An Introduction to Geometric Semantic Genetic Programming

Leonardo Vanneschi

Abstract For all supervised learning problems, where the quality of solutions is measured by a distance between target and output values (error), geometric semantic operators of genetic programming induce an error surface characterized by the absence of locally suboptimal solutions (unimodal error surface). So, genetic programming that uses geometric semantic operators, called geometric semantic genetic programming, has a potential advantage in terms of evolvability compared to many existing computational methods. This fosters geometric semantic genetic programming as a possible new state-of-the-art machine learning methodology. Nevertheless, research in geometric semantic genetic programming is still much in demand. This chapter is oriented to researchers and students that are not familiar with geometric semantic genetic programming, and are willing to contribute to this exciting and promising field. The main objective of this chapter is explaining why the error surface induced by geometric semantic operators is unimodal, and why this fact is important. Furthermore, the chapter stimulates the reader by showing some promising applicative results that have been obtained so far. The reader will also discover that some properties of geometric semantic operators may help limiting overfitting, bestowing on genetic programming a very interesting generalization ability. Finally, the chapter suggests further reading and discusses open issues of geometric semantic genetic programming.

L. Vanneschi (✉)
NOVA IMS, Universidade Nova de Lisboa, 1070-312 Lisboa, Portugal
e-mail: lvanneschi@novaims.unl.pt

O. Schütze et al. (eds.), *NEO 2015*, Studies in Computational Intelligence 663, DOI 10.1007/978-3-319-44003-3_1

1 Introduction

In the last 15 years, several studies [41, 45, 46] have pointed out the importance of *fitness landscapes*[1] for understanding the difficulty of a problem for Genetic Programming (GP) [25, 40]. In 2012, new genetic operators for GP were introduced, called geometric semantic operators [32], inducing a fitness landscape characterized by the absence of locally suboptimal solutions (unimodal fitness landscape) for any supervised learning problem where fitness is a distance (or error) between a set of target values and the corresponding set of calculated outputs (this is the typical situation, for instance, of classification or regression problems). GP using these operators was called Geometric Semantic GP (GSGP). Since its introduction, GSGP has raised a remarkable interest of GP researchers, probably attracted by the potential advantage in terms of evolvability [46] given by a unimodal fitness landscape. At the same time, researchers have tried to counteract an important drawback of geometric semantic operators, i.e. the rapid growth, in terms of code size, of the individuals in the population, that often makes GSGP impossible to be used in practice. A possible workaround to this problem appeared in 2013, with the introduction of a new implementation of GSGP that made it not only usable, but also very efficient [6, 49]. Using this implementation, researchers have recently begun to use GSGP for complex real-life applications, obtaining extremely interesting results. This implementation has also allowed us to shade a light on properties of geometric semantic operators that may help limiting overfitting.

In spite of the exciting results that were obtained with the new implementation, one limitation of GSGP still persists: the task of visualizing and understanding the final solution generated by GSGP is a very hard one, and may be even impossible, given the usually huge size of that solution. For these reasons and many others, research in GSGP is still much in demand and new forces to help advancing the research in GSGP would be more than welcome. This chapter is directed to all those researchers, for instance students or young researchers, that want to invest part of their efforts in this new and exciting field. The objective of this chapter is to present geometric semantic operators in a simple and clear way, so that readers that are not expert in the field can understand them. In particular, the chapter focuses on the reason why geometric semantic operators induce a fitness landscape with no locally suboptimal solutions. To make this presentation as clear as possible, so that a large public can benefit from it, nothing concerning fitness landscapes is given for granted. In other words, in the first part of the chapter, fitness landscapes are introduced at a very elementary level. That part represents the basis to understand the idea behind geometric semantic operators. Then, some results that have been obtained so far by GSGP are also presented. The aim of this last segment of the chapter is not the

[1]When the quality of a solution, or fitness, is equal to an error between calculated values and targets, like in the cases studied in this chapter, the terms *error surface* and *fitness landscape* are synonymous. The former term is generally used by the machine learning community, while the latter is more popular in the evolutionary computation terminology. In this chapter, these two terms will be used interchangeably.

one of presenting a complete discussion of those experimental results. Instead, it is to give an idea of the quality of the results that GSGP can obtain. Finally, the chapter terminates with a discussion of bibliographic material that can be useful to deepen one's knowledge of GSGP.

This chapter is structures as follows: Sect. 2 contains an introduction to optimization problems, which is preparatory for the simple introduction to fitness landscapes contained in Sect. 3. Section 4 presents geometric semantic operators and, using the concepts introduced earlier, explains why they induce a unimodal fitness landscape. Section 5 discusses the main drawback of GSGP and presents the new implementation that makes GSGP efficient. Section 6 presents synthetically some of the applicative results that it was possible to obtain using this implementation of GSGP. Section 7 contains advices on further bibliographic material, that may help deepening one's knowledge of GSGP beyond the reading of this chapter. Finally, Sect. 8 concludes the chapter and suggests ideas for future research.

2 Optimization Problems

Numerical optimization, or more simply optimization [35], is a well established research field of mathematics, computer science and operations research. In informal terms, the objective of an optimization problem is to look for the best solution (or solutions) in a (typically huge) set of possible candidate solutions. It is typical to assume that an optimization problem can be formulated unambiguously in terms of mathematical terminology and notation, that the *quality* of a solution ("how well that solution is able to solve the problem") can be unambiguously quantified, and that it can be compared with the quality of any other solution. Well known examples of possible optimization problems are, for instance, the traveling salesperson problem (TSP) [2] or the knapsack problem (KP) [29].

