Geometric Differential Evolution for Combinatorial and Programs Spaces

Alberto Moraglio, Julian Togelius, and Sara Silva

***Abstract*—Geometric differential evolution (GDE) is a very recently introduced formal generalization of traditional differential evolution (DE) that can be used to derive specific GDE for both continuous and combinatorial spaces retaining the same geometric interpretation of the dynamics of the DE search across representations. In this article, we first review the theory behind the GDE algorithm, then, we use this framework to formally derive specific GDE for search spaces associated with binary strings, permutations, vectors of permutations and genetic programs. The resulting algorithms are representation-specific differential evolution algorithms searching the target spaces by acting directly on their underlying representations. We present experimental results for each of the new algorithms on a number of benchmark problems.**

***Index Terms*—Differential evolution, representations, principled design of search operators, combinatorial spaces, genetic programming, theory.**

I. Introduction

Two relatively recent additions to the Evolutionary Algorithms (EAs) family are Particle Swarm Optimization (PSO) [5], inspired to the flocking behavior of swarms of birds, and Differential Evolution [21], which is similar to PSO, but it uses different equations governing the motion of the particles. Despite their relatedness, DE is known to produce consistently better performance than PSO on many problems. In fact, DE is one of the most competitive EAs for continuous optimization [21].

In their initial inception, both PSO and DE were defined only for continuous problems. In both algorithms, the motion of particles is produced by linear combinations of points in space and has a natural geometric interpretation. There are a number of extensions of DE to binary spaces [21] [20], spaces of permutations [2] [19] and to the space of genetic programs [18]. Some of these works recast combinatorial optimization problems as continuous optimization problems and then apply the traditional DE algorithm to solve these continuous problems. Other works present DE algorithms defined directly on combinatorial spaces that, however, are only loosely related to the traditional DE in that the original geometric interpretation is lost in the transition from continuous to combinatorial spaces. Furthermore, every time a new solution representation is considered, the DE algorithm needs to be rethought and adapted to the new representation.

GDE [17] is a very recently devised formal generalization of DE that, in principle, can be specified to any solution representation while retaining the original

A. Moraglio is with the School of Computing, University of Kent, Canterbury, UK, e-mail: a.moraglio@kent.ac.uk.

J. Togelius is with the School of Computing, IT University of Copenhagen, Denmark, e-mail: julian@togelius.com.

S. Silva is with the INESC-ID group, Lisbon, Portugal, and with the Center for Informatics and Systems of the University of Coimbra, Portugal, e-mail: sara@kdbio.inesc-id.pt, sara@dei.uc.pt.

geometric interpretation of the dynamics of the points in space of DE across representations. In particular, GDE can be applied to any search space endowed with a distance and associated with any solution representation to derive formally a specific GDE for the target space and for the target representation. GDE is related to Geometric Particle Swarm Optimization (GPSO) [10], which is a formal generalization of the particle swarm optimization algorithm [5]. Specific GPSOs were derived for different types of continuous spaces and for the Hamming space associated with binary strings [11], for spaces associated with permutations [15] and for spaces associated with genetic programs [23].

The objective of the present article is to review the theory behind the GDE algorithm, illustrate how this framework can be used in practice as a tool for the principled design of DE search operators for standard and more complex solution representations associated with combinatorial spaces, and finally to test experimentally the new GDE algorithms endowed with such operators on benchmark problems. In particular, as target spaces for the GDE, we consider combinatorial spaces associated with binary strings, permutations and vectors of permutations and computer programs represented as expression trees.

The remaining part of the article is organized as follows. Section II contains a gentle introduction to a formal theory of representations that forms the context for the generalization of the DE algorithm. Section III briefly introduces the classic DE algorithm, and section IV describes the derivation of the general GDE algorithm. Section V presents specific GDE search operators for binary strings, and section VI reports experimental results on NK-landscapes and also on a second set of standard benchmark problems based on binary strings. Section VII presents specific GDE search operators for permutations, and section VIII reports experiments on the TSP. Section IX presents specific GDE search operators for Sudoku for which candidate solution grids are represented as vectors of permutations, and section X reports experimental results for this problem. Section XI presents specific GDE search operators for expression trees, and section XII reports the experimental analysis on standard GP benchmark problems. Section XIII presents conclusions and future work.

II. The geometry of representations

In this section, we introduce the ideas behind a recent formal theory of representations [9] which forms the context for the generalization of DE presented in the following sections.

Familiar geometric shapes in the Euclidean plane such as circles, ellipses, segments, semi-lines, triangles and convex polygons can be defined using distances between points in space. For example, a circle is the locus of points from which the distance to the centre c is a given constant value, the radius r. By replacing in the definition of a shape, say a circle, the Euclidean distance with a different distance, say the Hamming distance, we obtain the definition of a circle in the Hamming space. A circle in the Hamming space looks quite different from a circle in the Euclidean plane, however they both share the same geometric definition. Analogously, if we replace the Euclidean distance with the Manhattan distance, we obtain the definition of a circle in the Manhattan space. A number of simple geometric shapes based on the Manhattan distance in the plane have been derived explicitly (see Taxicab Geometry [7]). We can in fact replace the Euclidean distance in the definition of any geometric shape with any distance meeting a minimum number of requirements (metric), obtaining the corresponding shape in a space with a different geometry.

We can also raise the level of abstraction and replace the Euclidean distance with a generic metric, obtaining an abstract shape, such as for example an abstract circle. An abstract circle captures what is common to all circles across all possible geometries. Any property of an abstract circle is also a property of any space-specific circle.

Search algorithms can be viewed from a geometric perspective. The search space is seen as a geometric space with a notion of distance between points, and candidate solutions are points in the space. For example, search spaces associated with combinatorial optimization problems are commonly represented as graphs in which nodes corresponds to candidate solutions and edges between solutions correspond to neighbour candidate solutions. We can endow these spaces with a distance between solutions equal to the length of the shortest path between their corresponding nodes in the graph. Geometric search operators are defined using geometric shapes to delimit the region of search space where to sample offspring solutions relative to the positions of parent solutions. For example, geometric crossover is a search operator that takes two parent solutions in input corresponding to the end-points of a segment, and returns points sampled at random within the segment as offspring solutions. The specific distance associated with the search space at hand is used in the definition of segment to determine the specific geometric crossover for that space. Therefore, each search space is associated with a different space-specific geometric crossover. However, all geometric crossovers have the same abstract geometric definition.

In analytic geometry, in which points of the Cartesian plane are in one-to-one correspondence with pairs of numbers, their coordinates, the same geometric shape can be equivalently expressed geometrically as a set of points in the plane, or algebraically, by an equation whose solutions are the coordinates of its points. This is an important duality which allows us to treat geometric shapes as equations and vice versa. There is an analogous duality that holds for geometric search operators. Candidate solutions can be seen as points in space, geometric view, or equivalently, as syntactic configurations of a certain type, algebraic view. For example, a candidate solution in the Hamming space can be considered as a point in space or as a binary string corresponding to that point. The binary string can then be thought as being the coordinates of the point in the Hamming space. This allows us to think of a search operator equivalently as (i) an algorithmic procedure which manipulates the syntax of the parent solutions to obtain the syntactic configurations of the offspring solutions using well-defined representation-specific operations (algebraic view), or (ii) a geometric description which specifies what points in the space can be returned as offspring for the given parent points and with what probability (geometric view). For example, uniform crossover for binary strings [22] is a recombination operator that produces offspring binary strings by inheriting at each position in the binary string the bit of one parent string or of the other parent string with the same probability. This is an algebraic view of the uniform crossover that tells how to manipulate the parent strings to obtain the offspring string. Equivalently, the same operator can be defined geometrically as the geometric crossover based on the Hamming distance that takes offspring uniformly at random in the segment between parents.

There are two important differences between these two definitions of the same operator. The geometric definition is declarative, it defines what offspring the operator returns given their parents without explicitly

telling how to actually generate the offspring from the parents. The algebraic definition, on the other hand, is operational, since it defines the search operator by telling for each combination of parents how to build the corresponding offspring. The second important difference is that the geometric description of a search operator is representation-independent and refers only indirectly to the specific solution representation via a distance defined on such representation (i.e. edit distances such as the Hamming distance which can be defined on the binary string representation as the minimum number of bit-flips to obtain one string from the other). In contrast, the algebraic definition of a search operator is representation-dependent and uses operations which are well-defined on the specific solution representation but that may not be well-defined on other representations (e.g. bit-flip on a binary string is not well-defined on a permutation).

The duality of the geometric search operators has surprising and important consequences [9]. One of them is the possibility of principled generalization of search algorithms from continuous spaces to combinatorial spaces, as sketched in the following.

1) Given a search algorithm defined on continuous spaces, one has to recast the definition of the search operators expressing them explicitly in terms of Euclidean distance between parents and offspring.
2) Then one has to substitute the Euclidean distance with a generic metric, obtaining a formal search algorithm generalizing the original algorithm based on the continuous space.
3) Next, one can consider a (discrete) representation and a distance associated with it (a combinatorial space) and use it in the definition of the formal search algorithm to obtain a specific instance of the algorithm for this space.
4) Finally, one can use this geometric and declarative description of the search operator to derive its operational definition in terms of manipulation of the specific underlying representation.

This methodology was used to generalize PSO and DE to any metric space obtaining GPSO [10] and GDE [17] and then to derive the specific search operators for GPSO for a number of specific representations and distances. In the following sections, we illustrate how this methodology can be used in practice to generalize DE and to specialize it to specific metric spaces associated with a number of representations. The same methodology can be used to generalize to combinatorial spaces other algorithms naturally based on a notion of distance. This includes search algorithms such as Response Surface Methods, Estimation of Distribution Algorithms and Lipschitz Optimization algorithms, and also Machine Learning algorithms.

III. Classic Differential Evolution

In this section, we describe the traditional DE [21] (see algorithm 1).

The characteristic that sets DE apart from other evolutionary algorithms is the presence of the differential mutation operator (see line 5 of algorithm 1). This operator creates a mutant vector U by perturbing a vector $X3$ picked at random from the current population with the scaled difference of other two randomly selected population vectors $F \cdot (X1 - X2)$. This operation is understood being important because it adapts the mutation direction and its step size to the level of convergence and spatial distribution of the current population. The mutant vector is then recombined with the currently considered vector $X(i)$ using discrete recombination and

Algorithm 1 DE with differential mutation and discrete recombination

initialize population of N_p real vectors at random
while stop criterion not met **do**
  for all vector $X(i)$ in the population **do**
    pick at random 3 distinct vectors from the current population $X1, X2, X3$
    create mutant vector $U = X3 + F \cdot (X1 - X2)$ where F is the scale factor parameter
    set V as the result of the discrete recombination of U and $X(i)$ with probability Cr
    if $f(V) \geq f(X(i))$ **then**
      set the i^{th} vector in the next population $Y(i) = V$
    else
      set $Y(i) = X(i)$
    end if
  end for
  for all vector $X(i)$ in the population **do**
    set $X(i) = Y(i)$
  end for
end while

the resulting vector V replaces the current vector in the next population if it has better or equal fitness.

The differential mutation parameter F, known as scale factor, is a positive real normally between 0 and 1, but it can take also values greater than 1. The recombination probability parameter Cr takes values in $[0, 1]$. It is the probability, for each position in the vector $X(i)$, of the offspring V inheriting the value of the mutant vector U. When $Cr = 1$, the algorithm 1 degenerates to a DE algorithm with differential mutation only (because $V = U$). When $F = 0$, the algorithm 1 degenerates to a DE algorithm with discrete crossover only, as $U = X3$. The population size N_p normally varies from 10 to 100.

IV. Geometric differential evolution

Following the methodology outlined in section II, in this section we generalize the classic DE algorithm to general metric spaces. To do this, we recast differential mutation and discrete recombination as functions of the distance of the underlying search space, thereby obtaining their abstract geometric definitions. Then, in the following sections, we derive the specific DE algorithms for binary strings, permutations, vectors of permutations and genetic programs by plugging distances associated with these representations in the abstract geometric definition of the search operators.

