Geometric PSO + GP = Particle Swarm Programming

Julian Togelius
IDSIA
USI/SUPSI, Galleria 2,
6928 Manno-Lugano, Switzerland
julian@idsia.ch

Renzo De Nardi
Dept. Computing and Electronic Systems
University of Essex
Colchester CO4 3SQ, UK
rdenar@essex.ac.uk

Alberto Moraglio
Dept. Informatics Engineering
University of Coimbra
3030-290 Coimbra, Portugal
moraglio@dei.uc.pt

***Abstract*—Geometric particle swarm optimization (GPSO) is a recently introduced formal generalization of traditional particle swarm optimization (PSO) that applies naturally to both continuous and combinatorial spaces. In this paper we apply GPSO to the space of genetic programs represented as expression trees, uniting the paradigms of genetic programming and particle swarm optimization. The result is a particle swarm flying through the space of genetic programs. We present initial experimental results for our new algorithm.**

I. Introduction

Particle Swarm Optimization (PSO) is a relatively recently devised population-based stochastic global optimization algorithm [5]. PSO has many similarities with evolutionary algorithms, and has also proven to have robust performance over a variety of difficult optimization problems. However, the original formulation of PSO requires the search space to be continuous and the individuals to be represented as vectors of real numbers.

There are a number of extensions of PSO to combinatorial spaces with various degrees of success [4] [1]. However, for every new solution representation, the PSO algorithm needs to be rethought and adapted to the new representation. Alternatively, the problem domain has to be "shoehorned" into a representation that the PSO algorithms can handle natively.

Geometric Particle Swarm Optimization (GPSO) is a very recently devised generic extension of PSO to almost any search space [9]. The requirements for GPSO to work in a given space is that there is a way of measuring the distance between two points (solutions), that there is a mutation operator that stochastically perturbs a point, and that there is a weighted geometrical crossover operator that given two parent points produces an offspring that lies between them. In a first application to GPSO to discrete spaces, it was shown to perform satisfactorily on the problem of finding solutions to Sudoku puzzles [15].

In this paper, we apply the GPSO algorithm to developing computer programs, represented as expression trees. Using a genetic algorithm to evolve expression trees is usually called Genetic Programming (GP); similarly, we will refer to the application of GPSO to expression trees as Particle Swarm Programming (PSP). The main purpose of this paper is to show that this is at all possible, thus opening up for using alternative optimization algorithms for automatic program generation, and for taking PSO beyond mere optimization.

The only previous combination of PSO and GP we know of works through representing the genetic programs indirectly, and using a developmental process to grow the phenotype (an expression tree) out of the genotype (a vector of integers) [16]. This way, a standard PSO algorithm can be used, but at the cost of the reconstitution of the search space inherent to an indirect representation. In contrast, the algorithm proposed in this paper works directly on expression trees, using a principled, almost representation-indepent, reformulation of the PSO algorithm.

The paper is structured as follows: first we describe the GPSO algorithm. Then we present three different weighted crossover operators for expression trees and discuss their relative merits. This is followed by the experimental section, where we present results for PSP on two GP benchmarks: the Santa Fe Ant Trail and a symbolic regression problem. We finish with a discussion of our results and directions for future research.

II. Geometric Particle Swarm Optimization

In order to theoretically derive the GPSO algorithm, we first need to define the concepts of geometric crossover and multi-parent geometric crossover, and use these to define the concept of a convex geometric combination in a metric space. This is done in the following sections. The algorithm itself, which can be used quite independently of its derivation, is found in section II-D.

A. Geometric crossover

Geometric operators [10] are search operators defined in geometric terms using the notions of line segment and ball. These notions and the corresponding genetic operators are well-defined once a notion of distance (metric) in the search space is defined.

In a metric space (S, d), a *closed ball* is a set of the form $B(x; r) = \{y \in S | d(x, y) \leq r\}$ where $x \in S$ and r is a positive real number called the radius of the ball. A *line 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 extremes of the segment. Metric ball and metric segment generalise the

familiar notions of ball and segment in the Euclidean space to any metric space through distance redefinition.

Definition 1: A binary operator is a geometric crossover under the metric d if each offspring lies in the segment between its parents.