In optimization, it is usual to distinguish between problems and problem instances: a problem instance is a concrete realization of an optimization problem and an optimization problem can be viewed as a collection of possible problem instances with the same properties. More specifically, an *instance of an optimization problem* is usually defined as a pair (S, f), where S is a set of possible solutions and $f : S \rightarrow \mathbb{R}$ is an evaluation (or alternatively quality, or cost) function, that assigns a real value to every element x of S. For any solution $x \in S$, the real number $f(x)$ is usually interpreted as a quantification of the quality of solution x. In other words, for every possible solution, function f tells us "how well" (or "how poorly") that solution solves the problem. The set of possible solutions S is often called *search space* and (especially in the context of Evolutionary Computation [3], which is the focus here) function f is often called *fitness function*. This is the terminology that will be adopted in this chapter. Considering an instance of an optimization problem, the goal is to find a solution $x^* \in S$, such that $f(x^*) \geq f(x)$ for all $x \in S$ (and in this case the problem is called a maximization problem, since its goal is to find the solution with maximum possible fitness) or a solution $x^* \in S$, such that $f(x^*) \leq f(x)$ for all $x \in S$ (and in this case the problem is called a minimization problem). In both cases, a solution like x^* (i.e. a solution such that a better solution does not exist in the entire search space) is called a *global optimum*.

In the last few decades, a wide variety of optimization problems has emerged from a vast set of different applicative domains, and considerable effort has been devoted to constructing and investigating methods for solving to optimality or proximity several optimization problems [1]. For instance, integer, linear and non-linear programming, as well as dynamic programming, have gained large popularity and are nowadays well established methods. Over the years, it has been shown that many theoretical and practical optimization problems belong to the class of *NP-complete* problems [19]. A direct consequence is that optimal solutions cannot be obtained in reasonable amounts of computation time. For this reason, it is a common trend to apply methods that are generally able to produce solutions quickly, at the risk of sub-optimality. These methods are usually called *heuristic*, or approximation, methods. A heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. Various variants of *local search* algorithms, like *Hill Climbing* [1] or *Simulated Annealing* [1], or population-based methods like *Evolutionary Algorithms* (EAs) [3] or *Particle Swarm Optimization* [23] are just few of the many possible existing heuristic methods. These computational methods share the property of being *general* (i.e. applicable to a wide variety of problems, as opposite to *tailored* algorithms that use problem-specific information, and thus are applicable to a restricted set of problems) and of being *stochastic* (generally meaning, with this term, that they depend on random events, and thus they may return different outputs, if executed several times on the same inputs).

Probably one of the simplest (and, thus, in many cases also one of the less effective) heuristic methods is Hill Climbing. Starting from an initial solution (which is typically randomly generated, unless some a priori information is known), Hill Climbing iteratively tries to improve the quality of the solution, using the concept of *neighborhood*. Given an instance of an optimization problem (S, f), a neighborhood structure is a mapping $\mathcal{N} : S \to 2^S$, that defines for each solution $i \in S$ a set $\mathcal{N}(i) \subset S$ of solutions, called neighbors of i. In synthesis, Hill Climbing works by maintaining in memory a *current* solution, which is improved at every iteration. At the first step the current solution is equal to the initial (typically randomly generated) one. Then, at each iteration, Hill Climbing chooses one neighbor j of the current solution i and makes j the new current solution ($i := j$) if the fitness of j is better than the fitness of i. The algorithm terminates when all the solutions in the neighborhood $\mathcal{N}(i)$ of the current solution i have a worse (or equal) fitness than i. In that situation, i is returned by Hill Climbing as the solution to the problem.

It is clear that Hill Climbing does not necessarily return a global optimum. The solution returned by Hill Climbing is, by definition, the best one in a given neighborhood, which corresponds to the definition of a *local optimum*. More specifically, given an instance of an optimization problem (S, f) and a neighborhood structure $\mathcal{N}$, a local optimum is a solution $k \in S$ such that $f(k) \leq f(j), \forall j \in \mathcal{N}(k)$ in case ofminimization problems, and such that $f(k) \geq f(j), \forall j \in \mathcal{N}(k)$ in case of maximization problems. A global optimum (being the best solution in all the search space *S*) is also a local optimum (it is also the best solution in its particular neighborhood). So, in some particularly lucky situations, the locally optimal solution returned by Hill Climbing can be a global optimum. But no guarantee is given a priori about the ability of Hill Climbing of finding, and returning, a global optimum. As a direct consequence, given that a local optimum can be an even much poorer solution (in terms of fitness) than a global one, no guarantee is given about the quality of the solution returned by Hill Climbing.