A. Generalization of differential mutation

Let $X1, X2, X3$ be real vectors and $F \geq 0$ a scalar. The differential mutation operator produces a new vector U as follows:

$$U = X3 + F \cdot (X1 - X2) \tag{1}$$

The algebraic operations on real vectors in equation 1 can be represented graphically [21] as in figure 1. Real vectors are represented as points. The term $X1 - X2$ is represented as a vector originating in $X2$ and reaching $X1$. The multiplication with the scaling factor F produces a vector with the same origin and direction but with a different length. The addition of $X3$ to the scaled vector corresponds to the translation of the origin of the scaled vector from $X2$ to $X3$ keeping invariant its direction and length. The point of the (graphical) vector so obtained corresponds to the real vector U.

Unfortunately, this graphical interpretation of equation 1 in terms of operations on vectors does not help us to generalize equation 1 to general metric spaces because the notions of vector and operations on vectors are not

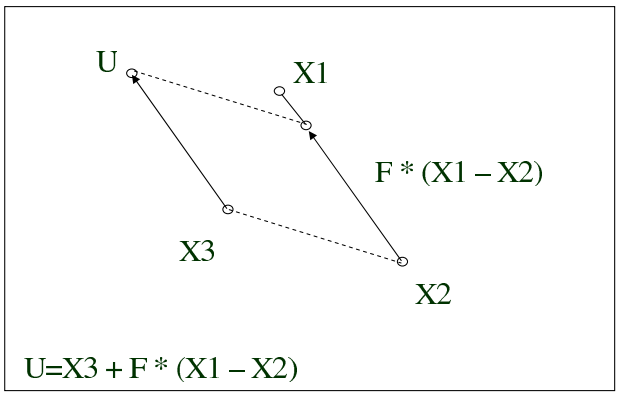

Fig. 1. Construction of U using vectors.

well-defined at this level of generality. In the following, we propose a generalization based on interpreting equation 1 in terms of segments and extension rays, which are geometric elements well-defined on any metric space. To do that, we need to rewrite equation 1 in terms of only convex combinations of two vectors, which are the algebraic dual of segments. A convex combination of a set of vectors is a linear combination of these vectors provided that their weights are all positive and sum up to one.

Equation 1 can be rewritten as:

$$U + F \cdot X2 = X3 + F \cdot X1 \tag{2}$$

By dividing both sides by $1 + F$ and letting $W = \frac{1}{1+F}$ we have:

$$W \cdot U + (1 - W) \cdot X2 = W \cdot X3 + (1 - W) \cdot X1 \tag{3}$$

Both sides of equation 3 are convex combinations of two vectors. On the left-hand side, the vectors U and $X2$ have coefficients W and $1 - W$, respectively. These coefficients sum up to one and are both positive because $W \in [0, 1]$ for $F \geq 0$. Analogously, the right-hand side is a convex combination of the vectors $X3$ and $X1$ with the same coefficients.

There is a interesting duality between the algebraic notion of convex combination of two vectors and the geometric notion of segment in the Euclidean space. Vectors represent points in space. The points P_C corresponding to the vectors C obtained by any convex combination of two vectors A and B lay in the line segment between their corresponding points P_A and P_B. The vice versa also holds true: the vector C corresponding to a point P_C in the segment $[P_A, P_B]$ can be obtained as a convex combination of the vectors A and B. The weights W_A and W_B in the convex combination localize the point on the segment $[P_A, P_B]$: distances to P_C from P_A and P_B are inversely proportional to the corresponding weights, W_A and W_B. So, the weight W_A of a vector A can be thought as the intensity of a linear attraction force towards a fix point P_A exerted on a movable point P_C. The stronger the force intensity W_A (relative to W_B) the closer the point P_C (understood as the equilibrium point of the attraction forces exerted by P_A and P_B) ends up being to P_A.

This duality allows for a geometric interpretation of equation 3 in terms of convex combinations (see figure 2). Let us call E the vector obtained by the convex combinations on both sides of equation 3. Geometrically the point E must be the intersection point of the segments $[U, X2]$ and $[X1, X3]$. The distances from E to the endpoints of these segments can be determined from equation 3 as they are inversely proportional to their respective weights. Since the point U is unknown (but its weight is known), it can be determined geometrically by firstly determining E as convex combination of $X1$ and $X3$; then, by projecting $X2$ beyond E (extension ray) obtaining a point U such that the proportions of the distances of $X2$ and U to the point E is inversely proportional to their weights. In the Euclidean space, the constructions of U using vectors (figure 1) and convex

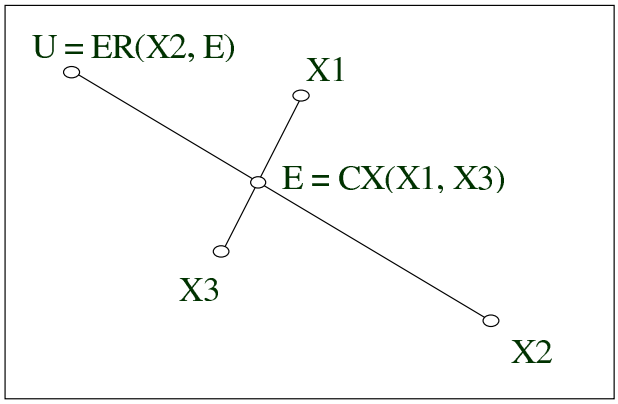

Fig. 2. Construction of U using convex combination and extension ray.

combinations (figure 2) are equivalent (algebraically, hence geometrically).

Segments and extension rays in the Euclidean space and their weighted extensions can be expressed in terms of distances, hence, these geometric objects can be naturally generalized to generic metric spaces by replacing the Euclidean distance with a generic metric. We will present their abstract definitions in section IV-C.

The differential mutation operator $U = DM(X1, X2, X3)$ with scale factor F can now be defined for any metric space following the construction of U presented in figure 2 as follows:

1) Compute $W = \frac{1}{1+F}$
2) Get E as the convex combination $CX(X1, X3)$ with weights $(1 - W, W)$ (generalizing $E = (1 - W) \cdot X1 + W \cdot X3$)
3) Get U as the extension ray $ER(X2, E)$ with weights $(W, 1 - W)$ (generalizing $U = (E - (1 - W) \cdot X2)/W$)

B. Generalization of discrete recombination

After applying differential mutation, the DE algorithm applies discrete recombination to U and $X(i)$ generating V. Discrete recombination is a geometric crossover under Hamming distance for real vectors [9]. The Hamming distance (HD) for real vectors is defined analogously to the Hamming distance between binary strings: it is the number of sites with mismatching values across the two vectors. From its definition, we can derive that the Cr parameter of the discrete recombination is proportional to the expected number of values that V inherits from U. Therefore, $E[HD(U, V)] = Cr \cdot HD(U, X(i))$ and $E[HD(X(i), V)] = (1 - Cr) \cdot HD(U, X(i))$. Consequently, Cr and $1 - Cr$ can be interpreted as the weights of U and $X(i)$, respectively, of the convex combination that returns V in the space of real vectors endowed with Hamming distance. In order to generalize the discrete recombination, by replacing hamming distance with a generic metric, we obtain the abstract convex combination operator CX introduced in the previous section. So, we have that the generalized discrete recombination of U and $X(i)$ with probability parameter Cr generating V is as follows: $V = CX(U, X(i))$ with weights $(Cr, 1 - Cr)$.

In the classic DE (algorithm 1), replacing the original differential mutation and discrete recombination operators with their generalizations, we obtain the formal Geometric Differential Evolution (see algorithm 2). When this formal algorithm is specified on the Euclidean space, the resulting Euclidean GDE does *not* coincide with the classic DE. This is because, whereas the original differential mutation operator can be expressed as a function of the Euclidean distance, the original discrete recombination operator can be expressed as a function of the Hamming distance for real vectors, not of the Euclidean distance. The Euclidean GDE coincides with an existing variant of traditional DE [21], which has the same differential mutation operator but in which the discrete recombination is replaced with blend crossover.

Interestingly, blend crossover lives in the same space as differential mutation and their joint behavior has a geometric interpretation in space.

Algorithm 2 Formal Geometric Differential Evolution

initialize population of N_p configurations at random
while stop criterion not met **do**
  for all configuration $X(i)$ in the population **do**
    pick at random 3 distinct configurations from the current population $X1, X2, X3$
    set $W = \frac{1}{1+F}$ where F is the scale factor parameter
    create intermediate configuration E as the convex combination $CX(X1, X3)$ with weights $(1 - W, W)$
    create mutant configuration U as the extension ray $ER(X2, E)$ with weights $(W, 1 - W)$
    create candidate configuration V as the convex combination $CX(U, X(i))$ with weights $(Cr, 1 - Cr)$ where Cr is the recombination parameter
    if $f(V) \geq f(X(i))$ **then**
      set the i^{th} configuration in the next population $Y(i) = V$
    else
      set $Y(i) = X(i)$
    end if
  end for
  for all configuration $X(i)$ in the population **do**
    set $X(i) = Y(i)$
  end for
end while

C. Definition of convex combination and extension ray

A notion of convex combination in metric spaces was introduced in the GPSO framework [10]. The notion of extension ray in metric spaces was introduced in the GDE framework [17]. That notion of convex combination requires distances of the fixed points to the equilibrium point to be a generic decreasing function of the weights of the fixed points. In the following, we present a more refined notion of convex combination in which the function relating weights and distances is given explicitly. Then the extended ray recombination can be naturally interpreted as the inverse operation of the convex combination.

Let us first recall the definition of segment and extension ray in metric spaces. Let (S, d) be a metric space. A (metric) segment is a set of the form $[x; y] = \{z \in S | d(x, z) + d(z, y) = d(x, y)\}$ where $x, y \in S$ are called end-points of the segment. The extension ray $ER(A, B)$ in the Euclidean plane is a semi-line originating in A and passing through B (note that $ER(A, B) \neq ER(B, A)$). The extension ray in a metric space can be defined indirectly using metric segments, as follows. Given points A and B, the (metric) extension ray $ER(A, B)$ is the set of points C that satisfy $C \in [A, B]$ or $B \in [A, C]$. Only the part of the extension ray beyond B will be of interest because the point C that we want to determine, which is, the offspring of the differential mutation operator, is never between A and B by construction.

We can now use these geometric objects as basis for defining the convex combination operator and the extended ray recombination operator, as follows.

The *convex combination* $C = CX((A, W_A), (B, W_B))$ of two points A and B with weights W_A and W_B (positive and summing up to one) in a metric space endowed with distance function d returns the set of points C such that $C \in [A, B]$ and $d(A, C)/d(B, C) = W_B/W_A$. In words, the weights of the points A and B are inversely proportional to their

distances to C [1]. When specified to Euclidean spaces, this notion of convex combination coincides with the traditional notion of convex combination of real vectors.

The *extension ray recombination* ER is defined as the inverse operation of the weighted convex combination CX, as follows. The weighted extension ray $ER((A, w_{ab}), (B, w_{bc}))$ of the points A (origin) and B (through) and weights w_{ab} and w_{bc} returns those points C such that their convex combination with A with weights w_{bc} and w_{ab}, $CX((A, w_{ab}), (C, w_{bc}))$, returns the point B. Notice that from this definition follows that the weights w_{ab} and w_{bc} in ER are positive real numbers between 0 and 1 and sum up to 1 because they must respect this condition in CX. The set of points returned by the weighted extension ray ER can be characterized explicitly in terms of distances to the input points of ER, as follows [17].

Lemma 1: The points C returned by the weighted extension ray $ER((A, w_{ab}), (B, w_{bc}))$ are exactly those points which are at a distance $d(A, B) \cdot w_{ab}/w_{bc}$ from B *and* at a distance $d(A, B)/w_{bc}$ from A.

Proof: From the definition of weighted extension ray we have that $B = CX((A, w_{ab}), (C, w_{bc}))$. Hence, $d(A, C) = d(A, B) + d(B, C)$ and the distances $d(A, B)$ and $d(B, C)$ are inversely proportional to the weights w_{ab} and w_{bc}. Consequently, $d(A, C) = d(A, B)/w_{bc}$ and substituting it in $d(B, C) = d(A, C) - d(A, B)$ we get $d(B, C) = d(A, B) \cdot w_{ab}/w_{bc}$, since $w_{ab} + w_{bc} + 1$. ■

This characterization is useful to construct procedures to implement the weighted extension ray for specific spaces. In fact, we used it, together with representation-specific properties of the extension ray, in the derivation of the extension ray recombination operators for all representations in this article.