The definition is *representation-independent* and, therefore, crossover is well-defined for any representation. Being based on the notion of metric segment, *crossover is only function of the metric d* associated with the search space.

This class of operators is really broad. For vectors of reals, various types of blend or line crossovers, box recombinations, and discrete recombinations are geometric crossovers [10]. For binary and multary strings, all homologous crossovers are geometric [10] [12]. For permutations, PMX, Cycle crossover, merge crossover and others are geometric crossovers [14]. For syntactic trees, the family of homologous crossovers are geometric [11]. Recombinations for several more complex representations are also geometric [12].

B. Multi-parent geometric crossover

To extend the geometric crossover to the case of multiple parents we need the notions of metric convex set and metric convex hull.

A set is a metric convex set if for every pair of points within the set, every point in the metric segment that joins them is also within the set.

The metric convex hull of a set of point P is the smallest metric convex set that includes all points in P. For example, in the Euclidean case, when the set P contains two points the convex hull of P is the segment whose extremes are the points in P. When the set P contains three points the convex hull of P is the triangle whose vertices are the points in P.

Definition 2: (Multi-parental geometric crossover) In a multi-parental geometric crossover, given n parents $p_1, p_2, \ldots, p_n$ their offsprings are contained in the metric convex hull of the parents $\mathcal{C}(\{p_1, p_2, \ldots, p_n\})$ for some metric d

Theorem 1: (Decomposable three-parent recombination) Every multi-parental recombination $RX(p_1, p_2, p_3)$ that can be decomposed as a sequence of 2-parental geometric crossovers under the same metric, GX and GX', so that $RX(p_1, p_2, p_3) = GX(GX'(p_1, p_2), p_3)$, is a three-parental geometric crossover

C. Convex combination in metric spaces

In order to define PSO for a generic metric space, we need to extend the notion of convex combination to generic metric spaces.

For the Euclidean space, a *convex combination* is a linear combination of vectors where all coefficients are non-negative and sum up to 1. It is called "convex combination", since, when a vector represents a point in space, all possible convex combinations (given the base vectors) will be within the convex hull of the given points. In fact, the set of all convex combinations constitutes the convex hull.

The weight of a point in a convex combination can be seen as a measure of relative linear attraction toward its corresponding point versus attractions toward the other points of the combination. The closer the weight to one, the stronger the attraction to its corresponding point. The resulting point of the convex combination can be seen as a weighted spatial average and it is the equilibrium point of all the attraction forces. The distance between the equilibrium point and a point of the convex combination is therefore a decreasing function of the level of attraction (weight) of the point: the stronger the attraction, the smaller its distance to the equilibrium point. This observation can be used to reinterpret the weights of a convex combination in a metric space as follows: $y = w_1x_1 + w_2x_2 + w_3x_3$ with w_1, w_2 and w_3 greater than zero and $w_1 + w_2 + w_3 = 1$ is generalized to $d(x_1, y) \sim 1/w_1$, $d(x_2, y) \sim 1/w_2$ and $d(x_3, y) \sim 1/w_3$.

This definition is formal and valid for all metric spaces but it is non-constructive. A convex combination for the Euclidean space, not only defines a convex hull, but it tells also how to reach all its points. For convex combinations based on combinatorial spaces, how to reach the points in the convex hull is not obvious. Weighted multi-parental geometric crossovers can be used to pick points specified by convex combinations in combinatorial spaces.

For the Euclidean space, a weighted three-parental geometric crossover can be actually decomposed into two sequential applications of weighted two-geometric crossover: $\Delta GX((a, w_a), (b, w_b), (c, w_c)) = GX((GX((a, \frac{w_a}{w_a+w_b}), (b, \frac{w_b}{w_a+w_b})), w_a + w_b), (c, w_c))$. This formula can be used as a rule of thumb to build weighted three parental geometric crossovers from weighted bi-parental geometric crossovers for any solution representations.