3 Fitness Landscapes

Using a landscape metaphor to develop insight about the workings of a complex system originates with the work of Wright on genetics [53]. Probably, the most simple definition of fitness landscape in optimization is the following one: a fitness landscape is a plot where the points in the horizontal direction represent the different solutions in a search space and the points in the vertical direction represent the fitness of each one of these solutions [27]. If the space of solutions can be visualized in two dimensions, the plot can be seen as a three-dimensional "map", which may contain peaks and valleys. The task of finding the best solution to the problem is equivalent to finding the highest peak (for maximization problems). The problem solver is seen as a short-sighted explorer searching for it. For instance, Hill Climbing can be seen as an explorer that cannot see the landscape around him except for a very limited distance given by one step (represented by the distance between one solution and its neighbors), and thus tries one step in each direction and chooses the steepest one. Once the top of a peak has been reached, Hill Climbing terminates, returning the solution corresponding to that pick. Thus, if the reached peak is not the highest one, the latter will never be reached. Many enhancements to Hill Climbing have been proposed. For example, the hill climber may start exploring again from a new randomly chosen starting point, once a peak has been found (Hill Climbing with restarts), or the searcher may be sometimes allowed to climb downhill, with a certain probability (like for instance in *Simulated Annealing* [24]). Possible further alternatives are, for instance, Genetic Algorithms (GAs) and GP, that can be imagined as operating via a population of explorers, initially scattered at random across the landscape. Mimicking the Darwinian principle of natural selection [16], those explorers (*individuals* in the terminology of GAs and GP) that have found relatively high fitness points are rewarded with a high probability of surviving and reproducing.

Crucial for the definition of a fitness landscape is the concept of neighborhood:

> *solutions should be positioned in the horizontal direction consistently with the neighborhood structure.*

For instance, individuals that are positioned close to each other in the horizontal direction should also be neighbors according to the chosen neighborhood structure. In this way, the shape of the landscape can actually give an idea of the "moves" that a searcher can do: moving from one neighbor to the other really corresponds to one step in the landscape. Under this perspective, the definition of fitness landscape is similar to the one of instance of an optimization problem, with the neighborhood structure as a further piece of information. In other words, a fitness landscape can be defined as a triple $(S, f, \mathcal{N})$, where S is the search space, f is the fitness function, and $\mathcal{N}$ the chosen neighborhood structure.

The fitness landscape metaphor can be helpful to understand the difficulty of a problem, i.e. the ability of a searcher to find the optimal solution for that problem. For example, a very smooth and regular landscape with a single hill top (unimodal fitness landscape) is the typical fitness landscape of an easy problem: all searching strategies are usually able to find the top of the hill in a straightforward manner (even the simple Hill Climbing). The opposite is true for a very rugged landscape, with many hills which are not as high as the best one. In this case, even approaches based on populations of explorers, like GAs or GP, would have problems: it might happen that each one of the explorers remains blocked on a sub-optimal peak and none of them reaches the highest one. A fitness landscape might also contain totally flat areas. These might be situated at high or low fitness values, but in both cases there is no gradient and a Hill Climbing explorer has no guidance as to where he should go next.

It is worth stressing again that the definition of the neighborhood relationship is a crucial step in the construction of a fitness landscape: the main characteristics of the landscape, like peaks, valleys, plateaus, among others, are strongly dependent on the neighborhood relationship and deeply change if this relationship is modified. Generally:

> *the neighborhood relationship is defined in terms of the variation operators used to traverse the search space.*

Two individuals are assumed to be neighbors if it is possible to obtain the second one by the application of one step of an operator to the first one. This can be done for unary operators, like mutation, but it is clearly more difficult if binary or multi-parent operators, like crossover, are considered. The lack of a natural ordering for the structures in the search space and the difficulty in defining a neighborhood relationship may prevent one from being able to plot the landscape in practical cases (see for instance the example in Sect. 3.3 in the continuation).

3.1 Example. A Simple Optimization Problem. Hill Climbing and Fitness Landscape

Let us consider the following instance of an optimization problem:

- $S = \{i \in \mathbb{N} \mid 0 \leq i \leq 15\}$
- $\forall i \in S : f(i) =$ number of bits equal to 1 in the binary representation of number i.
- $\forall i, j \in S : i \in \mathcal{N}(j) \iff |i - j| = 1$
- Maximization problem.

In informal terms, the search space consists in all the natural numbers between 0 and 15. The fitness function of each solution is quantified by the number of digits equal to 1 in its binary representation, and a large fitness is better than a small one (maximization problem). Finally, the neighborhood structure is defined in such a way that two solutions are neighbors if their difference in absolute value is equal to 1. This corresponds to the "intuitive" neighborhood of natural numbers, where for instance, number 5 is neighbor of numbers 4 and 6, number 8 is neighbor of numbers 7 and 9, etc.

It is absolutely clear that such an instance of optimization problem is so simple that it can be straightforwardly solved by exhaustively analyzing the search space. A search space characterized by just 16 solutions is, in fact, so tiny that it is completely unrealistic. Furthermore, looking at the definition of this problem, it is immediately clear that the globally optimal solution (the one any algorithm is supposed to look for), is 15, the only solution in the search space represented by four bits equal to 1. In such a situation, using an optimization heuristic does not make sense at all, because the sought for solution is already clear and easy to find. Nevertheless, we simulate an execution of Hill Climbing on this problem, just to exemplify its functioning.

Let us assume that the initial solution, randomly generated with uniform distribution on the entire search space, is 6. The fitness of that solution is 2 (since the binary representation of number 6 is 0110). Also, let us assume that we are using a version of Hill Climbing in which the best neighbor of the current solution is always the chosen candidate for becoming the new current solution (this step is possible in this case, where the neighborhood of each solution is a set of very small size, but can be unrealistic in practice). In such a situation, Hill Climbing will analyse the neighborhood of the current solution, which is the set {5, 7}. Given that the fitness of solution 5 (which in binary is 0101) is equal to 2 and the fitness of solution 7 (which in binary is 0111) is equal to 3, the chosen neighbor is 7. Also, given that solution 7 has a better fitness than 6, 7 will be chosen by Hill Climbing as the new current solution. Iterating the process, Hill Climbing will now analyze the neighborhood of solution 7, which is the set {6, 8}. Given that none of the solutions belonging to this set has a better fitness than the current solution 7, Hill Climbing will terminate and return 7 as the final solution for the problem. It is straightforward to remark that, in this case, Hill Climbing was able to return a locally optimal solution which is not the global one, given that the only global optimum in this search space is solution 15.

