best objective function values obtained by both ADE$^+$ and APSO$^+$ with respect to the best known solutions available in literature. In particular, these latter have been obtained from: the CBBOC website for NKL (see footnote 2), the LOLIB website for LOP (see footnote 4), and from [8, Table III] for PFSP. With this regard, it is important to stress out that the best known solutions have been obtained by a variety of different algorithms, often purposely designed for the problem at hand. We think that this comparison is much more explanatory and comprehensive than comparing the AEAs with a bunch of chosen algorithms.

Both for ADE$^+$ and APSO$^+$, the analysis is summarized by the indices provided in Table 9 where, aggregated over the instances of every problem, it is reported: the average, maximum and minimum percentage improvement with respect to the best known solution (BKS) together with the number of BKSs that have been matched or improved by our proposals.

In the NK landscapes, all the best known solutions have been improved (46 cases), or at least reached (14 cases), by both ADE$^+$ and APSO$^+$. Though these instances have been only investigated during the CBBOC competition, the results are anyway interesting since they clearly show that algebraic algorithms

Table 9: Comparison with state-of-the-art results

| Performance Measure | NKL Instances | | PFSP Instances | | LOP Instances | |
|---|---|---|---|---|---|---|
| | **ADE$^+$** | **APSO$^+$** | **ADE$^+$** | **APSO$^+$** | **ADE$^+$** | **APSO$^+$** |
| Avg Improvement | +2.06% | +2.05% | −0.60% | −0.93% | −0.0005% | −0.02% |
| Max Improvement | +10.96% | +10.85% | 0.00% | 0.00% | 0.00% | 0.00% |
| Min Improvement | 0.00% | 0.00% | −2.54% | −2.85% | −0.03% | −0.19% |
| BKSs reached | 60/60 | 60/60 | 20/60 | 20/60 | 58/60 | 31/60 |
| BKSs improved | 46/60 | 46/60 | 0/60 | 0/60 | 0/60 | 0/60 |

are competitive on binary problems.

In the selected LOP instances, it is worth to note that the best known solutions have been proven to be optimal [21]. Hence, Table 9 shows that ADE$^+$ has been able to reach the optimum in almost the totality of the instances while its average deviation from the optimum, though negative, is almost negligible. Moreover, the worst deviation obtained by APSO$^+$ is only 0.19%.

For PFSP, though both of our proposals have matched the state-of-the-art results in the 20 smaller instances, the average deviations from the best known solutions are slightly larger than in LOP, but they are anyway not greater than 1%.

In conclusion, this analysis shows that the algebraic framework here proposed is quite general and effective also from a practical point of view.

## 8. Conclusion and Future Work

The main contribution of this work is the general algebraic framework by which it is possible to adapt a large class of numerical evolutionary algorithms to tackle an important class of combinatorial optimization problems.

The framework has been abstractly described by means of the algebraic properties of finitely generated groups, while concrete implementations are provided for binary and permutation search spaces. Discrete vector operations have been proposed in such a way that their geometric interpretations are consistent

throughout the different spaces and with respect to their numerical counterparts.

The algebraic variants of two popular algorithms have been proposed: Algebraic Differential Evolution (ADE) and Algebraic Particle Swarm Optimization (APSO).

These algorithms have been implemented and compared with six algorithms in literature which use the decoder-based schemes, i.e., classical numerical algorithms endowed with decoding procedures that convert continuous individuals to discrete solutions. To the best of our knowledge, this is the only technique in literature that guarantees a level of applicability comparable to our framework.

Experiments have been held both on binary and permutation problems by considering two scenarios: standalone and enhanced implementations of the algorithms. The experimental results show that the algebraic algorithms clearly outperform the competitors in both scenarios. Importantly, a further comparison with state-of-the-art results in literature shows that our proposals are also competitive with pure combinatorial algorithms. These empirical results, together with the strong mathematical foundations over which our proposal is built, allow us to suggest our framework in the design of new algorithms for combinatorial optimization problems.