The above definitions of convex combination and extension ray can be relaxed to obtain the required relation between weights and distances to the offspring only in expectation. These relaxed versions of the operators have the advantage of being more naturally suited to combinatorial spaces and being easier to implement for such spaces.

V. Binary GDE

In this section, we derive formally specific convex combination and extension ray recombination for the Hamming space for binary strings. These specific operators can then be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the Hamming space, the Binary GDE.

A. Convex combination

Let us consider the convex combination $C = CX((A, W_A), (B, W_B))$ of two points A and B with weights W_A and W_B (positive and summing up to one). In the Euclidean space, C is uniquely determined, however this is not the case for all metric spaces. In particular, it does not hold for Hamming spaces. When CX is specified to Hamming spaces on binary strings, it can be formally shown that we obtain the recombination operator outlined in algorithm 3 [10]. This algorithm returns a offspring binary string C of parent binary strings A and B such that $HD(A, C)/HD(B, C) = W_B/W_A$ in expectation, where HD denotes the Hamming distance between binary strings. This differs from the Euclidean case where this ratio is guaranteed.

B. Extension ray

In order to gain an intuitive understanding of how an extension ray looks like in the Hamming space, let us

[1] To allow weights and distances to assume value zero and avoid problems with division by zero, the requirement in the definition can be changed as $d(A, C) \cdot W_A = d(B, C) \cdot W_B$.

Algorithm 3 Binary Convex Combination Operator

1: inputs: binary strings A and B and weights W_A and W_B (weights must be positive and sum up to 1)
2: **for all** position i in the strings **do**
3: **if** random(0,1) $\leq W_A$ **then**
4: set $C(i)$ to $A(i)$
5: **else**
6: set $C(i)$ to $B(i)$
7: **end if**
8: **end for**
9: return string C as offspring

consider an example of extension ray originating in $A = 110011$ and passing through $B = 111001$.

The relation $C \in [A, B]$ is satisfied by those C that match the schema $S1 = 11 * 0 * 1$. This is the set of the possible offspring of A and B that can be obtained by recombining them using the uniform crossover.

The relation $B \in [A, C]$ is satisfied by all those C that match $S2 = **1*0*$. This is the set of all those C that when recombined with A using the uniform crossover can produce B as offspring.

The following theorem characterizes the extension ray in the Hamming space in terms of schemata.

Theorem 2: Let A and B be fixed binary strings in the Hamming space:

1) the relation $C \in [A, B]$ is satisfied by those strings C that match the schema obtained by keeping the common bits in A and B and inserting $*$ where the bits of A and B do not match.
2) the relation $B \in [A, C]$ is satisfied by all those strings C that match the schema obtained by inserting $*$ where the bits are common in A and B and inserting the bits coming from B where the bits of A and B do not match.

Proof:

Proof of statement 1: the schema so defined corresponds to the set of the possible offspring of A and B that can be obtained by recombining them using the uniform crossover. This crossover operator corresponds to the uniform geometric crossover under Hamming distance which returns offspring in the segment between parents.

Proof of statement 2: all C matching the schema S defined in this statement recombined with A can produce B as offspring. This is because at each position (in A, B and C) when in the schema S there is $*$ the bit in B at that position can be inherited from A. When in the schema there is a bit (0 or 1) the bit in B at that position can be inherited from C. Furthermore, only the strings C matching S can produce B when C is recombined with A. ■

Using the characterization of the weighted extension ray in terms of distances (lemma 1) and the characterization of the extension ray in the Hamming space in terms of schemata (theorem 2), we were able to derive the weighted extension ray recombination for this space (see algorithm 4). Theorem 3 proves that this recombination operator conforms to the definition of weighted extension ray for the Hamming space (in expectation).

Theorem 3: Given parents A and B, the recombination in algorithm 4 returns an offspring C such that $E[HD(B, C)]/HD(A, B) = W_{AB}/W_{BC}$, where $E[HD(B, C)]$ is the expected Hamming distance between B and the offspring C.

Proof: This can be shown as follows. The number of bits in which A and B differ are $HD(A, B)$. The number of bits in which A and B do not differ is $n - HD(A, B)$. For the bits in which A and B differ, the string C equals B. For each bit in which A and B do not differ, C does not equal B with

Algorithm 4 Binary Extension Ray Recombination

1: inputs: binary strings A (origin) and B (through) of length n and weights W_{AB} and W_{BC} (weights must be positive and sum up to 1)
2: set $HD(A,B)$ as Hamming distance between A and B
3: set $HD(B,C)$ as $HD(A,B) \cdot w_{AB}/w_{BC}$ (compute the distance between B and C using the weights)
4: set p as $HD(B,C)/(n - HD(A,B))$ (this is the probability of flipping bits away from A and B beyond B)
5: **for all** position i in the strings **do**
6: set $C(i) = B(i)$
7: **if** $B(i) = A(i)$ and random(0,1) $\leq p$ **then**
8: set $C(i)$ to the complement of $B(i)$
9: **end if**
10: **end for**
11: return string C as offspring

probability p. So, the expected distance between B and C is $E[HD(B,C)] = (n - HD(A,B)) \cdot p$. By substituting $p = HD(B,C)/(n - HD(A,B))$, we have $E[HD(B,C)] = HD(B,C) = HD(A,B) \cdot W_{AB}/W_{BC}$. So, $E[HD(B,C)]/HD(A,B) = W_{AB}/W_{BC}$. ■

Theorem 3 holds under the assumption that the diameter of the space is at least as large as the wanted Hamming distance between A and C. That is, that the requested point on the extension ray does not go beyond the boundary of the space. When such a condition does not hold, the offspring C returned by the algorithm 4 is the point on the extension ray at maximum distance from A. In this case, the required relation between distance and weights does not hold.

Now we have operational definitions of convex combination and extension ray for the space of binary strings under HD. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of binary strings.

VI. Experiments for Binary GDE

We implemented the GDE algorithm for binary spaces within a Java framework,[2] and investigated its performance on some benchmark problems. The proposed algorithm was compared with three other algorithms:

- cGA: A canonical Genetic Algorithm, with roulette wheel fitness-proportionate selection, uniform crossover and bitflip mutation.
- tGA: a Genetic Algorithm with truncation selection, with a selection threshold of $popsize/2$.
- ES: A $\mu + \lambda$ Evolution Strategy, with $\mu = \lambda = popsize/2$ and bitflip mutation.

For the first benchmark suite, we also compared it with:

- BPSO: Discrete Binary PSO of Kennedy and Eberhart, using the results presented in [4].

For the ES and GAs, the bitflip mutation works as follows: each bit in the chromosome is considered, and with probability p this bit is flipped. In the experiments involving these algorithms, this parameter was systematically varied between 0.0 and 0.5 in increments of 0.01. For the experiments involving GDE, the key parameters F and Cr were systematically varied between 0.0 and 1.0 in increments of 0.1.

In all experiments, the length of any single run was set to 4000 function evaluations, in order to be directly comparable with the results of Kennedy and Eberhart. For GDE, GA and ES the population size was varied systematically: sizes of 10, 20, 40, 80 and 160 were tried,

[2]Source code is available upon request from the second author.

with the numbers of generations limited appropriately: 400, 200, 100, 50 and 25.

A. Spears-DeJong functions

We used three of the same benchmark problems that Kennedy and Eberhart tested their binary PSO on. These are William Spears' binary versions of DeJong's functions $f1$, $f2$ and $f3$.[3] (We did not use $f4$ and $f5$ due to unresolved differences between different versions of the code, which might be due to differing numerical precision in different systems; further, Kennedy and Eberhart do not report precise results for $f4$.)

Each configuration (parameters and population size) was tested twenty times, and the average best score of each run was recorded, as well as how many of the runs that reached the global optimum. The results are summarized in table I. The parameters were optimized separately for each combination of benchmark function and algorithm, and only the results of the best configuration are reported here. The best parameter settings found are reported in tables II and III.

The GDE algorithm appears to work best with small population sizes, 10 or 20 individuals (and thus more generations). On $f1$, there is a clear preference for high values (e.g. 0.9) of both F and Cr, whereas on $f2$ the algorithm seems to work best with values of around 0.3 for both parameters.

In comparison, both GAs always work best with large populations and relatively high mutation rates (> 0.1). The ES seems to be relatively insensitive to population size, as long as the mutation rate is in the region 0.05–0.1.

The compressed fitness structure of $f1$ and $f2$, with many local optima with values differing from the global optimum only in the third decimal, is apparently a bad match with fitness-proportional selection; the landscape of $f3$ has similar characteristics but to a lesser degree. Therefore, the results of the canonical GA are the worst on all problems.

Both the evolution strategy and the GA with truncation selection are strictly better on all benchmarks than the canonical GA; binary PSO is better than ES in that it reaches optimum more often on $f1$ and $f2$; and GDE is the best algorithm overall, as it is as good as BPSO on $f2$ and $f3$ but reaches the global optimum almost twice as often on $f1$.

Algorithm	$f1$ (78.6)		$f2$ (3905.93)		$f3$ (55.0)	
BPSO	-	10	-	4	-	20
GDE	78.5999	19	3905.9296	4	55.0	20
cGA	78.2152	0	3905.8052	0	52.1	1
tGA	78.5993	4	3905.9266	2	55.0	20
ES	78.5998	7	3905.9291	2	55.0	20

TABLE I

RESULTS ON THE SPEARS-DEJONG BENCHMARK SUITE. THE MAXIMA OF THE FUNCTIONS ARE REPORTED NEXT TO THEIR NAMES. FOR EACH COMBINATION OF ALGORITHM AND PROBLEM, THE RESULTS OF THE BEST PARAMETERIZATION OF THAT COMBINATION ARE REPORTED. THE FIRST NUMBER IS THE BEST FITNESS OF THE LAST GENERATION, AVERAGED OVER 20 RUNS. THE SECOND NUMBER IS THE NUMBER OF THOSE RUNS THAT REACHED THE GLOBAL OPTIMUM.

Function	pop/gen	F	Cr
$f1$	10/400	0.9	0.8
$f2$	20/200	0.3	0.3
$f3$	*	*	*

TABLE II

BEST PARAMETER SETTINGS FOUND FOR GDE ON THE SPEARS-DEJONG BENCHMARKS. THE ASTERISKS DENOTE THAT MANY COMBINATIONS ARE OPTIMAL.

[3]The original c source code of these functions can be found at http://www.cs.uwyo.edu/∼wspears/functs/dejong.c

cGA	pop/gen	mutation
$f1$	160/25	0.12
$f2$	160/25	0.16
$f3$	80/50	0.39
tGA	pop/gen	mutation
$f1$	80/50	0.29
$f2$	80/50	0.1
$f3$	80/50	0.45
ES	pop/gen	mutation
$f1$	10/400	0.13
$f2$	160/25	0.1
$f3$	*	*

TABLE III

BEST PARAMETER SETTINGS FOUND FOR GA (WITH TRUNCATION AND ROULETTE-WHEEL SELECTION) AND ES ON THE SPEARS-DEJONG BENCHMARKS. THE ASTERISKS DENOTE THAT MANY COMBINATIONS ARE OPTIMAL.

B. NK Landscapes

In order to more systematically test the behaviour of GDE on landscapes with varying amount of epistasis, we performed additional experiments using NK fitness landscapes, as proposed by Kauffman [3]. NK landscapes have two paramters: N, the number of dimensions, was fixed to 100 in our experiments; K, the number of dependencies on other loci per locus was varied between 0 and 4. The parameters of the algorithms (mutation rate, F and Cr) were varied in the same way as with the Spears-DeJong experiments above. All evolutionary runs lasted for 10000 function evaluations, which were allocated either as population size 100 and 100 generations or as population size 10 and 1000 generations.

The results in table IV show that GDE is a very competitive algorithm overall. For population size 100, GDE is the best of the four algorithms for K of 1, 2 and 3, and a close second for K of 0 and 4. For population size 10, GDE is the best algorithm for all K except $K = 0$. The results further show that the ES and the GA with truncation selection perform significantly better than the canonical GA for all K.