D. The algorithm

Algorithm 1 Standard PSO algorithm

1: **for all** particle i **do**
2: initialise position $x_i \in U[\mathbf{a}, \mathbf{b}]$ and velocity $v_i = \mathbf{0}$
3: **end for**
4: **while** stop criteria not met **do**
5: **for all** particle i **do**
6: set personal best $\hat{x}_i$ as best position found so far from the particle
7: set global best $\hat{g}$ as best position found so far from the whole swarm
8: **end for**
9: **for all** particle i **do**
10: update velocity using equation $v_i(t+1) = \omega v_i(t) + \phi_1 R_1(\hat{g}(t) - x_i(t)) + \phi_2 R_2(\hat{x}_i(t) - x_i(t))$
11: update position using equation $x_i(t+1) = x_i(t) + v_i(t+1)$
12: **end for**
13: **end while**

The main feature of the canonical PSO (Algorithm 1) that allows the motion of particles is the ability to perform linear combinations of points in the search space. To obtain a generalisation of PSO to generic search spaces, we can achieve this same ability by using multiple (geometric) crossover operations.

Theorem 2: In a PSO with no momentum ($\omega = 0$) and where learning rates are such that $\phi_1 + \phi_2 < 1$, the future position of each particle x' is within the triangle formed by its current position x, its local best $\hat{x}$ and the swarm best $\hat{g}$. Furthermore, x' can be expressed without involving the particle's velocity as $x' = (1 - w_2 - w_3)x + w_2\hat{x} + w_3\hat{g}$.

The generic Geometric PSO algorithm is illustrated in Algorithm 2.

Algorithm 2 Geometric PSO algorithm

for all particle i **do**
  initialise position x_i at random in the search space
end for
while stop criteria not met **do**
  for all particle i **do**
    set personal best $\hat{x}_i$ as best position found so far by the particle
    set global best $\hat{g}$ as best position found so far by the whole swarm
  end for
  for all particle i **do**
    update position using a randomized convex combination
    $x_i = CX((x_i, \omega), (\hat{g}, \phi_1), (\hat{x}_i, \phi_2))$
    mutate x_i
  end for
end while

This differs from the standard PSO (Algorithm 1) in that: there is no explicit velocity, the equation of position update is the convex combination (CX), there is mutation and the parameters ω, ϕ_1, and ϕ_2 are positive and sum up to one.

III. Weighted crossover operators for expression trees

As detailed in the above section, the GPSO algorithm relies on a weighted crossover operator, which must be defined separately for each type of search space. As far as we are aware, no studies on weighted crossover for expression trees have so far been published. We have therefore modified three previously well-known crossover operators, two of which are geometric, to create three different weighted crossover operators for expression trees. They all operate according to the simple assumption that the relative influence of two parent trees on an offspring is proportional to the relative number of nodes they contribute to the offspring. In the descriptions below, when we choose a node randomly among a set of nodes, all nodes in the set have the same probability of being chosen.

A. Weighted subtree swap

The weighted subtree swap is a modification of the operator invented in [2] and popularized in [6], which is also the perhaps most intuitive and probably most commonly used crossover operator for expression trees. In the unweighted subtree swap, one node is chosen randomly in each parent tree (usually different nodes in the two trees), and the subtrees rooted at these nodes are exchanged. Weighted subtree swap chooses nodes at depths proportional to $(1-w)$, where w is the weight of the second parent. To take an example, if the first tree has depth 5, the second tree depth 10, and the weight of the second tree is 0.2, the crossover point is chosen (proportionally to $1 - 0.2 = 0.8$) at depth 4 in the first tree and depth 8 in the second tree. The only offspring that is returned is the one based on the root from the first tree.

As subtree swap was proved to be non-geometric in [13], this crossover does not strictly satisfy the requirements laid out in the definition of GPSO, but is included because of its simplicity.

B. Homologous crossover and one-point crossover

The common region is the largest rooted region where two parent trees have the same topology. In homologous crossover [7] 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. One special case of homologous crossover is one-point crossover in which a common crossover point is picked randomly from the nodes belonging to the common region and then the two sub-trees rooted at the crossover point are swapped.

In [3] an edit distance specific to syntactic trees, structural distance, was defined. 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. The differences near the root have more weight. The normalized structural hamming distance (SHD) for trees [11] is a variation of the structural distance.