A future research of direction is to apply the proposed framework to other search spaces such as, for instance, the space of integer vectors. Moreover, since the Cartesian product of search spaces representable as groups form itself a group (the so-called product group), we are also planning to expand the applications of the framework to more complex search spaces that are usually tackled by means of co-evolutionary algorithms. Another research direction is to propose algebraic variants for other numerical algorithms like, for example, Firefly Algorithm or Bacterial Foraging Optimization scheme. Finally, it is interesting to study the potentialities of the framework in order to provide a new perspective in the theoretical analysis of combinatorial meta-heuristics. Preliminary investigations in some of these directions are presented in [63, 64].

## References


[1] S. Mirjalili, A. Lewis, S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization, Swarm and Evolutionary Computation 9 (2013) 1–14.

[2] G. Pampara, A. P. Engelbrecht, N. Franken, Binary Differential Evolution, in: Proc. of 2006 IEEE International Conference on Evolutionary Computation (CEC 2006), 2006, pp. 1873–1879.

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

[4] M. K. Marichelvam, T. Prabaharan, X. S. Yang, A Discrete Firefly Algorithm for the Multi-Objective Hybrid Flowshop Scheduling Problems, IEEE Transactions on Evolutionary Computation 18 (2) (2014) 301–305.

[5] M. Ayodele, J. A. W. McCall, O. Regnier-Coudert, RK-EDA: A Novel Random Key Based Estimation of Distribution Algorithm, in: Proc. of 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV), 2016, pp. 849–858.

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

[7] J. Kennedy, R. C. Eberhart, A discrete binary version of the particle swarm algorithm, in: Proc. of 1997 IEEE International Conference on Systems, Man, and Cybernetics (SMC 1997), Vol. 5, 1997, pp. 4104–4108.

[8] V. Santucci, M. Baioletti, A. Milani, Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem With Total Flowtime Criterion, IEEE Transactions on Evolutionary Computation 20 (5) (2016) 682–694. doi:10.1109/TEVC.2015.2507785.

[9] M. Baioletti, A. Milani, 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 (PPSN XIV), 2016, pp. 123–133. doi:10.1007/978-3-319-45823-6_12.

[10] M. Baioletti, A. Milani, V. Santucci, Algebraic particle swarm optimization for the permutations search space, in: Proc. of 2017 IEEE Congress on Evolutionary Computation (CEC 2017), 2017, pp. 1587–1594. doi:10.1109/CEC.2017.7969492.

[11] M. Baioletti, A. Milani, V. Santucci, MOEA/DEP: An Algebraic Decomposition-Based Evolutionary Algorithm for the Multiobjective Permutation Flowshop Scheduling Problem, in: Evolutionary Computation in Combinatorial Optimization, 2018, pp. 132–145. doi:10.1007/978-3-319-77449-7_9.

[12] V. Santucci, M. Baioletti, G. Di Bari, A. Milani, A Binary Algebraic Differential Evolution for the MultiDimensional Two-Way Number Partitioning Problem, in: Evolutionary Computation in Combinatorial Optimization, 2019, pp. 17–32. doi:10.1007/978-3-030-16711-0_2.

[13] M. Baioletti, A. Milani, V. Santucci, Variable neighborhood algebraic Differential Evolution: An application to the Linear Ordering Problem with Cumulative Costs, Information Sciences 507 (2020) 37–52. doi:10.1016/j.ins.2019.08.016.

[14] H. H. Hoos, T. Stützle, Stochastic local search: Foundations & applications, Elsevier, 2004.

[15] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[16] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. of IEEE International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948.

[17] X.-S. Yang, Firefly algorithms for multimodal optimization, in: Proc. of 5th International Symposium on Stochastic Algorithms, 2009, pp. 169–178.

[18] S. Das, A. Biswas, S. Dasgupta, A. Abraham, Bacterial foraging optimization algorithm: theoretical foundations, analysis, and applications, Foundations of Computational Intelligence 3 (2009) 23–55.