Table V shows the best parameter settings for GDE for different K. Apparently, for low K larger population sizes are preferred, and for higher K smaller populations do better. Interestingly, for all K the best configuration is very low F and medium to high Cr. Table VI presents the best parameter settings found for ES and GA. A very clear trend is that ES works best with small populations and both GAs with larger populations; ES also generally prefers lower mutation rate than the GAs.

10	$K=0$	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$
GDE	0.623	0.730	0.732	0.758	0.751	0.741
cGA	0.521	0.509	0.515	0.536	0.519	0.517
tGA	0.597	0.621	0.613	0.621	0.641	0.641
ES	0.667	0.721	0.746	0.740	0.736	0.727
100	$K=0$	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$
GDE	0.665	0.750	0.738	0.756	0.736	0.719
cGA	0.552	0.594	0.613	0.610	0.600	0.610
tGA	0.664	0.707	0.713	0.736	0.737	0.730
ES	0.677	0.696	0.710	0.717	0.717	0.720

TABLE IV

RESULTS ON THE NK LANDSCAPE BENCHMARK. AVERAGE MAXIMUM FITNESS AT THE LAST GENERATION FOR EACH ALGORITHM USING K VALUES BETWEEN 0 AND 5, USING POPULATION SIZES OF BOTH 10 AND 100. 50 RUNS WERE PERFORMED FOR EACH CONFIGURATION.

VII. PERMUTATION-BASED GDE

In this section, we derive formally specific convex combination and extension ray recombination for the space of permutations. We use the swap distance between permutations as basis for the GDE. These specific operators can then be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of permutations, the permutation-based GDE. Notice, however, that in principle, we could choose any other distance between

K	pop/gen	F	Cr
0	100/100	0.0	0.8
1	100/100	0.0	0.7
2	100/100	0.0	0.5
3	10/1000	0.1	0.9
4	10/1000	0.1	0.8
5	10/1000	0.1	0.8

TABLE V
BEST PARAMETER SETTINGS FOUND FOR GDE ON THE NK LANDSCAPE BENCHMARK.

K	cGA	tGA	ES
0	0.01	0.35	0.01
1	0.01	0.43	0.03
2	0.28	0.47	0.03
3	0.16	0.19	0.04
4	0.39	0.36	0.02
5	0.20	0.30	0.02

TABLE VI
BEST MUTATION SETTINGS FOUND FOR GA AND ES ON THE NK LANDSCAPE BENCHMARK. THE GAS ALWAYS PERFORMED BEST WITH POPULATION SIZE 100, AND THE ES WITH POPULATION SIZE 10.

permutations (e.g., adjacent swap distance, reversal distance, insertion distance, etc.) as a basis of the GDE. In that case, for each distance, we would obtain a different permutation-based GDE.

A. Swap distance

The swap distance between two permutations is the minimum number of swaps needed to order one permutation into the order of the other permutation. It can be implemented as in Algorithm 5.

Algorithm 5 Swap distance

inputs: permutations p_a and p_b
set $dist = 0$
for all position i in the permutations **do**
  if $p_a(i) \neq p_b(i)$ **then**
    find $p_a(i)$ in p_b and be j its position in p_b
    swap contents of $p_b(i)$ and $p_b(j)$
    $dist = dist + 1$
  end if
end for
return $dist$

B. Convex combination

Algorithm 6 presents a recombination operator for permutations that was introduced in the GPSO framework [11]. This operator produces an offspring by sorting by swaps the two parent permutations one towards the other until they converge to the same permutation. Which of the two permutations has to be sorted toward the other at each position is controlled by the contents of a random recombination mask generated using the parents weights interpreted as probabilities of the outcome of tossing a biased coin being the respective parent. This operator is called 'convex combination' because it is allegedly a convex combination for permutations under swap distance. However, this needs to be proved.

Theorem 4: The convex combination in Algorithm 6 is a geometric crossover under swap distance [11].

Additionally, in previous work [11], it was shown that the distances of the parents to the offspring are decreasing functions of their weights in the convex combination. In the following, we give a stronger result that says that that these distances are inversely proportional to the corresponding weights, as required by the refined definition of convex combination introduced in this article.

Theorem 5: The convex combination in Algorithm 6 is (in expectation) a convex combination in the space of permutations endowed with swap distance.

Algorithm 6 Convex combination

inputs: permutations p_a and p_b, and their weights w_a and w_b
generate a recombination mask m randomly with 'a' and 'b' with probabilities w_a and w_b
for all position i in the permutations **do**
  if $p_a(i) \neq p_b(i)$ **then**
    if $m(i) = a$ **then**
      find $p_a(i)$ in p_b and be j its position in p_b
      swap contents of $p_b(i)$ and $p_b(j)$
    else
      find $p_b(i)$ in p_a and be j its position in p_a
      swap contents of $p_a(i)$ and $p_a(j)$
    end if
  end if
end for
return p_a as offspring

Proof: The convex combination for permutations is a geometric crossover under swap distance. Therefore, the offspring of the convex combination are in the segment between parents as required to be a convex combination. To complete the proof, we need to show that the weights w_a and w_b of the convex combination are inversely proportional to the expected distances $E[SD(p_a, p_c)]$, $E[SD(p_b, p_c)]$ from the parents p_a and p_b to their offspring p_c, as follows.

The recombination mask m contains a set of independently generated choices. The effect of each choice is sorting p_a a single swap towards p_b with probability w_b and sorting p_b a single swap towards p_a with probability w_a, when p_a and p_b differ at the current position. When p_a and p_b are equal at the current position, the effect of the choice is to leave p_a and p_b unchanged. When all choices in the mask m have been applied p_a and p_b have become equal in all positions, hence converged to the offspring p_c. Since the convex combination operator is a geometric crossover, the offpring p_c is on a shortest path between p_a and p_b (shortest sorting trajectory by swaps). The expected number of swap moves on the shortest path from p_a toward p_b to reach p_c, i.e., $E[SD(p_a, p_c)]$, is given by the number of swap moves on the shortest path, i.e., $SD(p_a, p_b)$, multiplied by the probability that any swap move on the shortest path was obtained by ordering p_a toward p_b, i.e., w_b. Hence $E[SD(p_a, p_c)] = SD(p_a, p_b) \cdot w_b$. Analogously for the other parent we obtain: $E[SD(p_b, p_c)] = SD(p_a, p_b) \cdot w_a$. Therefore, the expected distances of the parents to the offspring are inversely proportional to their respective weights. ■

C. Extension ray

Algorithm 7 presents a recombination operator that is allegedly the extension ray recombination for permutations under swap distance. This operator produces an offspring permutation by sorting by swaps parent permutation p_b away from parent permutation p_a. The number of swaps away is calculated in a way to obtain consistency between weights and distances of the offspring to the parents as required from the general definition of extension ray recombination in metric space. The following theorem proves that this is indeed an extension ray recombination for permutations under swap distance.

Theorem 6: The extension ray recombination in Algorithm 7 is (in expectation) an extension ray operator in the space of permutations endowed with swap distance.

Proof: First we prove that $p_c = ER(p_a, p_b)$ by proving that p_b is in the segment between p_a and p_c under swap distance. Then we prove that the expected distances $E[SD(p_a, p_b)]$ and $E[SD(p_b, p_c)]$ are inversely proportional to the weights w_{ab} and w_{bc}, respectively.

Algorithm 7 Extension ray recombination

1: inputs: parent p_a (origin point of the ray) and p_b (passing through point of the ray), with corresponding weights w_{ab} and w_{bc} (both weights are between 0 and 1 and sum up to 1)
2: output: a single offspring p_c (a point on the extension ray beyond p_b on the ray originating in p_a and passing through p_b)
3: compute the swap distance $SD(p_a, p_b)$ between p_a and p_b
4: set $SD(p_b, p_c) = SD(p_a, p_b) \cdot w_{ab}/w_{bc}$ (compute the distance between p_b and p_c using the weights)
5: set $p = SD(p_b, p_c)/(n - 1 - SD(p_a, p_b))$ (the probability p of swapping elements away from p_a and p_b beyond p_b)
6: set $p_c = p_b$
7: **for all** position i in the permutations **do**
8: **if** $p_c(i) = p_a(i)$ and random(0,1) $\leq p$ **then**
9: select at random a position j
10: swap contents of $p_c(i)$ and $p_c(j)$
11: **end if**
12: **end for**
13: return p_c as offspring

Every swap move applied to p_b that increases the Hamming distance between p_a and p_b generate a permutation p'_b such that p_b is on a swap shortest path between p_a and p'_b. This is because (i) p'_b is a swap away from p_b, i.e., $SD(p_b, p'_b) = 1$ and (ii) p'_b is a swap further away from p_a since $HD(p_a, p'_b) > HD(p_a, p_b)$, i.e., $SD(p_a, p_b) + 1 = SD(p_a, p'_b)$. Hence $SD(p_a, p_b) + SD(p_b, p'_b) = SD(p_a, p'_b)$. This construction can be continued applying a swap move to p'_b obtaining a p''_b such that p'_b and p_b are on a swap shortest path between p_a and p''_b. Analogously, for any further reiteration, we obtain $p_b^{(n)}$ such that p_b is on a swap shortest path between p_a and $p_b^{(n)}$. Since the operator ER constructs the offspring p_c (corresponding to $p_b^{(n)}$) from parents p_a and p_b following the above procedure, we have that p_b is in the segment between p_a and p_c under swap distance.

The probability p is the probability of applying a swap away from p_a for each position i, for which p_a equals p_b. The wanted distance $SD(p_b, p_c)$ to have distances and weights of parents inversely proportional is calculated from the weights w_{ab} and w_{bc}, and the known distance $SD(p_a, p_b)$. The probability p is then set to $SD(p_b, p_c)$ over the number of positions for which p_a equals p_b. This number is well estimated by the maximum number of swaps away from p_a that can be applied to p_b. The last number is given by the length of the diameter of the space (maximum swap distance between any two permutations), which is $n-1$ where n is the number of elements in the permutation, minus the swap distance between p_a and p_b. Hence, the expected number of swaps away from p_b done equals the wanted distance $SD(p_b, p_c)$. ■

As for the case of the Hamming space, the extension ray recombination operator for permutations cannot return points which are farther away than the diameter of the space. When input weights require this, the point actually returned by the operator is the farthest away point on the extension ray.

Now we have operational definitions of convex combination and extension ray for the space of permutations under swap distance. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of permutations.

VIII. Experiments with GDE on TSP

We have tested the permutation-based GDE on randomly generated instances of the Travelling Salesman Problem (TSP), which is perhaps the most famous permutation-based optimization problem. We do not expect the GDE to be comparable in performance with the state-of-the-art search algorithms customized to such a well-studied problem. Also, the neighborhood structure on the TSP that works best with local search heuristics is that based on the 2-opt move which reverses the order of the elements in a continuous section of the permutation. Analogously to the swap move, this move gives rise to a distance between permutations (known as reversal distance). This would be perhaps the most suitable distance to use as a base for GDE when applied to TSP. We will test this in future work.

Local search heuristics based on the swap move are known to do reasonably well on the TSP. Also, genetic algorithms with the PMX crossover operator for permutation, which is known to be a geometric crossover under swap distance, does reasonably well on the TSP. Therefore, as a reference, we compare the GDE on the swap space with a stochastic hill-climber based on the swap move and with a genetic algorithm with rank-based selection, PMX crossover and swap mutation.

The TSP instances used in our experiments are randomly generated, with either 50 cities for small problems or 200 cities for large problems. The distance between each pair of cities lies between 0 and 1, and the instances are symmetric but not Euclidean. Twenty TSP instances were generated at the beginning of the experiments; every algorithm configuration is run once per instance, and the fitness averaged over all instances.

Moderately extensive tuning experiments were performed for the population-based algorithms. All algorithms (GDE, GA and hill climber) were run for 100000 (hundred thousand) function evaluations. For GDE and GA, population sizes of 10, 20, 50, 100, 1000 were tried, with the number of generations set to $100000/popsize$. For both algorithms, their two respective key parameters were varied between 0 and 1 in increments of 0.2; for GDE, the parameters are F and Cr. For the GA, these parameters were defined as the elite proportion (how large part of the rank-ordered population is used as the elite; the lesser fit rest of the population is replaced each generation) and mutation probability (the probability that a new offspring is created through swap mutation from the previous individual at the same position in the population rather than using PMX crossover of two randomly selected individuals in the population). We note that some extreme settings yield degenerate versions of both algorithms. Alas, for the hillclimber, there is nothing to tune.