The hyperschema [7] associated with two trees is the tree structure that has the topology of the common region of the two trees; its nodes are '=' when two matched nodes differ in the content, or '#' replacing two subtrees whose roots are matched but their arities differ, or any other content when it is the same in both matched nodes. Figure 1 illustrates at the top, two parent trees P1 and P2; at the bottom left, their associated hyperschema H(P1,P2) at the bottom right, all the potential offspring applying homologous crossover to parents P1 and P2 (the part in bold means alternative content of the tree; in this case there are 5 independent binary alternatives, resulting in 32 possible offspring). The SHD distance between two trees is a function of the hyperschema associated with the two trees and not directly of the two trees (see figure 2).

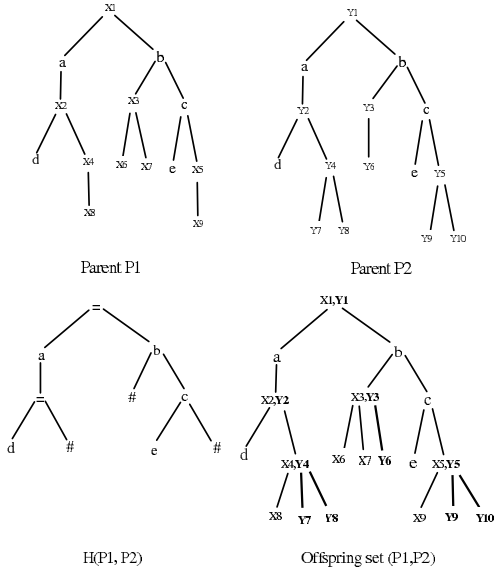

Fig. 1. Hyperschema and offspring set

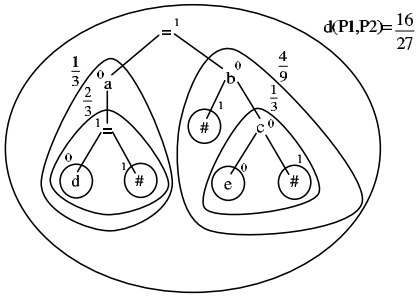

Fig. 2. Hyperschema and structural distance

Theorem 3: Homologous crossover and one-point crossover are geometric under SHD.

C. Weighted homologous crossover

Definition 3: (Weighted homologous crossover). Let P_1 and P_2 two parents 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 as for each position of the common region, P_1 appears in the crossover mask with probability W_1, and P_2 with probability W_2.

Theorem 4: (Coherence between weights and distances). 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 are decreasing functions of the weights W_1 and W_2, respectively. So, if $W_1 < W_2$ then $E(SHD(P_1, O)) > E(SHD(P_2, O))$.

Proof:

For each position of the common region of P_1 and P_2, the expected contribution to the distance $SHD(P_1, O)$ of that specific position is a decreasing function of the weight W_1. This is because 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. From the linearity of the expectation and from the definition of SHD, the expected distance $E(SHD(P_1, O))$ is a linear combination with positive weights of the expected contributions to the distance $SHD(P_1, O)$. Since all these contributions are decreasing functions of W_1, also any linear combination with positive weights is a decreasing function of W_1. So, $E(SHD(P_1, O))$ is a decreasing function of W_1. ■

D. Weighted one-point crossover

Definition 4: (Weighted one-point crossover)

Let A and B be two parent trees. A is the donor, from which a subtree is taken, and B is the recipient to which the subtree is given. Only one offspring O is produced, which consists of the parent B with the new subtree taken from A. Let w_a and w_b be the weights associated with A and B, such that $w_a + w_b = 1$. The crossover point is chosen in a way that the smaller the value of w_b, the closer to the root the crossover point.

In the following we discuss the coherence between distances and weights for one-point crossover.

Let D be the donor tree and R be the recipient tree, with non-negative weights w_d and w_r such that $w_d + w_r = 1$. We can interpret the weights of the parents as the relative contribution of genetic material of the parents in the offspring. So, in the offspring tree with N nodes, $N_d = w_d * N$ has to come from parent D and $N_r = w_r * N$ has to come from parent R.

A rough way of approximating this would be using the weights to determine the depths of the trees at which to perform the crossover cut. For the donor tree, the larger the weight w_d the closer to the root the crossover cut has to be. In this way the size of the subtree extracted from D is roughly an increasing function of the weight w_d. For the recipient tree, it has to be the other way around, the smaller the weight w_r the closer to the root the crossover cut. In this way the size of the subtree removed from R is roughly a decreasing function of the weight w_r. So, the part of the tree R that is going to be part of the offspring is roughly an increasing function of the w_r. Since $w_r = 1 - w_d$, the previous rule can be restated as follows. For both trees D and R, the larger the weight w_d of the donor tree the closer to the root the crossover cut has to be.