Let us now plot the fitness landscape for this problem. This is an easy task in this case, given that the search space has very limited size and the neighborhood has a maximum possible cardinality equal to 2. In such a situation, the fitness landscape is a simple bi-dimensional plot, like the one represented in Fig. 1. As one can imagine, following the previous execution of Hill Climbing, the algorithm has performed a "walk" on the fitness landscape, starting from point 6, then moving to point 7 and finally stopping in that point, that is a "pick" in the fitness landscape, although not the tallest one. Looking once again at Fig. 1, it is not hard to convince oneself that the tallest pick (corresponding to solution 15) can be reached only in case the current solution is randomly initialized on a value that is larger or equal than 12. Technically, we often identify the set of values that, in the case of this example, are larger or equal

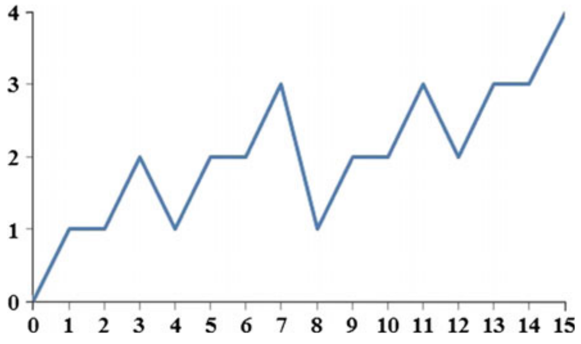

Fig. 1 The simple fitness landscape of the problem studied in the example of Sect. 3.1. On the horizontal axis, all the solutions in the search space, sorted consistently with the used neighborhood structure. On the vertical axis the fitness of the solutions

than 12 using the term *basin of attraction* of solution 15. The basin of attraction of a solution $x \in S$ corresponds to the set B of solutions such that, if any of the solutions in B are ever the current solution of Hill Climbing, then the algorithm will terminate returning x. If x is a pick in the fitness landscape, then its basin of attraction visually corresponds to the "hill" that stands around it.

The behaviour of Hill Climbing that we have just observed is general: Hill Climbing "walks" the fitness landscape by performing small (neighbor to neighbor) steps and by climbing the fitness landscape until it reaches the top of a hill (for maximization problems). When a pick is reached, the corresponding solution is returned, even though that pick is not the tallest one.

Once the three elements that determine a fitness landscape are fixed (search space S, fitness function f and neighborhood structure $\mathcal{N}$), the returned solution can be, or not, a global optimum according to the initialization, which is a random event. The initial solution must fall in the basin of attraction of a global optimum, otherwise Hill Climbing will not be able to reach a global optimum. The probability of the initial solution to fall in the basin of attraction of a global optimum depends on the wideness of those basins of attraction, which are clearly related to the shape of the fitness landscape.

Let us now extend the problem instance studied so far, by increasing the size of the search space. More in particular, let us consider all numbers between 0 and 1023 (instead of all numbers between 0 and 15 like it has been the case so far). In other words, the new search space is now: $S = \{i \in \mathbb{N} \mid 0 \leq i \leq 1023\}$. Let us consider the same fitness function and the same neighborhood that have been considered so far. The fitness landscape corresponding to this new problem instance is reported in Fig. 2. The fitness landscape is clearly very rugged, which indicates that the problem, although extremely simple to define, is very hard to solve using Hill Climbing, or any other optimization heuristic.

Given its very simple definition, it is straightforward to implement this problem and try to solve it using Hill Climbing, or any other existing optimization heuristic. The interested reader is invited to try. It will be immediate to see that finding the globally optimal solution for this problem is extremely hard for all these algorithms.

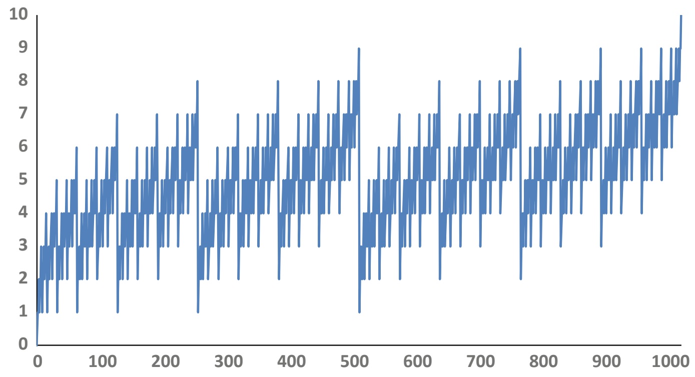

Fig. 2 The fitness landscape of the same problem as in Fig. 1, but in which the search space has been extended. Instead of $S = \{i \in \mathbb{N} \mid 0 \le i \le 15\}$, as it was in Fig. 1, here we consider $S = \{i \in \mathbb{N} \mid 0 \le i \le 1023\}$. All the rest is as in Fig. 1

3.2 Example. Same Problem, Different Fitness Function

Let us consider an instance of an optimization problem very similar to the one studied in the example of Sect. 3.1:

- $S = \{i \in \mathbb{N} \mid 0 \le i \le 15\}$
- $\forall i \in S : \ f(i) = \dfrac{-i^2}{2} + 10\,i$
- $\forall i, j \in S : i \in \mathcal{N}(j) \iff |i - j| = 1$
- Maximization problem.