Algorithm	Fitness	Population	Parameters
Hillclimber	5.37	-	-
GA	5.13	10	elite 0.2, mut 0.6
GDE	5.35	10	F 0.0, Cr 0.2

TABLE VII
Results on TSP instances of size 50. Lower fitnesses are better.

Algorithm	Fitness	Population	Parameters
Hillclimber	14.83	-	-
GA	21.92	10	elite 0.2, mut 0.4
GDE	4461	10	F 0.2, Cr 0.2

TABLE VIII
Results on TSP instances of size 200. Lower fitnesses are better.

The results (table VII) show that a well-tuned GDE

F/Cr	0.0	0.2	0.4	0.6	0.8	1.0
0.0	20.91	22.03	21.58	21.89	21.63	21.37
0.2	5.35	6.96	7.08	7.22	7.43	7.53
0.4	9.38	8.44	9.83	10.77	11.43	11.92
0.6	11.46	7.91	11.61	12.83	13.4	13.93
0.8	13.8	9.79	12.15	13.63	13.93	13.41
1.0	19.23	14.63	13.86	12.86	13.15	13.22

TABLE IX

PARAMETER SETTINGS AND CORRESPONDING AVERAGE FITNESS ON TSP INSTANCES OF SIZE 50, USING POPULATION SIZE 10, WHERE THE BEST SETTING WAS FOUND. THE F PARAMETER ON THE HORIZONTAL AXIS, AND CR ON THE VERTICAL. LOWER FITNESSES ARE BETTER.

outperforms a hillclimber and is competitive with a well-tuned GA (it outperforms many settings of the GA) on small instance sizes. On larger instances, it seems GDE is no longer competitive with the GA, at least for not for the parameter settings explored here (see Table VIII).

Table IX presents a comparison of the performance of GDE with population size 10 on small instances under different settings of the parameters F and Cr. It is clear that the Cr parameter should be small but non-zero; with the current search granularity, a Cr setting of 0.2 is optimal for all settings of F. The setting of F is less obvious, as it is possible to find good Cr settings for all values of F; however, in general lower settings are better. It should be pointed out that a parameter search with finer granularity would reveal even better settings, as informal investigations have revealed.

IX. GDE FOR SUDOKU

The Sudoku puzzle is a perfect candidate to test new algorithmic ideas because it is entertaining and instructive as well as a non-trivial constrained combinatorial problem. We have used it in previous work to test GPSO [15] and a GA [16] with geometric operators based on a vector-of-permutations solution representation, which is a natural representation for Sudoku grids, associated with row-wise swap distance. In this section, we derive the specific GDE for Sudoku based on this space. Then, in section X, we present experimental results and compare the performance of GA, GPSO and GDE.

A. Sudoku solving as optimization problem

Sudoku is a logic-based placement puzzle. The aim of the puzzle is to enter a digit from 1 through 9 in each cell of a 9x9 grid made up of 3x3 subgrids (called "regions"), starting with various digits given in some cells (the "givens"). Each row, column, and region must contain only one instance of each digit. Sudoku puzzles with a unique solution are called proper sudoku, and the majority of published grids are of this type. The general problem of solving Sudoku puzzles on $n^2 \times n^2$ boards of n x n blocks is known to be NP-complete [24].

Sudoku is a constraint satisfaction problem with 4 types of constraints:

1) Fixed elements
2) Rows are permutations
3) Columns are permutations
4) Boxes are permutations

It can be cast as an optimization problem by choosing some of the constraints as hard constraints that all solutions have to respect, and the remaining constraints as soft constraints that can be only partially fulfilled and the level of fulfillment is the fitness of the solution. We consider a space with the following characteristics:

- *Hard constraints*: fixed positions and permutations on rows
- *Soft constraints*: permutations on columns and boxes
- *Distance*: sum of swap distances between paired rows (row-swap distance)

Fitness function (to maximize): sum of number of unique elements in each row, plus, sum of number of unique elements in each column, plus, sum of number of unique elements in each box. So, for a 9×9 grid we have a maximum fitness of $9 \cdot 9 + 9 \cdot 9 + 9 \cdot 9 = 243$ for a completely correct Sudoku grid and a minimum fitness little more than $9 \cdot 1 + 9 \cdot 1 + 9 \cdot 1 = 27$ because for each row, column and square there is at least one unique element type.

It is possible to show that the fitness landscapes associated with this space is smooth, making the search operators proposed a good choice for Sudoku.

B. Geometric crossovers and mutation for Sudoku

In previous work [16], we presented geometric crossovers and mutations based on the space of vectors of permutations endowed with the row-swap distance. The geometric mutation swaps two non-fixed elements in a row. The geometric crossovers are the row-wise PMX and row-wise cycle crossover.

This mutation preserves both fixed positions and permutations on rows because swapping elements within a row that is a permutation returns a permutation. The mutation is 1-geometric under row-swap distance.

Row-wise PMX and row-wise cycle crossover recombine parent grids applying respectively PMX and cycle crossover to each pair of corresponding rows. In case of PMX the crossover points can be selected to be the same for all rows, or be random for each row. In terms of offspring that can be generated, the second version of row-wise PMX includes all the offspring of the first version.

Simple PMX and simple cycle crossover applied to parent permutations return always permutations. They also preserve fixed positions. This is because both are geometric under swap distance and in order to generate offspring on a minimal sorting path between parents using swaps (sorting one parent into the order of the other parent) they have to avoid swaps that change common elements in both parents (elements that are already sorted). Therefore also row-wise PMX and row-wise cycle crossover preserve both hard constraints.

Using the product geometric crossover theorem [13], it is immediate that both row-wise PMX and row-wise cycle crossover are geometric under row-swap distance, since simple PMX and simple cycle crossover are geometric under swap distance. Since simple cycle crossover is also geometric under Hamming distance (restricted to permutations), row-wise cycle crossover is also geometric under Hamming distance.

Finally, notice that to restrict the search to the space of grids with fixed positions and permutations on rows, the initial population must be seeded with feasible random solutions taken from this space. Generating such solutions can be done still very efficiently.

C. Extension ray and Convex combination in product spaces and subspaces

In the following, we present general theoretical results that allow us to build new convex combination (or extension ray recombination) by combining operators that are known to be convex combinations (or extension ray recombinations) and by restricting the domain of known convex combinations (or extension ray recombinations). These results are very useful to deal in a natural way with the compound structure of Sudoku solutions and their hard constraints. We illustrate their application to Sudoku in the following section. Notice that the results on convex combination presented in this section refine those presented earlier within the GPSO framework [15].

Theorem 7: The operator on the product space obtained by combining vector-wise a set of convex combi-

nation operators is a convex combination on the product space endowed with the distance obtained by summing the distances of the composing convex operators.

Proof: Let us consider the convex combination operators $CX_1(S_1, S_1) \rightarrow S_1, CX_2(S_2, S_2) \rightarrow S_2, ..., CX_n(S_n, S_n) \rightarrow S_n$. Let the compound operator on the product space $S = S_1 \times S_2 \times ... \times S_n$ $CX(S, S) \rightarrow S$ be defined as $CX(S, S) = (CX_1(S_1, S_1), CX_2(S_2, S_2), ..., CX_n(S_n, S_n))$. Since $CX_1, CX_2, ..., CX_n$ are convex combination operators they are also geometric crossover under distances $d_1, d_2, ..., d_n$. For the product geometric crossover theorem [13], the compound operator CX is a geometric crossover under the distance $d = d_1 + d_2 + ... + d_n$.

To prove that CX is a convex combination on d, we need to prove that applying $CX_1, CX_2, ..., CX_n$ all with the same parent weights w_a and w_b on their respective spaces $(S_1, d_1), (S_2, d_2), ..., (S_n, d_n)$ and grouping their offspring in a vector is equivalent to applying CX on the space (S, d) with weights w_a and w_b. Let be $c' = CX_1(a', b'), c'' = CX_1(a'', b''), ..., c^{(n)} = CX(a^{(n)}, b^{(n)})$. We have that $d_1(a', c') = d_1(a', b') \cdot w_b, d_2(a'', c'') = d_2(a'', b'') \cdot w_b, ..., d_n(a^{(n)}, c^{(n)}) = d_n(a^{(n)}, b^{(n)}) \cdot w_b$. Summing these equations we obtain $d_1(a', c') + d_2(a'', c'') + ... + d_n(a^{(n)}, c^{(n)}) = (d_1(a', b') + d_2(a'', b'') + ... + d_n(a^{(n)}, b^{(n)})) \cdot w_b$. This can be rewritten in terms of the distance d as $d((a', a'', ..., a^{(n)}), (c', c'', ..., c^{(n)})) = d((a', a'', ..., a^{(n)}), (b', b'', ..., b^{(n)})) \cdot w_b$. An analogous result holds for the parents $b', b'', ..., b^{(n)}$ with respect to the weight w_a. This means that CX is a weighted combination with respect to the distance d. ■

Theorem 8: The operator on the sub space obtained by restricting the domain of application of a convex combination operator is a convex combination operator on the sub space endowed with the distance of the original convex combination operator.

Proof: Let $C = \{c_i\}$ the set offspring obtained by $CX(a, b)$ with weights w_a and w_b on the original space (S, d). The operator CX is a convex combination if and only if for any c_i we have that $d(a, c_i) + d(c_i, b) = d(a, b)$ and that $d(a, c_i)/d(c_i, b) = w_b/w_a$. By restricting the space S to $S' \subset S$, we have that if $a, b \in S'$ then the set C' of their offspring by $CX(a, b)$ with weights w_a and w_b on the restricted space is $C' \subset C$. The properties on each of the offspring in C' defining the convex combination operator CX on the subspace S' holds because they hold on for every offspring in the superset C. ■

The product space theorem and the sub space theorems apply as well to extension ray operators. Essentially the reason is because the equality $c = CX(a, b)$ involving the convex combination CX with weights w_a, w_b and the equality $b = ER(a, c)$ involving the extension ray recombination ER with weights w_a, w_b, from a declarative point of view are equivalent, as they entail exactly the same relationship between the points a, b and c, which is, the point c is in the line between the points a and b and their distances to it are inversely proportional to their weights. The aspect in what the two operators differ is what is considered as known and what unknown. In the case of CX, a and b are known and c unknown; in the case of ER, a and c are known and b unknown. Since the theorems above do not rely on this difference, they apply to both CX and ER.

The correct practical application of these theorems may require careful understanding of the difference between declarative definition and operational definition of the recombination operators. Also, these theorems hold when distances are deterministic objects. However, in the operators defined in this paper distances are treated as stochastic objects (random variables) and distance

relationship between points are guaranteed only in expectation. Special care needs to be taken when applying these theorems on stochastic operators.

D. Convex combination and extension ray recombination for Sudoku

In this section we use the theoretical results in the previous section, to build the convex combination and extension ray recombination operators for Sudoku starting from those for permutations under swap distance. As usual, once we have these specific operators we can plug them in the formal GDE (algorithm 2) to obtain a specific GDE for the space of vectors of permutations under row-wise swap distance, hence obtaining a specific GDE for Sudoku.

The product convex combination theorem allows us to build a convex combination for an entire Sudoku grid by applying row-wise a convex combination operator defined on permutations. So, let cx a convex combination operators on permutations under swap distance, with weights w_a and w_b, and p_a, p_b, p_c be the two parent permutations and the offspring permutation respectively, i.e., $p_c = cx(p_a, p_b)$ (as the one presented in algorithm 6). By applying cx to each paired rows of sudoku grids G_a and G_b and grouping the offspring permutations in a grid G_c, we obtain a convex combination operator CX on grids under row-wise swap distance, with weights w_a and w_b.