We could then extend the weighted recombination of two trees to the weighted recombination of three trees by doing two sequential weighted recombinations and using the formula proposed in the original geometric PSO paper to pass from the weights of a combination of 3 points to the weights of the equivalent two sequential combinations of two points.

However, because of the asymmetry of treatment of parent A and B the same number assigned to w_a and w_b is likely to have a different impact on the equilibrium point. To remove this potential bias, we assign randomly the roles of donor and recipient every time to the objects to recombine. So, for example, when one recombines global best and particle best, we pick at random the role to assign to these two objects. We do not fix, for example, global best as donor.

E. Small common regions and fixed-size populations

During initial testing of the operators, we discovered the problem that for two arbitrarily chosen trees in a population, the common region is likely to be very small. This leads to all crossovers based on the common region choosing crossover points close to the root, and thus that the weights associated with the crossover have little effect, and that most crossover operations become very destructive.

One way of dealing with this problem is to ensure that all trees in the population have the same size and topology (they can of course still have different nodes). The easiest way to do this is to require that all nonterminals have the same arity, and then design the initialization and mutation operators so as to ensure that all trees always are completely expanded up to a certain depth, i.e. that all nodes at depths up to the maximum depth are nonterminals and all nodes at the maximum depth are terminals.

A possible problem with the fixed-size approach is that not all good solutions to a particular problem using a particular function set might be expressible using this particular tree topology, and that even if they are, constraining the population to fixed-size trees might alter the structure of the search space in unforeseen ways. The effects of the fixed-size constraint on different search spaces will, as we see it, most easily be estimated through direct experiments.

Alternatively, a way of making all functions have the same arity without altering the semantic of the function (but potentially inserting a lot of neutrality in the search) is to consider all functions to have the arity of the function with the largest arity in the set of functionals and discarding the extra inputs due to the larger arity of the function. For example, if you have a the functional set $\{+, *, sin\}$ the maximum arity is 2 (which is the arity of both $+$ and $*$). The arity of sin is 1. We can then artificially extend the sin function to arity two by defining $sin(x, y) = sin(x)$ so the input y is simply discarded. In this spirit, we could allow for constants occuring not only at the bottom of the tree, but anywhere in the tree by defining the constant function $k(x, y) = k$ where k is a real number for example. So, the arguments of a constant function are do not affect its output at all. This would allow us to express any tree whose topology is covered by the fixed size tree considered, without forcing all solutions of the problem to necessarily be full trees. This approach was not taken in the experiments in this paper; here we have decided to restrict ourselves to functions of arity two, allowing terminals only as leaves of the tree.

IV. Experiments

In all experiments, we used the *Regrow* mutation, which randomly selects a point in the tree and re-grows the tree from that point in the same way as the uniform initialization does, so that all branches reach maximum depth. This preserves the fixed size of the trees in the population (if combined with homologous or one-point crossover).

For both problem domains, we compared GPSO, using the three different weighted crossover operators and under a number of different settings for the three main parameters, with random search and genetic programming. For each algorithm, what counted was the maximum fitness resulting from allowing the algorithm to make 50000 tree evaluations, averaged over a number of separate runs. In all cases, trees were initialized using uniform growth to depth 7, so that all trees of the initial population of all experiments were complete and had 64 terminals and 64 nonterminals. We used the lattice neighbourhood topology, which has previously been shown to work well with GPSO [15].

We tested two versions of random search. The first form of random search (*initialization search*) consisted in simply creating 50000 different trees using uniform growth initialization, evaluating them and selecting the highest fitness of any of these trees. This was repeated 100 times, and the mean *highest fitness* and standard deviation the highest fitnesses was recorded. (Note that this is the mean of the 100 trials, not the mean of the 50000 initializations, which was always considerably lower.) The second form of random search (*mutation search*) starts with a tree initialized through uniform growth, and then applies 50000 Macro mutations to this tree. Mean highest fitness over 100 repetitions of this experiment was recorded, as was the standard deviation of the highest fitnesses.