Basically, this problem is identical to the previous one, except for the fitness function. The fitness landscape corresponding to this problem instance can be represented as in Fig. 3.

Without any further consideration, it is straightforward to see that the fitness landscape, this time, is unimodal: only one pick exists, corresponding to the unique global optimum (10 in this example). This means that this problem should be very easy to solve by any heuristic method, including Hill Climbing. Once again, the interested reader is invited to implement this problem instance and to run Hill Climbing on it. It will be immediate to observe that Hill Climbing will be able to find the globally optimal solution with no difficulty. This behaviour will be observed also if the search space will be extended, as it was done in the last part of the example of Sect. 3.1.

It is instructive to notice that the two instances of optimization problems considered in Sects. 3.1 and 3.2 have exactly the same search space, and exactly the same neighborhood structure. Nevertheless, they have two completely different difficulties: the problem of the example in Sect. 3.1 is very difficult, while the problem of the

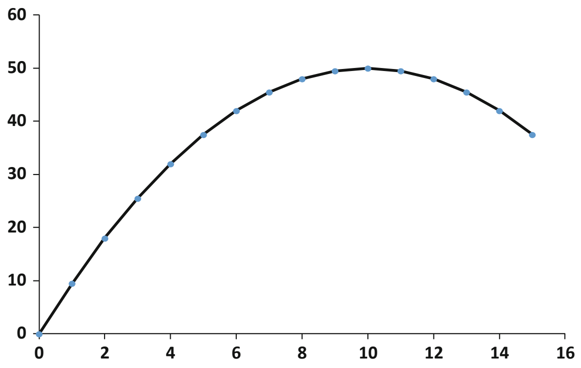

Fig. 3 The simple fitness landscape of the problem studied in the example of Sect. 3.2. On the horizontal axis, all the solutions in the search space, sorted consistently with the used neighborhood structure. On the vertical axis the fitness of the solutions

example in Sect. 3.2 is very easy. Only changing the fitness function has completely changed the shape of the landscape, and thus the difficulty of the problem.

3.3 Example. Same Problem, Different Neighborhood

Let us now consider an apparently minor modification of the problem studied in the example of Sect. 3.1:

- $S = \{i \in \mathbb{N} \mid 0 \leq i \leq 15\}$
- $\forall i \in S: \ f(i) =$ number of bits equal to 1 in the binary representation of number i.
- $\forall i, j \in S : i \in \mathcal{N}(j) \iff$ the binary representations of i and j differ by just 1 bit.
- Maximization problem.

In other words, we are considering the same search space and the same fitness function (and then, actually, we are studying the same instance of an optimization problem) as in Sect. 3.1, but with a different neighborhood structure. Neighborhoods, this time, are larger than in Sect. 3.1. In fact, considering that all integer numbers between 0 and 15 can be represented in binary code using 4 bits, each solution x has exactly 4 neighbors, each of which can be obtained by flipping one bit in the binary representation of x. In this situation, it is clear that it is impossible to draw a fitness landscape as we have done for the previous problems. In fact, the plot should be multidimensional.

Nevertheless, even though we are not able to draw the fitness landscape, it is not difficult to *imagine* its shape, and some of its more relevant characteristics. More in particular, it is not difficult to understand that the unique global optimum is the only solution whose binary code is composed only by bits equal to 1. In fact, all the solutions that are not global optima have at least one bit equal to 0 in their binary representation. So, they have at least one neighbor that is better than themselves, and that neighbor can be obtained by changing that 0 into a 1. As a direct consequence, we can conclude that the fitness landscape of this instance of an optimization problem has no local optima, except the unique global optimum. In other words, the fitness landscape is unimodal. From this reasoning, it is possible to infer that this problem should be easy to solve for any heuristic method, including Hill Climbing, which can easily be confirmed experimentally.

We can even say something more about the fitness landscape of this problem instance: we can imagine a "layered" structure in which solutions belonging to consecutive layers are neighbors. The lowest level (containing the worst solution in the search space) contains only solution 0, whose binary code, of course, contains only bits equal to 0. On the upper level, we can find all solutions whose binary code contains exactly one bit equal to 1. On the further upper level, we can find all solutions whose binary code contains exactly two bits equal to 1. And so on and so fourth until the top layer, which contains only one solution (the global optimum), whose binary code contains only bits equal to 1. Even though this fitness landscape cannot be drawn, one can easily imagine it as a very smooth and regular one, with just one pick representing the global optimum.

It is very instructive to notice that two instances of exactly the same problem, that only differ for using two different neighborhood structures (like in Sects. 3.1 and 3.3) have two completely different fitness landscapes and consequently two completely different difficulties: the instance in Sect. 3.1 is very difficult, while the one considered here is very easy to solve.