The subspace convex combination theorem allows us to restrict the search space to the subspace of Sudoku grids in which all givens are fixed. Let us call grids belonging to this subspace feasible grids. Otherwise they are called unfeasible grids. Let us consider the operator CX' derived from the operator CX above, as follows. The operator CX' corresponds to CX, when CX is applied to feasible grids and returns feasible offspring grids. However, when CX returns unfeasible offspring grids, CX' discards them and runs CX again until feasible offspring grids are found and returned. The subspace convex combination theorem tells us that CX', like CX, is a convex combination operator under row-wise swap distance with the same weights w_a and w_b. Clearly, in practice, implementing the operator CX' in this way is inefficient. However, we can implement an operator which has behavior equivalent to CX', but that instead of discarding unfeasible offspring, it does not generate them in the first place. The operator in algorithm 6 does exactly this.

Analogously to the product convex combination theorem, the product extension ray theorem allows us to build an extension ray recombination operator on entire Sudoku grids by applying row-wise an extension ray recombination operator defined on permutations (such as the one in algorithm 7).

Analogously to the subspace convex combination theorem, the subspace extension ray theorem allows us use the original ER operator to the search the subspace of feasible grids by discarding unfeasible grids and trying again until a feasible grid is found. Let us consider the case of a single row (a permutation) rather then the entire grid. Let say we want to search the subspace obtained by fixing some of the elements in the permutation (i.e., the givens in that particular row). This can be done by using the extension ray recombination on unrestricted permutation (Algorithm 7) as a base for the above procedure to search the subspace. However, there is a subtle problem with this. This is that the recombination algorithm guarantees the wanted distance ratio only in expectation, rather than exactly all the time as assumed in the theorem, so the theorem is not necessarily applicable, as it happens in this case. Let see why and determine how to modify the Algorithm 7 to obtain the

wanted recombination operator. If one applies the procedure above and simply discards unfeasible offspring and reiterates the application of the ER operator until a feasible offspring is found, the algorithm returns an offspring that is not at the wanted expected distance. This is because the algorithm in order to achieve the wanted expected distance assumes that with some probability the fixed elements can be changed. If one prevents the fixed elements to be changed (so obtaining feasible offspring) its effect is to have less swaps applied to b to obtain c, which is, a shorter expected swap distance between b and c. To compensate for this is sufficient to increment adequately the probability p of swaps to obtain the wanted expected swap distance between b and c. This probability can be easily determined by noticing that searching a permutation subspace with permutations of size n with ng fixed elements is, in fact, equivalent to search a permutation space with permutations of size $n-np$ obtained by removing the fixed elements that can be added back when the search is over. So the probability p for the extension ray recombination operator on this space is $p = SD(p_b, p_c)/(n - ng - 1 - SD(p_a, p_b))$.

X. Experiments with GDE on Sudoku

We implemented GDE for Sudoku in the same publicly available codebase as our previous experiments on evolutionary algorithms with geometric operators and geometric particle swarm optimization [15] [16]. As our previous results define the state of the art for this problem, we chose to compare our GDE results with our previous results, and have thus not run any additional experiments for other algorithms than GDE.

The same parameter search was performed as for Sudoku as for TSP (see section VIII for details). However, instead of 20 randomly generated instances the algorithms were tested on two of the same Sudoku grids as used in our previous papers, one rated as "easy" and the other as "hard" by a Sudoku web site. For each parameter configuration, the algorithm was run 50 times. Average fitness was calculated, as well as the number of times out of 50 the run resulted in a fitness of 243, which means the grid was solved.

Algorithm	Easy 1	Hard 1
Hillclimber	35	1
GA	50	15
GPSO	36	N/A
GDE	50	13

TABLE X
NUMBER OF RUNS OUT OF 50 THE GRID WAS SOLVED; GDE COMPARED WITH OTHER SEARCH ALGORITHMS FROM PREVIOUS PAPERS, ON THE SAME TWO SUDOKU GRIDS. THE BEST CONFIGURATIONS FOUND AFTER PARAMETER TUNING ARE REPORTED FOR ALL ALGORITHMS. THE BEST GDE SETTINGS WERE POPULATION SIZE 50, F 1.0, CR 0.6 FOR EASY 1, AND POPULATION SIZE 100, F 0.0, CR 0.6 FOR HARD 1.

F/Cr	0.0	0.2	0.4	0.6	0.8	1.0
0.0	206.24	224.08	226.84	228.22	229.0	226.52
0.2	234.94	239.46	241.0	240.8	241.04	241.04
0.4	240.0	241.62	241.84	241.76	242.12	241.86
0.6	242.08	242.6	242.64	242.8	242.8	242.92
0.8	242.92	242.96	242.76	242.92	242.96	243.0
1.0	205.92	206.4	206.24	205.8	205.74	205.68

TABLE XI
PARAMETER SETTINGS AND CORRESPONDING AVERAGE FITNESS ON THE "EASY 1" SUDOKU GRID , USING POPULATION SIZE 50, WHERE THE BEST SETTING WAS FOUND. THE F PARAMETER IS ON THE HORIZONTAL AXIS, AND CR ON THE VERTICAL. HIGHER FITNESSES ARE BETTER.

From table X we can see that GDE is on par with a finely-tuned GA on both easy and hard Sudoku grids, and significantly outperforms both Geometric PSO and hill climbers. It should be noted that more extensive

tuning was performed for the GA than for GDE for this problem, as a number of different geometric crossover and mutation operators were tried; similar attention given to the GDE might improve the results further.

Table XI presents a comparison of parameter settings for GDE with population size 50 on the easy grid. We can see a general preference for high values of both parameters, though the effect is marked for Cr than for F. Additionally, extreme values of Cr (0 and 1) yield much lower performance, which is understandable as these lead to a degenerate algorithm.

XI. GDE for Genetic Programs

In order to specify the GDE algorithm to the specific space of genetic programs, we need to choose a distance between genetic programs. A natural choice of distance would be a distance (metric) associated to the Koza-style crossover [6]. This would allow us to derive the specific GDE that searches the same fitness landscape seen by this crossover operator. Unfortunately, the Koza-style crossover is provably non-geometric under any metric [14], so there is no distance associated with it[4] we can use as basis for the GDE. Another crossover operator, the homologous crossover [8] is provably geometric under structural hamming distance (SHD) [12] which is a variant of the well-known structural distance for genetic programming trees [1]. We use this distance as basis for the GDE because we will be able to use the homologous crossover as a term of reference. Notice, however, that in principle, we could choose any distance between genetic programming trees as a basis of the GDE.

[4]In the sense that there is no distance such that the offspring trees are always within the metric segment between parent trees.

A. Homologous crossover and structural hamming distance

The common region is the largest rooted region where two parent trees have the same topology. In homologous crossover [8] parent trees are aligned at the root and recombined using a crossover mask over the common region. If a node belongs to the boundary of the common region and is a function then the entire sub-tree rooted in that node is swapped with it.

The structural distance [1] is an edit distance specific to genetic programming trees. In this distance, two trees are brought to the same tree structure by adding null nodes to each tree. The cost of changing one node into another can be specified for each pair of nodes or for classes of nodes. Differences near the root have more weight. The structural hamming distance [12] is a variant of the structural distance in which when two subtrees are not comparable (roots of different arities) they are considered to be at a maximal distance. When two subtrees are comparable their distance is at most 1.

Definition 1: (Structural hamming distance (SHD))

$dist(T_1, T_2) = hd(p, q)$ if $arity(p) = arity(q) = 0$

$dist(T_1, T_2) = 1$ if $arity(p) \neq arity(q)$

$dist(T_1, T_2) = \frac{1}{m+1}(hd(p, q) + \sum_{i=1\ldots m} dist(s_i, t_i))$ if $arity(p) = arity(q) = m$

Theorem 9: Homologous crossover is a geometric crossover under SHD [12].

B. Convex combination

In the following, we first define a weighted version of the homologous crossover. Then we show that this operator is a convex combination in the space of genetic programming trees endowed with SHD. In other words, the weighted homologous crossover implements a convex combination CX in this space.

Definition 2: (Weighted homologous crossover). Let P_1 and P_2 two parent trees, and W_1 and W_2 their weights, respectively. Their offspring O is generated using a crossover mask on the common region of P_1 and P_2 such that for each position of the common region, P_1 nodes appear in the crossover mask with probability W_1, and P_2 nodes appear with probability W_2.

Theorem 10: The weighted homologous crossover is (in expectation) a convex combination in the space of genetic programming trees endowed with SHD.

Proof: The weighted homologous crossover is a special case of homologous crossover so it is also geometric under SHD. Therefore, the offspring of the weighted homologous crossover are in the segment between parents as required to be a convex combination. To complete the proof we need to show that the weights W_1 and W_2 of the weighted homologous crossover are inversely proportional to the expected distances $E[SHD(P_1, O)]$, $E[SHD(P_2, O)]$ from the parents P_1 and P_2 to their offspring O, as follows.

Given two trees P_1 and P_2, the SHD can be seen as a weighted Hamming distance on the common region of P_1 and P_2 where the weight w_i on the distance of the contribution of a position i in the common region depends on the arities of the nodes on the path from i to the root node. For each position i of the common region, the expected contribution $SHD_i(P_1, O)$ to the distance $SHD(P_1, O)$ of that specific position is directly proportional to w_i and inversely proportional to the weight W_1 (so, $E[SHD_i(P_1, O)] = w_i/W_1$). This is because, from the definition of weighted homologous crossover, W_1 is the probability that at that position the offspring O equals the parent P_1. So, the higher this probability, the smaller the expected contribution to the distance at that position. Furthermore the contribution to the distance is proportional to the weight w_i of the position i by definition of weighted Hamming distance. From the linearity of the expectation operator, we have that $E[SHD(P_1, O)] = E[\sum_i SHD_i(P_1, O)] = \sum_i E[SHD_i(P_1, O)] = \sum_i w_i/W_1 = 1/W_1$. The last passage holds true because by definition of SHD the sum of the weights on the common region equals 1 (this corresponds to the case of having two trees maximally different on the common region and their distance is 1). Analogously, for the other parent one obtains $E[SHD(P_2, O)] = 1/W_2$. This completes the proof. ■

C. Extension ray

In the following, we first define two weighted homologous recombinations. Then we show that these operators are extension ray recombinations in the space of genetic programming trees endowed with SHD. The first recombination produces offspring with the same tree structure of the second parent. The second recombination is more general and can produce offspring with tree structure different from both parents. From a geometric viewpoint, these weighted homologous recombinations implement two different versions of extension ray recombination ER in the space of genetic programming trees endowed with SHD, where the first operator produces a subset of the points produced by the second operator.

To determine a recombination that implements an extension ray operator, it is useful to think of an extension ray operator as the inverse of a convex combination operator, as follows. Given a parent P_1 (the origin of the extension ray) and the offspring C (the point the extension ray passes through), one wants to determine a parent P_2 (the point on the extension ray) such that O results from the convex combination of P_1 and P_2.

The first weighted extension ray homologous recombination is described in Algorithm 8. The second recombi-

Algorithm 8 Weighted extension ray homologous recombination 1

1: inputs: parent trees T_A (origin point of the ray) and T_B (passing through point of the ray), with corresponding weights w_{AB} and w_{BC} (both weights are between 0 and 1 and sum up to 1)
2: output: a single offspring tree T_C (a point on the extension ray beyond T_B on the ray originating in T_A and passing through T_B)
3: compute the structural Hamming distance $SHD(T_A, T_B)$ between T_A and T_B
4: set $SHD(T_B, T_C) = SHD(T_A, T_B) \cdot w_{AB}/w_{BC}$ (compute the distance between T_B and T_C using the weights)
5: set $p = SHD(T_B, T_C)/(1 - SHD(T_A, T_B))$ (the probability p of flipping nodes in the common region away from T_A and T_B beyond T_B)
6: set $T_C = T_B$
7: **for all** position i in the common region between T_A and T_B **do**
8: consider the paired nodes $T_B(i)$ and $T_A(i)$ in the common region
9: **if** $T_B(i) = T_A(i)$ and random(0,1) $\leq p$ **then**
10: set $T_C(i)$ to a random node with the same arity of $T_A(i)$ and $T_B(i)$
11: **end if**
12: **end for**
13: return string T_C as offspring

nation is the same operator as the first with the following addition before line 8 in Algorithm 8. In the common region, if two subtrees $S_A(i)$ and $S_B(i)$ coincide in structure and contents (not only if their root nodes $T_A(i)$ and $T_B(i)$ coincide), put in the corresponding position i in the offspring T_C a random subtree S_C (with in general different structure and contents from S_A and S_B). Skip the remaining nodes in the common region covered by $S_A(i)$ and $S_B(i)$.