For a comparison, we chose the deterministic crowding implementation of genetic programming as defined in [8]. In deterministic crowding, two elements of the population are randomly paired and (with a certain probability) recombined to produce two offsprings. The offsprings are mutated and then, after the evaluation, they replace their most similar parent if they have equal or higher fitness, are discarded otherwise. In our implementation the SHD distance was used to evaluate the similarity between parent and offspring. We tried all the three crossover operators in their non-weighted form together with this algorithm. For each crossover type, the mutation and crossover rate were varied respectively within the set $\{0.3, 0.6, 1\}$ and $\{0.1, 0.3, 0.6, 1\}$.

Being the subject of this paper, the geometric PSO was the algorithm whose parameters we tested most extensively. For each problem domain, all three of the above defined weighted crossover operators were tested, and for each crossover operator a range of settings for the three main parameters of the GPSO algorithm were tried. Settings of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 were tried for both the inertia and the memory parameters; for the sociality parameter, the setting was $1 - inertia - memory$ (negative values of the sociality parameter led to that configuration not being tested). The mutation ratio was fixed at 0.1. For each configuration, 20 independent runs were made, and the average and standard deviation of the highest fitnesses of the last update of each of the runs were recorded. The runs had a population size of 100 and lasted for 500 updates (generations), meaning that each run performed 50000 tree evaluations.

A. Santa Fe Ant Trail

The Santa Fe Ant Trail, a classic and rather hard GP Benchmark problem, was introduced in [6]; we have no space

for a full description here. Basically the target is to steer an ant along a trail in a grid world, eating all the food, in a limited number of time steps. The minimum fitness is 0 (no food eaten) and the maximum is 89 (all food eaten). In the original formulation, the allowed nonterminals were IfFoodAhead (evaluates its left child if there is food immediately ahead of the agent, and its right otherwise), Progn2 (evaluates both children in succession) and Progn3 (has three children, evaluates all of them in succession). In all experiments in this paper, the Progn3 nonterminal has been eliminated in order to ensure that all nonterminals have the same arity, making fixed-size populations possible. Both in the original formulation and in this paper, the terminal set consists of Left, Right and Move.

1) Random search: Using initialization search, the mean *highest fitness* was 57.7 (standard deviation 6.6). (The mean fitness of any particular randomly initialized tree was around 1 or 2.) Using mutation search, the mean highest fitness was 54.1 (9.4). None of the random searches ever reached the global optimum of 89.

2) Genetic programming: With subtree swap, the deterministic crowding algorithm worked best with mutation rate 1.0 and crossover rate 0.6, reaching an average highest fitness (AHF) of 76.7 (standard deviation 14.2). One-point crossover, with mutation 0.3 and crossover 1.0, allowed the algorithm to reach AHF 65.7 (12.6); using homologous crossover, the best configuration was mutation 1.0 and crossover 0.3, reaching fitness 66.1 (12.83). Global optima were only reached in the experiments that used subtree swap.

3) Geometric PSO: The results of the GPSO runs were as follows: For weighted subtree swap, the best configuration of parameters found was inertia 0.2, memory 0.4 and sociality 0.4. Using this configuration, the AHF was 77.9 (standard deviation 10.5), and a global optimum was reached 4 times out of 20. For homologous and one-point crossover results were worse. For one-point crossover, the best configuration found was inertia 0.2, memory 0.6 and sociality 0.2, yielding AHF 60.1 (10.2) and reaching global optimum one time out of 20. For homologous crossover the best configuration was inertia 0.2, memory 0.8 and sociality 0.0, bringing the AHF to 69.9 (9.5).

B. Symbolic Regression

The second benchmark, also a classic problem in the GP literature, is symbolic regression. The well known sextic polynomial function

$$y = x^6 - 2x^4 + x^2 \tag{1}$$

was used in an experimental setup very similar to the one introduced in [6].

As in the original formulation the only allowed functionals are $\{+, -, *, \%\}$ while the terminals are the input x or a constant value that is drawn (at the creation of the tree) from a uniform distribution in the interval $[-1, 1]$. All the functions used have the same arity (two), fulfilling the requirement for a fixed-size population discussed in III-E. The fitness is evaluated in 50 points randomly drawn with uniform distribution from the interval $[-1, 1]$. If in the selected point the absolute error between the true value of the function and the value returned by the individual under evaluation is lower than 0.01, the point is considered a hit, a miss otherwise. The problems becomes a maximization one in which the fitness of the individual is simply the total number of hits, its value is therefore between 0 (no fitting at all) and 50 (good fitting).