Furthermore, one should remind that every neighborhood structure has a transformation (or mutation) *operator* associated to it. This operator is such that, if applied to a solution, it can generate any of its neighbors. One can imagine this operator as the operator that is used by the agent to make *one move* to traverse, or explore, the search space. For instance, the neighborhood structure of the example in Sect. 3.1 corresponds to an operator that can add or subtract 1 to an integer number, while the neighborhood structure of Sect. 3.3 corresponds to an operator that flips one bit in the binary code of a number.

As a general conclusion, we can state that from Sects. 3.1 and 3.3 we have learned the following lesson:

The same problem can be very easy to solve, or very difficult to solve according to the neighborhood structure used, or, which is the same thing, according to the operator used to explore the search space.

This lesson is extremely important for the continuation of this chapter, whose objective is introducing new operators for GP that turn *any* supervised learning problem into a potentially easy one.

3.4 Example. The CONO Problem

Given any $n \in \mathbb{N}$, let us now consider the following instance of an optimization problem:

- $S = \{\overrightarrow{v} \in \mathbb{R}^n \mid \forall i = 1, 2, \ldots, n : v_i \in [0, 100]\}$

- $\forall \overrightarrow{v} \in S: \; f(\overrightarrow{v}) = d(\overrightarrow{v}, \overrightarrow{t})$
- $\forall \overrightarrow{v}, \overrightarrow{w} \in S: \overrightarrow{v} \in \mathcal{N}(\overrightarrow{w}) \iff \forall i = 1, 2, \ldots, n: w_i = v_i + r_i$, where $r_i \in [-1, 1]$.
- Minimization problem.

where d is a distance measure (think, for simplicity, of the Euclidean distance, but the following reasoning holds also for any other metric), and $\overrightarrow{t} \in \mathbb{R}^n$ is a predefined, and known, global optimum.

In informal terms, solutions of this problem instance are vectors of n real numbers included in [0, 100]. The fitness of an individual is equal to the distance of that individual to a unique, and known, globally optimal solution. Finally, the neighborhood structure is defined in such a way that two solutions $\overrightarrow{v}$ and $\overrightarrow{w}$ are neighbors if $\overrightarrow{w}$ consists in a weak perturbation (the maximum positive and negative perturbation is fixed, equal to 1 in this example) of the coordinates of $\overrightarrow{v}$. A mutation operator corresponding to this neighborhood could be an operator that perturbs of a random quantity in $[-1, 1]$ some of the coordinates of a solution.

This is clearly a theoretical problem, because in practice global optima cannot be known a priori (the objective of the heuristic method is exactly finding them). Nevertheless, it is very useful to study and understand this problem for what will follow in this chapter.

Also in this case, like in the previous example, it is impossible to draw the fitness landscape, due to the fact that the size of the search space and the size of the neighborhoods are equal to infinity. Nevertheless, also in this case, some of the characteristics of the fitness landscape can be understood. Consider any solution $\overrightarrow{v}$ different from the unique global optimum $\overrightarrow{t}$. It is not difficult to convince oneself that it is always possible to perturb some of the coordinates of $\overrightarrow{v}$ in order to obtain a solution $\overrightarrow{w}$ (neighbor of $\overrightarrow{v}$) that is closer than $\overrightarrow{v}$ to $\overrightarrow{t}$, and thus with better fitness (given that fitness is equal to distance to $\overrightarrow{t}$). In other words, any solution that is different from the global optimum has a set of neighbors that are better than itself. As a consequence, this problem has no local optima, except for the global optimum. The fitness landscape is unimodal, and the problem is very easy to solve[2] by means of any heuristic method, including Hill Climbing.

A very simple way of perturbing a solution $\overrightarrow{v}$ in order to generate a solution $\overrightarrow{w}$ that is closer than $\overrightarrow{v}$ to $\overrightarrow{t}$ is to modify only one coordinate of $\overrightarrow{v}$ by "approximating" it to the corresponding coordinate of $\overrightarrow{t}$. In other terms, if $\overrightarrow{v} = [v_1, v_2, \ldots, v_n]$ and $\overrightarrow{t} = [t_1, t_2, \ldots, t_n]$, we can consider any $i = 1, 2, \ldots, n$ and replace v_i with any number in the interval $(v_i, t_i]$ if $v_i < t_i$, or in $[t_i, v_i)$ if $v_i > t_i$ (of course, the modification must respect the definition of the neighborhood structure, and thus, in the case of this example, it cannot have a magnitude larger than 1).

Let us consider a simple numeric example: let the global optimum $\overrightarrow{t}$ be equal to $[2, 10, 8, 5]$ and let a solution $\overrightarrow{v}$ be equal to $[4.7, 8.5, 6.3, 7.2]$. If we

[2]The example in Sect. 3.4, contrarily to the previous examples, is a case of continuous optimization. Thus, it is practically impossible to find *exactly* the global optimum. From now on, when continuous optimization is considered, the term "solving" the problem will be used to indicate that it is possible to approximate a globally optimal solution with any prefixed precision.