[19] S. A. Kauffman, E. D. Weinberger, The NK model of rugged fitness landscapes and its application to maturation of the immune response, Journal of Theoretical Biology 141 (2) (1989) 211–245.

[20] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2) (2005) 479–494.

[21] T. Schiavinotto, T. Stützle, The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms, Journal of Mathematical Modelling and Algorithms 3 (4) (2004) 367–402.

[22] A. Milani, V. Santucci, Asynchronous differential evolution, in: Proc. of 2010 IEEE Congress on Evolutionary Computation (CEC 2010), 2010, pp. 1–7. doi:10.1109/CEC.2010.5586107.

[23] Q. Guo, L. Tang, Modelling and discrete differential evolution algorithm for order rescheduling problem in steel industry, Computers & Industrial Engineering 130 (2019) 586 – 596.

[24] G. Zhang, K. Xing, F. Cao, Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion, Engineering Applications of Artificial Intelligence 76 (2018) 96–107.

[25] J. Bock, J. Hettenhausen, Discrete particle swarm optimisation for ontology alignment, Information Sciences 192 (2012) 152–173.

[26] M. F. Tasgetiren, P. Suganthan, Q.-K. Pan, An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem, Applied Mathematics and Computation 215 (9) (2010) 3356–3368.

[27] W. Chen, J. Zhang, H. S. Chung, W. Zhong, W. Wu, Y. Shi, A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems, IEEE Transactions Evolutionary Computation 14 (2) (2010) 278–300.

[28] Y. Liu, W. Chen, Z. Zhan, Y. Lin, Y. Gong, J. Zhang, A Set-Based Discrete Differential Evolution Algorithm, in: Proc. of 2013 IEEE International Conference on Systems, Man, and Cybernetics, Manchester (SMC 2013), 2013, pp. 1347–1352.

[29] M. Clerc, Discrete particle swarm optimization, illustrated by the traveling sales-man problem, in: New Optimization Techniques in Engineering, Studies in Fuzziness and Soft Computing, 2004, pp. 219–239.

[30] A. Moraglio, J. Togelius, S. Silva, Geometric Differential Evolution for Combinatorial and Programs Spaces, Evolutionary Computation 21 (4) (2013) 591–624.

[31] X. Li, M. Yin, An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure, Advances in Engineering Software 55 (2013) 10–31.

[32] L. Wang, X. Fu, Y. Mao, M. I. Menhas, M. Fei, A novel modified binary differential evolution algorithm and its applications, Neurocomputing 98 (2012) 55–75.

[33] M. F. Taşgetiren, Y.-C. Liang, A binary particle swarm optimization algorithm for lot sizing problem, Journal of Economic and Social Research 5 (2) (2003) 1–20.

[34] S. Pookpunt, W. Ongsakul, Optimal placement of wind turbines within wind farm using binary particle swarm optimization with time-varying acceleration coefficients, Renewable Energy 55 (2013) 266–276.

[35] L.-Y. Chuang, C.-H. Yang, J.-C. Li, Chaotic maps based on binary particle swarm optimization for feature selection, Applied Soft Computing 11 (1) (2011) 239–248.

[36] I. Babaoglu, O. Findik, E. Ulker, A comparison of feature selection models utilizing binary particle swarm optimization and genetic algorithm in determining coronary artery disease using support vector machine, Expert Systems with Applications 37 (4) (2010) 3177 – 3183.

[37] G. Pampara, N. Franken, A. P. Engelbrecht, Combining particle swarm optimisation with angle modulation to solve binary problems, in: Proc. of 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Vol. 1, 2005, pp. 89–96.

[38] L. Liu, W. Liu, D. A. Cartes, I.-Y. Chung, Slow coherency and angle modulated particle swarm optimization based islanding of large-scale power systems, Advanced Engineering Informatics 23 (1) (2009) 45 – 56.