Notice that in theory any arbitrarily large subtree S_C could be generated to be included in T_C. However, in practice its size should be limited. In the experiment, we generate S_C with the same number of nodes of S_A and S_B.

Theorem 11: The weighted extension homologous ray recombinations 1 and 2 are (in expectation) extension ray operators in the space of genetic programming trees endowed with SHD.

Proof: First we prove that $T_C = ER(T_A, T_B)$ by showing that $T_B = CX(T_A, T_C)$. Then we prove that the expected distances $E[SHD(T_A, T_B)]$ and $E[SHD(T_B, T_C)]$ are inversely proportional to the weights w_{AB} and w_{BC}, respectively.

Let us consider recombination 1. The offspring T_C has the same structure of T_B. This is because T_C was constructed starting from T_B and then for each node of the common region between T_A and T_B, T_C was not changed or it was randomly chosen but preserving the arity of that node in T_B.

The structures of the common regions $CR(T_A, T_B)$ and $CR(T_A, T_C)$ coincide. This is because the structure of the common region between two trees is only function of their structures. So, since T_B and T_C have the same structure, $CR(T_A, T_B)$ and $CR(T_A, T_C)$ have the same structure.

The tree T_B can be obtained by Homologous crossover applied to T_A and T_C (hence, $T_C = ER(T_A, T_B)$). This can be shown considering two separate cases, (i) nodes of T_B inherited from the common region $CR(T_A, T_C)$ and (ii) subtrees of T_B inherited from subtrees of T_A and T_C at the bottom of the common region. Let us consider nodes on the common region. For

each node with index i in the common region, the node $T_B(i)$ matches $T_A(i)$ or $T_C(i)$. This is true from the way $T_C(i)$ was chosen on the basis of the values of $T_A(i)$ and $T_B(i)$. We have two cases. First, $T_C(i)$ was chosen at random, when $T_A(i) = T_B(i)$. In this case $T_B(i)$ can be inherited from $T_A(i)$, since it may be $T_B(i) \neq T_C(i)$ but $T_B(i) = T_A(i)$. Second, $T_C(i)$ was chosen to equal $T_B(i)$, when $T_A(i) \neq T_B(i)$. In this case $T_B(i)$ can be inherited from $T_C(i)$. In either cases, for nodes on the common region the corresponding nodes of T_B can be inherited from T_A or T_C. The subtrees of T_B at the bottom of the common region can be inherited all from T_C (both structures and contents). Since by construction T_C inherited those subtrees from T_B without modifying them.

To show that recombination 1 is a weighted extension homologous ray recombination, we are left to show that the expected distances $E[SHD(T_A, T_B)]$ and $E[SHD(T_B, T_C)]$ are inversely proportional to the weights w_{AB} and w_{BC}. The probability p of flipping nodes in the common region away from T_A and T_B beyond T_B was chosen as an appropriate function of w_{AB} and w_{BC} and of $SHD(T_A, T_B)$ to obtain $SHD(T_B, T_C)$ such that the above requirement holds true. It is possible to prove that the chosen p is the correct one using the same argument used in the proof of theorem 10.

Let us consider now recombination 2. In this case, the offspring T_C by construction may have structure different from T_A and T_B. Also, the structures of the common regions $CR(T_A, T_B)$ and $CR(T_A, T_C)$ do not coincide. The structure of $CR(T_A, T_C)$ is covered by the structure of $CR(T_A, T_B)$ ($CR(T_A, T_C)$ is a substructure of $CR(T_A, T_B)$). The part of $CR(T_A, T_B)$ that does not cover $CR(T_A, T_C)$ comprises subtrees that are identical in structures and contents in T_A and T_B.

The tree T_B can be obtained by Homologous crossover applied to T_A and T_C (hence, $T_C = ER(T_A, T_B)$). This can be shown similarly as for recombination 1 but with an extra case to consider. Nodes of T_B corresponding to nodes in the common region $CR(T_A, T_C)$ can be inherited from T_A or T_B. The subtrees of T_B at the bottom of the common region $CR(T_A, T_C)$ can be inherited all from T_C (both structures and contents). The extra case is for the subtrees of T_B that are in the part of $CR(T_A, T_B)$ that does not cover $CR(T_A, T_C)$. These subtrees cannot be inherited from T_C, which differs form T_B by construction, but they can always be inherited from T_A.

As for the requirement on the expected distances being inversely proportional to the weights, the probability p can be chosen as the case for recombination 1 due to the recursive definition of SHD that treats nodes and subtrees uniformly. ■

Now we have operational definitions of convex combination and extension ray for the space of genetic programming trees under SHD. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the genetic programming trees space, the GDE-GP.

XII. Experiments for GP-GDE

This section reports an initial experimental analysis of the GDE-GP behavior on four standard GP benchmark problems: Symbolic Regression of the quartic polynomial, Artificial Ant on the Santa Fe trail, 5-Bit Even Parity, and 11-Bit Multiplexer. In all the experiments we used $F = 0.8$ and $Cr = 0.9$, according to [21]. Both extension ray recombinations 1 and 2 were tested, giving rise to distinct techniques we designate as GDE1 and GDE2. As a baseline for comparison we used standard GP with homologous crossover (70%) and reproduction

(30%), always applying point mutation with probability $1/L$, where L is the number of nodes of the individual. We call this baseline HGP. All the experiments were performed using populations of two different sizes (500 and 1000 individuals) initialized with the Ramped Half-and-Half procedure [6] with an initial maximum depth of 8, allowed to evolve for 50 generations. Each experiment was repeated 20 times. Statistical significance of the null hypothesis of no difference was determined with pairwise Kruskal-Wallis non-parametric ANOVAs at $p = 0.05$. A non-parametric ANOVA was used because the data is not guaranteed to follow a normal distribution. For the same reason, the median was preferred over the mean in all the evolution plots that follow. The median is also more robust to outliers.

Figure 3 shows the boxplots of the best fitness achieved along the run, using populations of 500 individuals (left column) and 1000 individuals (right column). With a population size of 500, in all four problems there is a statistically significant difference between HGP and each of the GDE-GP techniques, and no significant difference between GDE1 and GDE2. GDE-GP is consistently better than HGP, regardless of the extension ray recombination used.

It may be argued that HGP is being crippled by such a small population size, which may reduce diversity along the run. This could be true, because when doubling the population size HGP significantly improves its best fitness of run in all except the Parity problem. However, the GDE-GP techniques also show significant improvements in most cases, and remain consistently better than HGP, regardless of the extension ray recombination used, exactly as before.

However, the observation of diversity, measured as the percentage of genotypically distinct individuals in the population, revealed somewhat unexpected results.

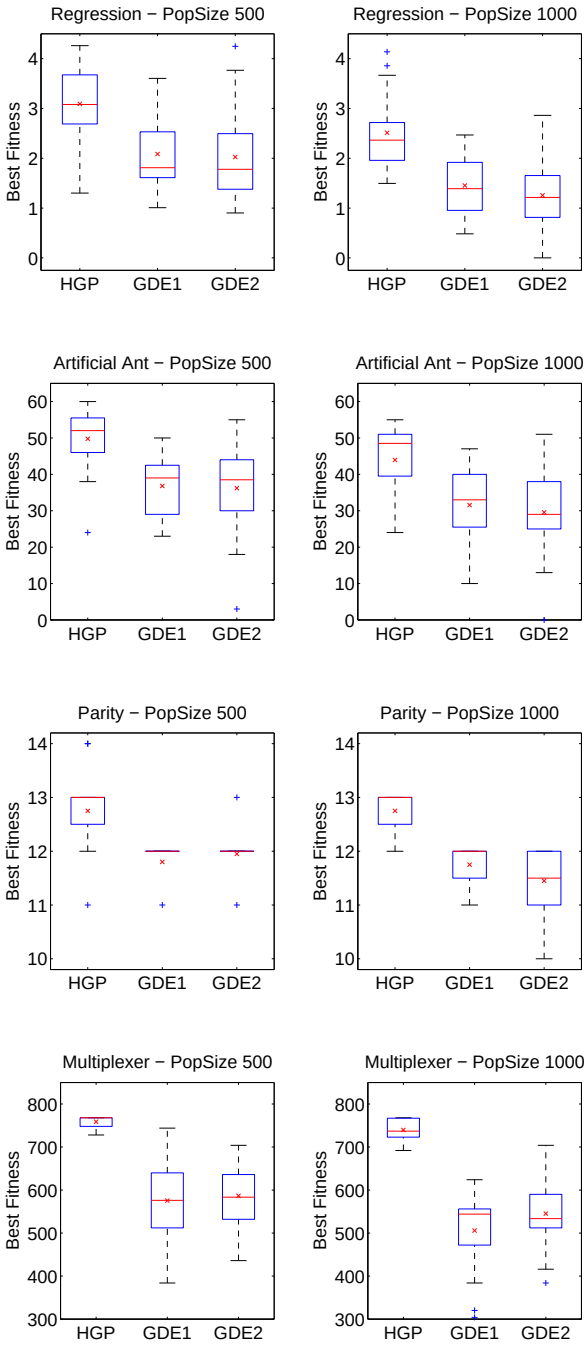

Fig. 3. Boxplots of the best fitness achieved in each problem ($\times$ marks the mean). Population sizes of 500 individuals (left column) and 1000 individuals (right column)

Figure 4 shows the evolution of the median values of diversity along the run, for both population sizes. Not only does not HGP show any clear signs of diversity loss, regardless of population size, but GDE-GP exhibits an extraordinarily varied behavior, approaching both extreme values in different problems (in Regression and Artificial Ant it practically reaches 0% while in Parity it reaches 100%), in some cases undergoing large fluctuations along the run (Multiplexer). Figure 5 shows these fluctuations on the individual runs of GDE1 on Multiplexer, with population size of 500. Also shown are the individual runs of HGP which, although also exhibiting some variation, do not present a rather undulating pattern or reach anywhere near the extreme values.

Finally, in Figure 7 we look at the evolution of the median values of average program length along the run, for both population sizes. Once again GDE-GP behaves radically differently from HGP, with both GDE1 and GDE2 presenting large but smooth fluctuations in most problems, when compared to the more constrained but somewhat erratic behavior of HGP. The most interesting case is probably the Artificial Ant, where GDE-GP quickly and steadily increases the average program length until a plateau is reached, followed by a steep decrease to very low values. Figure 6 shows the average program length of the individual runs of both HGP and GDE1 on Artificial Ant, with population size of 500.