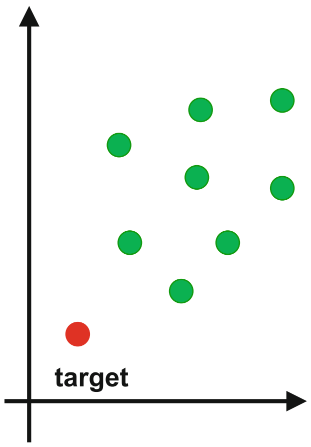

Fig. 4 A simple geometric interpretation of some solutions of the problem of the example in Sect. 3.4 (called CONO problem, as discussed in the text at the end of Sect. 3.4). All solutions are points in an n-dimensional space (the simple case $n = 2$ is represented here for simplicity), including the unique and known global optimum, indicated as *target* in figure

modify $\vec{v}$ by replacing, say, the second coordinate with any value included in (8.5, 10], then we generate a solution that is closer to $\vec{t}$ than $\vec{v}$, and thus it is better in terms of fitness. Actually, given the restrictions in the definition of the neighborhood structure, we can be only able to replace that coordinate with a value included in (8.5, 9.5]. For instance, a solution like [4.7, **8.7**, 6.3, 7.2] is closer than $\vec{v}$ to $\vec{t}$, and thus has a better fitness than $\vec{v}$.

Also, it is worth pointing out that modifying just one coordinate is not the only way of generating a better neighbor. For instance, a solution like [4.8, 8.9, 6.9, 6.4] is closer than $\vec{v}$ to $\vec{t}$. In this example, it is interesting to notice that, even though not all the coordinates went closer to the corresponding coordinate in $\vec{t}$ (the first coordinate is further away from the first coordinate of $\vec{t}$ than it was in $\vec{v}$), the global distance to $\vec{t}$ decreased.

One of the interesting characteristics of this example is that it has a very nice and intuitive *geometric* interpretation. Solutions, in fact, can be interpreted as points in a n-dimensional Cartesian space, as exemplified in Fig. 4, where, for simplicity, the bi-dimensional case ($n = 2$) has been considered. In this perspective, the mutation operator discussed so far (the one that perturbs weakly some coordinates of a solution) can be interpreted as a step in any direction of the space, assuming the movement is included in a *box* of a given side (the maximum possible movement was 1 in this example). For this reason, this mutation operator is called *box mutation*.[3] Also, we call the possible maximum movement in any direction *mutation step* (*ms*).

The fact that the fitness landscape, in a case like this, is unimodal should be easy to understand looking at Fig. 5. That figure shows a simple bi-dimensional space of

[3]In several references [49], the term *ball mutation*, instead of *box mutation*, can be found for indicating this operator. If the area of variation induced by this operator can geometrically be represented as a "box" or a "ball", it depends on the particular metric used, as explained in [36]. In this simple example, we are considering the intuitive Euclidean distance and this is why we use the term box mutation.

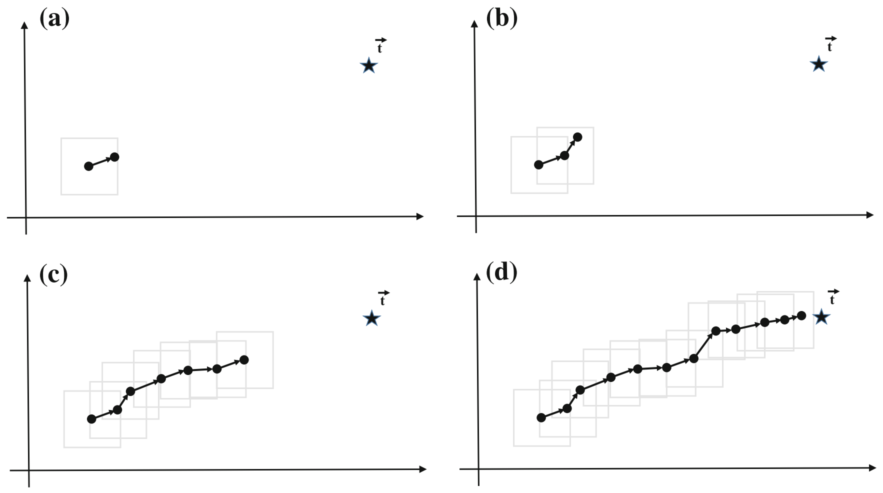

Fig. 5 A simple graphical representation of a chain of solutions $\mathcal{C} = \{s_1, s_2, \ldots, s_h\}$ where, for each $i = 1, 2, \ldots, h-1$, s_{i+1} is a neighbor of s_i (in fact, it is a point that stands inside the box of side $2 \times ms$ centered in s_i. The *boxes* are represented in *gray*). The known global optimum is represented by a *star*. $\mathcal{C}$ could represent the set of solutions visited by a heuristic method like Hill Climbing, step after step. Plot **a** shows a possible first step, with the initial solution and its neighbor, that will become the next current solution. Plot **b** shows the next step. Plots **c** and **d** represent subsequent phases of the execution of the heuristic method, showing how applying box mutation we always have the possibility of generating a neighbor that is closer to the target, and thus with better fitness, than the current solution. This gives us the possibility of approximating the target with any prefixed precision

solutions, inside which a solution is represented by a point. The plots also report the "box" inside which mutation allows to move. The known global optimum $\vec{t}$ is also a point in this Cartesian space, and it is represented in the plot using a star symbol. As it is visible, the neighbor reported in Fig. 5a is closer to the target than the initial solution, which stands in the center of the box. What is represented in Fig. 5a could be the first step of an execution of a heuristic method, like Hill Climbing. The entire execution consists in an iteration of steps like this one. The next step is represented in Fig. 5b, and Fig. 5c, d represent subsequent phases of the execution, showing how applying box mutation we are free to move in any position inside the box and thus we always have the possibility of getting closer to the target. This implies that no local optima, except the global optimum, can exist and the fitness landscape for this problem is unimodal.