[39] B. J. Leonard, A. P. Engelbrecht, Angle Modulated Particle Swarm Variants, in: Proc. of 9th International Conference on Swarm Intelligence (ANTS 2014), 2014, pp. 38–49.

[40] H. Gao, S. Kwong, B. Fan, R. Wang, A hybrid particle-swarm tabu search algorithm for solving job shop scheduling problems, IEEE Transactions on Industrial Informatics 10 (4) (2014) 2044–2054.

[41] E. Cao, M. Lai, H. Yang, Open vehicle routing problem with demand uncertainty and its robust strategies, Expert Systems with Applications 41 (7) (2014) 3569–3575.

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

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

[44] S. Lang, Algebra, Vol. 211, Springer, 2002.

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

[46] J. Scharnow, K. Tinnefeld, I. Wegener, The analysis of evolutionary algorithms on sorting and shortest paths problems, Journal of Mathematical Modelling and Algorithms 3 (4) (2005) 349–366.

[47] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied mathematics and computation 214 (1) (2009) 108–132.

[48] H.-G. Beyer, H.-P. Schwefel, Evolution strategies – A comprehensive introduction, Natural Computing 1 (1) (2002) 3–52.

[49] K. Price, R. M. Storn, J. A. Lampinen, Differential evolution: a practical approach to global optimization, Springer Science & Business Media, 2006.

[50] V. Santucci, M. Baioletti, A. Milani, Tackling Permutation-based Optimization Problems with an Algebraic Particle Swarm Optimization Algorithm, Fundamenta Informaticae 167 (1-2) (2019) 133–158. doi:10.3233/FI-2019-1812.

[51] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (2) (1993) 278 – 285.

[52] V. Santucci, M. Baioletti, A. Milani, Supplementary material for "An Algebraic Framework for Evolutionary Algorithms in Combinatorial Optimization".
URL `https://bit.ly/2RL3MMC`

[53] J. Derrac, S. García, D. Molina, 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 1 (1) (2011) 3–18.

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

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

[56] J. Liu, C. R. Reeves, Constructive and composite heuristic solutions to the $p//\sum c_i$ scheduling problem, European Journal of Operational Research 132 (2) (2001) 439–452.

[57] R. Martí, G. Reinelt, The linear ordering problem: exact and heuristic methods in combinatorial optimization, Springer Science & Business Media, 2011.

[58] J. Zhang, X. Zhu, W. Wang, J. Yao, A fast restarting particle swarm optimizer, in: Proc. of 2014 IEEE Congress on Evolutionary Computation (CEC 2014), 2014, pp. 1351–1358.

[59] Q. Duan, T. Liao, H. Yi, A comparative study of different local search application strategies in hybrid metaheuristics, Applied Soft Computing 13 (3) (2013) 1464 – 1477.

[60] J. Brest, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark

problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.

[61] A. Ismail, A. P. Engelbrecht, The self-adaptive comprehensive learning particle swarm optimizer, in: Proc. of 8th International Conference on Swarm Intelligence (ANTS 2012), 2012, pp. 156–167.

[62] D. Sudholt, C. Witt, Runtime analysis of a binary particle swarm optimizer, Theoretical Computer Science 411 (21) (2010) 2084–2100.

[63] M. Baioletti, A. Milani, V. Santucci, Automatic algebraic evolutionary algorithms, in: Proc. of International Workshop on Artificial Life and Evolutionary Computation (WIVACE 2017), 2018, pp. 271–283. doi:10.1007/978-3-319-78658-2_20.

[64] M. Baioletti, A. Milani, V. Santucci, Learning bayesian networks with algebraic differential evolution, in: Proc. of 15th International Conference on Parallel Problem Solving from Nature (PPSN XV), 2018, pp. 436–448. doi:10.1007/978-3-319-99259-4_35.

## Credit Author Statement

All the authors indicated in the article equally contributed to the work.