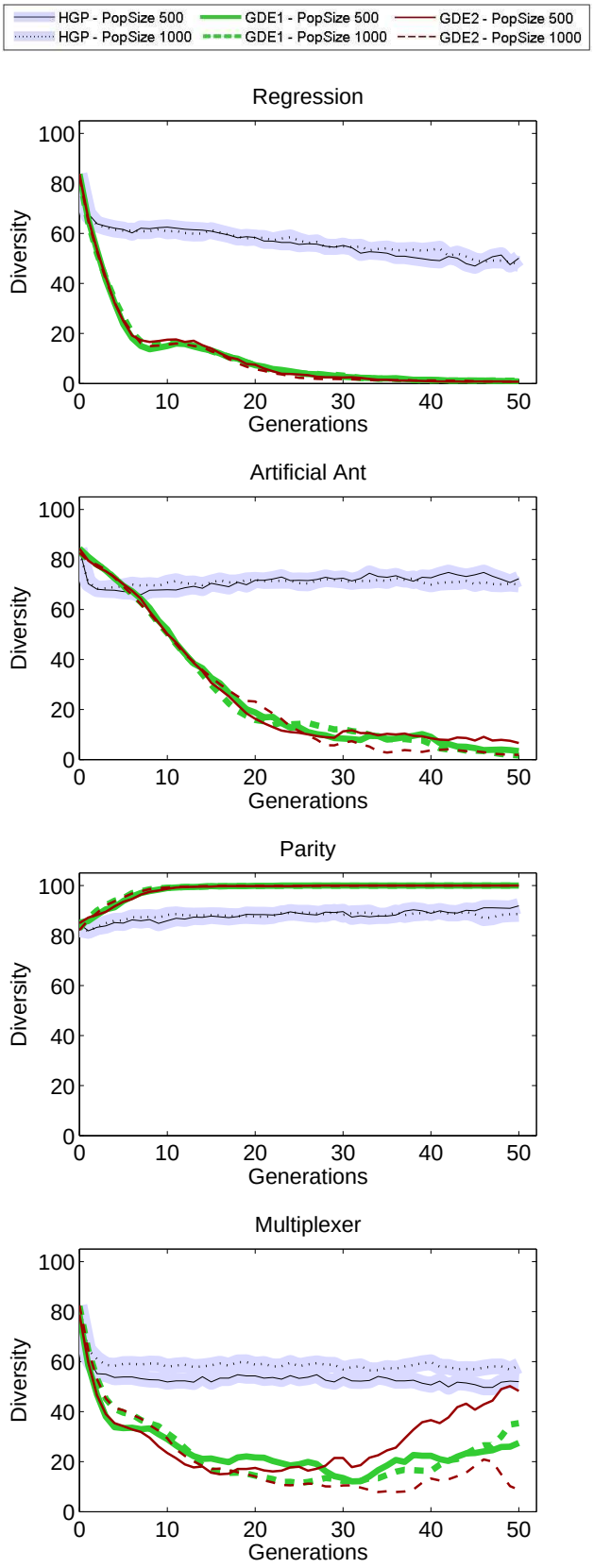

Fig. 4. Evolution of the median values of diversity in each problem

XIII. Conclusions

Geometric differential evolution is a formal generalization of DE on continuous spaces that retains the original geometric interpretation and that applies to generic combinatorial spaces. GDE can be formally specified to specific spaces associated, in principle, to any solution representation. In this article, we have illustrated that this is indeed possible in practice by deriving the

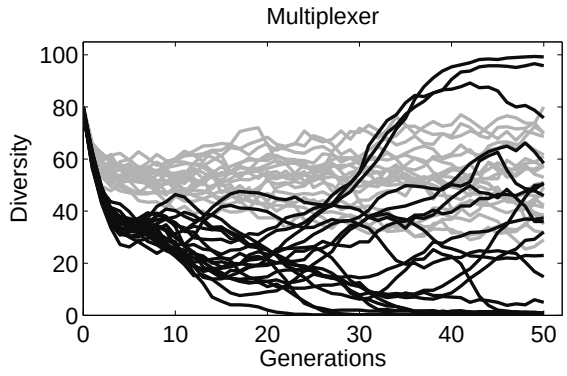

Fig. 5. Evolution of diversity on individual runs of HGP (grey) and GDE1 (black) on the Multiplexer problem, with population size 500

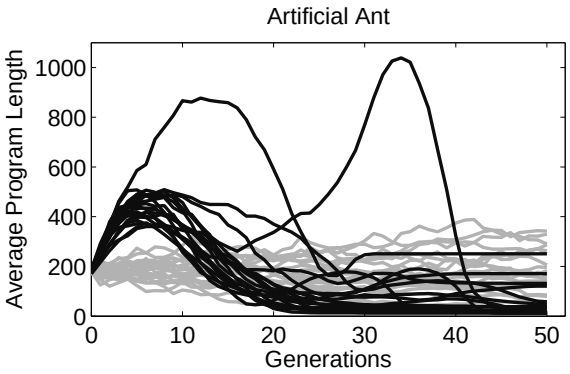

Fig. 6. Evolution of average program length on individual runs of HGP (grey) and GDE1 (black) on the Artificial Ant problem, with population size 500

specific GDEs for the Hamming space associated with binary strings, for the space of permutations endowed with the swap distance, for the space of vectors of permutations endowed with the row-swap distance, and for the space of genetic programs endowed with the structural hamming distance. These are quite different spaces based on non-trivial solution representations. The derived representation-specific GDEs are, in a strong mathematical sense, the same algorithm doing the same type of search on different spaces.

We have tested each specific GDE algorithm experimentally on standard benchmarks and compared it

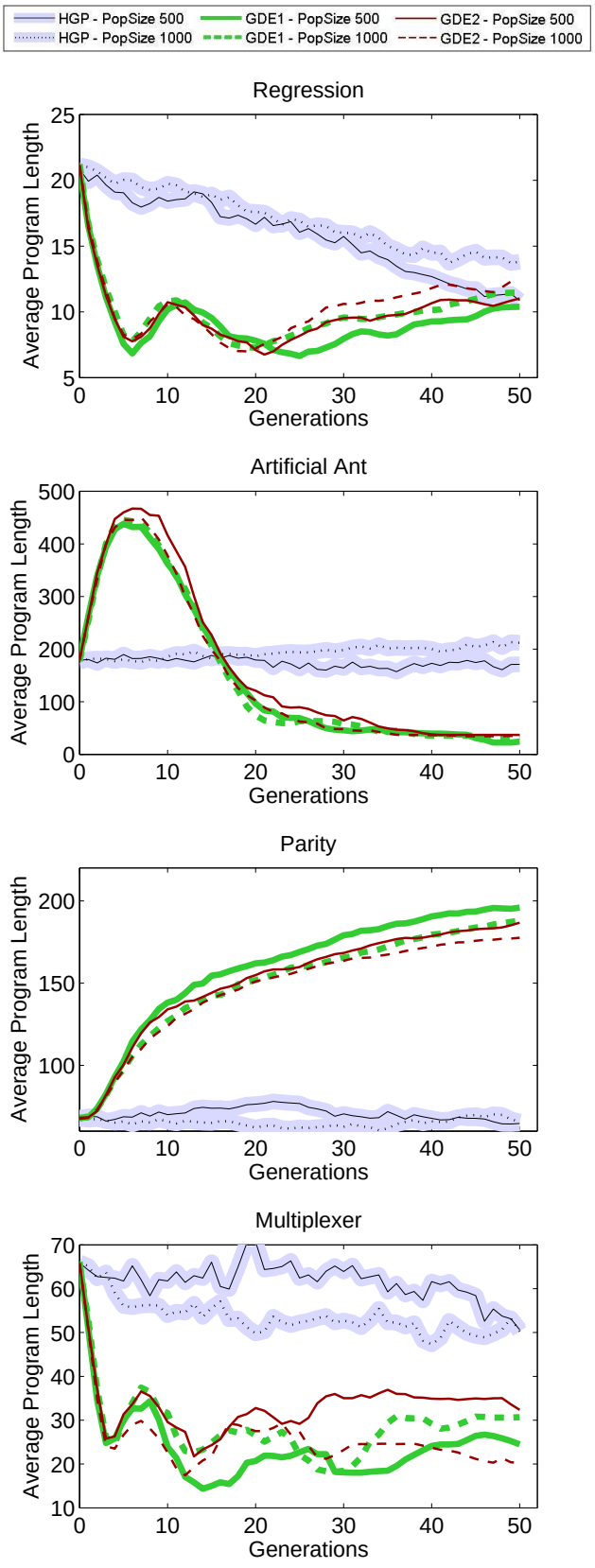

Fig. 7. Evolution of the median values of average program length in each problem

against a set of classic evolutionary algorithms defined on the same search space and representation. The binary GDE and the GP-GDE outperformed the other algorithms in the comparison. The GDEs based on permutations on the TSP did less well but the GDE on vectors of permutations on Sudoku did almost as well as a very finely-tuned GA. We believe these are very promising initial results. GDE is a very recent algorithm and further experimentation is needed to explore its potential more thoroughly.

The formal generalization methodology employed to generalize differential evolution, which is the same that was used to generalize particle swarm optimization, can be applied in principle to generalize virtually any search algorithm for continuous optimization to combinatorial spaces. Interestingly, this generalization methodology is rigorous, conceptually simple and promising as both GDE and GPSO seem to be quite good algorithms in practice. In future work, we will generalize using this methodology other classical derivation-free methods for continuous optimization that make use of geometric constructions of points to determine the next candidate solution (e.g. Nelder and Mead method and Controlled Random Search method).

ACKNOWLEDGEMENTS

We would like to thank William Spears for his assistance in adapting the code for the De Jong benchmark functions to Java. Additionally, we would like to thank Riccardo Poli for passing us the code of the Homologous Crossover for Genetic Programs. The third author also acknowledges project PTDC/EIA-CCO/103363/2008 from FCT, Portugal.

REFERENCES

[1] A. Ekart and S. Z. Nemeth, *A metric for genetic programs and fitness sharing*, Genetic Programming, Proceedings of EuroGP'2000, 2000, pp. 259–270.

[2] T. Gong and A. L. Tuson, *Differential evolution for binary encoding*, Soft Computing in Industrial Applications, Springer, 2007, pp. 251–262.

[3] S. Kauffman, *Origins of order: self-organization and selection in evolution*, Oxford University Press, 1993.

[4] J. Kennedy and R. C. Eberhart, *A discrete binary version of the particle swarm algorithm*, IEEE Transactions on Systems, Man, and Cybernetics **5** (1997), 4104–4108.

[5] ———, *Swarm intelligence*, Morgan Kaufmann, 2001.

[6] John R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, The MIT Press, 1992.

[7] Eugene F. Krause, *Taxicab geometry: An adventure in non-euclidean geometry*, Courier Dover Publications, 1986.

[8] W. Langdon and R. Poli, *Foundations of genetic programming*, Springer-Verlag, 2002.

[9] A. Moraglio, *Towards a geometric unification of evolutionary algorithms*, Ph.D. thesis, University of Essex, 2007.

[10] A. Moraglio, C. Di Chio, and R. Poli, *Geometric particle swarm optimization*, European Conference on Genetic Programming, 2007, pp. 125–136.

[11] A. Moraglio, C. Di Chio, J. Togelius, and R. Poli, *Geometric particle swarm optimization*, Journal of Artificial Evolution and Applications **2008** (2008), Article ID 143624.

[12] A. Moraglio and R. Poli, *Geometric landscape of homologous crossover for syntactic trees*, Proceedings of IEEE congress on evolutionary computation, 2005, pp. 427–434.

[13] ———, *Product geometric crossover*, Proceedings of Parallel Problem Solving from Nature conference, 2006, pp. 1018–1027.

[14] ———, *Inbreeding properties of geometric crossover and non-geometric recombinations*, Proceedings of the workshop on the Foundations of Genetic Algorithms, 2007, (to appear).

[15] A. Moraglio and J. Togelius, *Geometric pso for the sudoku puzzle*, Proceedings of the Genetic and Evolutionary Computation Conference, 2007, pp. 118–125.

[16] A. Moraglio, J. Togelius, and S. Lucas, *Product geometric crossover and the sudoku puzzle*, Proceedings of IEEE congress on evolutionary computation, 2006, pp. 470–476.

[17] Alberto Moraglio and Julian Togelius, *Geometric differential evolution*, Proceedings of the 11th Annual conference on Genetic and evolutionary computation, 2009, pp. 1705–1712.

[18] Michael O'Neill and Anthony Brabazon, *Grammatical differential evolution*, Proceedings of the 2006 International Conference on Artificial Intelligence, CSREA Press, 2006, pp. 231–236.

[19] G. C. Onwubolu and D. Davendra (eds.), *Differential evolution: A handbook for global permutation-based combinatorial optimization*, Springer, 2009.

[20] G. Pampara, A.P. Engelbrecht, and N. Franken, *Binary differential evolution*, IEEE Congress on Evolutionary Computation, 2006.

[21] K. V. Price, R. M. Storm, and J. A. Lampinen, *Differential evolution: A practical approach to global optimization*, Springer, 2005.

[22] G. Sywerda, *Uniform crossover in genetic algorithms*, Proceedings of the third international conference on Genetic algorithms, 1989.

[23] Julian Togelius, Renzo De Nardi, and Alberto Moraglio, *Geometric pso + gp = particle swarm programming*, Proceedings of the Congress on Evolutionary Comptutation (CEC), 2008.

[24] T. Yato, *Complexity and completeness of finding another solution and its application to puzzles*, Master's thesis, University of Tokyo, Japan, 2003.