1) Random search: Using initialization search, the mean highest fitness was 18 (standard deviation of 2.6). With mutation search, the mean highest fitness was as well 18 (2.5). The average fitness for randomly created trees was 1.27 (2.7). None of the random search methods reached the maximum fitness.

2) Genetic programming: The best configuration found for the deterministic crowding algorithm with subtree swap was where both mutation and crossover were set to 1; this resulted in an AHF of 38.25 (standard deviation 6.95). With homologous crossover the best configuration was mutation 0.3 and crossover 1.0, which yielded an AHF of 35.35 (6.2). One-point crossover performed best with both mutation and crossover set to 1.0; AHF 35.6 (10.08). Those were the only settings that gave maximum fitness, twice in 20 repetitions.

The results obtained in [6] are better than those presented here, possibly due to the population size we used. However, for fairness of comparison we used the same population size for both PSP and GP.

3) Geometric PSO: For weighted subtree swap, the best configuration of parameters found was inertia 0.8, memory 0.2 and sociality 0.0; the AHF was 33.4 (standard deviation 5.66). For one-point crossover, the best configuration was inertia 0.2, memory 0.6 and sociality 0.2, yielding interestingly the same AHF 33.4 (6.1) which was obtained with subtree swap. Homologous crossover worked best with inertia 0.4, memory 0.6 and sociality 0.2, with an AHF 28.55 (4.22). On a few occasions, using subtree swap, global optima were reached.

V. Conclusion

Geometric PSO is a generalization of the classical PSO to general metric spaces. In particular, it applies to combinatorial spaces. In this paper we have demonstrated the application of the GPSO algorithm to the space of GP expression trees, inventing PSP. We believe this to be the first algorithm to use PSO for directly represented expression trees. Our initial experiments show that it performs about as well as a sophisticated genetic algorithm on two standard benchmarks, though better results on these benchmarks can be found in the literature.

We have also defined and compared three weighted crossover operators, with weighted subtree swap turning out to perform best. However, significant work remains in finding out which operators work best for PSP, and on what sort of landscapes PSP works best. In particular, for the common region-based crossover operators to work well, we need to find a way of maximizing common regions without the adverse effects of fixed-size populations.

As the two classic benchmarks have been subject to much experimentation in numerous papers over the years, it would be preposterous to think that we would beat all other algorithms with the first implementation of a new algorithm. Maybe further refinement of the operators will lead to a PSP algorithm that excels in these benchmarks, but it is also possible that PSO is a less efficient algorithm for GP than the GA's currently employed are; this in itself would be an interesting result and merit further analysis. In any case, the point of this paper is not beat the benchmarks but the proof of conecpt. Above, we provide an initial implementation that shows that it is possible to use a theoretically sound form of PSO, operating directly in the space of expression trees, for GP. We believe that these results open up both for the use of particle swarm optimization in a variety of new spaces, and for the development of a family of new algorithms to complement the genetic algorithm in GP.

VI. Acknowledgement

The third author would like to thank Stefano Cagnoni for his helpful comments and suggestions on the initial concept of PSP presented in this paper.

References

[1] M. Clerc, *Discrete particle swarm optimization, illustrated by the traveling salesman problem*, New Optimization Techniques in Engineering, Springer, 2004, pp. 219–239.

[2] N. L. Cramer, *A representation for the adaptive generation of simple sequential programs*, Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985.

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

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

[8] S. W. Mahfoud, *Niching methods for genetic algorithms*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1995.

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

[10] A. Moraglio and R. Poli, *Topological interpretation of crossover*, Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 1377–1388.

[11] ———, *Geometric landscape of homologous crossover for syntactic trees*, Proceedings of IEEE congress on evolutionary computation, 2005, pp. 427–434.

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

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

[14] ———, *Topological crossover for the permutation representation*, Journal of the Italian Association for Artificial Intelligence (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] M. O'Neill and A. Brabazon, *Grammatical swarm*, Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 163–174.