Concerning this last sentence, it is very important to remark that the definition of local optimum is a solution that does not have *any* neighbor better than itself. If a set of neighbors with better fitness (or even only one of them) exists, then the solution is not a local optimum. This does *not* necessarily imply that a mutation operator will actually *find* one of those better neighbors. Under this perspective, the sequence of solutions represented in Fig. 5 has to be interpreted as a *lucky* sequence in which, at every step, we have been able to get closer to the global optimum. This does not have to necessarily be the case, and this is generally *not* what happens in practice. On the other hand, when executing a heuristic method on this problem, it is possible that several attempts and some computational effort may be spent before a better neighbor than the current solution is found at each step.

Nevertheless, in this problem, we are *sure* that, for each possible solution different from the global optimum, such a better neighbor *exists*, and thus it makes sense to continue looking for it!

This is an extremely important observation. Much more important than it may seem at first sight. This characteristic distinguishes this problem from practically all the real-life problems that we are used to study, which are instead characterized by rugged fitness landscapes with the presence of many local optima (i.e. solutions that *cannot* possibly be improved!).

The fact of having the *possibility* of improving is actually such an important fact that it makes *all the difference*. In order to convince oneself about this crucial point, one should really try to implement this problem, run Hill Climbing on it, and see with her own eyes how actually easy it is for Hill Climbing to approximate the global optimum in this case.

On YouTube, it is possible to find a video from the movie "Jurassic Park" in which the protagonist pronounces the famous sentence "Life finds a way" (https://www.youtube.com/watch?v=SkWeMvrNiOM). In the movie, the protagonist is trying to explain a simple concept: if there is a possibility of an event to happen, and even in case that event is unlikely, sooner or later evolution will find a way of making it happen, provided that we give it enough time. Extrapolating, and possibly also a bit abusing of that sentence, we could here assert that "optimization algorithms find a way".

What makes the difference is having, or not having, the possibility of improving!

If that possibility exists, the algorithm will eventually find a way to make it happen. And that possibility is ultimately the information that fitness landscapes are giving to us. If only one pick exists in the fitness landscape, this clearly indicates that the problem is easy, in the sense that the algorithm will find a way to get closer and closer to it.

The problem discussed here is so important for the continuation of this chapter, that it deserves to be identified by a name. We choose to call it CONO, which stands for Continuous Optimization with Notorious Optimum. We find this acronym particularly appropriate, because the shape of the fitness landscape for this problem is actually a cone,[4] where the vertex represents the global optimum.

[4]The word "cono" actually means cone in several languages of Latin origin, among which Italian and Spanish.

4 Geometric Semantic Genetic Programming

4.1 A Brief Introduction to Genetic Programming

Evolutionary Algorithms (EAs) are a set of computational intelligence methods that mimic the basic features of the theory of evolution of C. Darwin [16] (reproduction, likelihood of survival, variation, inheritance and competition). EAs work by maintaining and evolving a set, or *population*, of candidate solutions. These potential solutions (called *individuals* from now on, following EAs terminology) are generally data structures, for each of which (as in the examples of the previous section), it is possible to calculate fitness. The process on which EAs are based is an iterative stepwise refinement of the fitness of the individuals, that uses principles inspired by natural evolution. Synthetically, a *selection* mechanism gives the best individuals in the population a high probability of being chosen for mating, following Darwin's idea of likelihood of survival and competition, and genetic operators allow us to explore the search space by generating new individuals, implementing Darwin's principles of reproduction, inheritance and variation. The two main genetic operators of EAs are *crossover* and *mutation*. Crossover exchanges some features of a set of (usually two) individuals to create offspring that are a combination of their parents. Mutation changes some parts of an individual's structure.

One of the main features distinguishing one type of EA from another is what kind of data structures are evolving in the population or, in other terms, what is the representation of the individuals. Genetic Programming (GP) is one of the most recent and sophisticated EAs. In its general definition, the individuals in the population are computer programs. In this perspective, GP can be seen as a method of automatic generation of programs, able to solve a pre-specified task. In its original, and probably most popular, formulation, the programs representing the individuals can be represented as *trees* (this idea was inspired by the Lisp programming language, in which both programs and data structures are represented as trees). More in particular, programs are defined using two sets: a set of primitive functions $\mathcal{F} = \{f_1, f_2, \ldots, f_p\}$ (that appear in the internal nodes of the trees) and a set of terminal symbols $\mathcal{T} = \{t_1, t_2, \ldots, t_q\}$ (which represent the leaves of the trees). With this representation, crossover usually selects at random one subtree in one of the parents and one subtree in the other parent and exchanges these two subtrees to create the offspring. Mutation usually selects at random one subtree from an individual and replaces it with a random tree. Several other alternative operators have been defined so far (see for instance [40]), but these two are the first ones that have been defined and they are probably still nowadays the most used. Finally, besides the tree-based representation of GP individuals, several others have been introduced so far, including linear representations, grammars, graph based representations, etc. (see again [40] for a rather complete presentation of the different existing representations).

4.2 Symbolic Regression with Genetic Programming

Given:

- a set of vectors $\mathbf{X} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}\}$, where for all $i = 1, 2, \ldots, n$ $\overrightarrow{x_i} \in \mathbb{R}^m$,
- a vector $\overrightarrow{t} = [t_1, t_2, \ldots, t_n]$, where for all $i = 1, 2, \ldots, n$ $t_i \in \mathbb{R}$

a symbolic regression problem can be generally defined as the problem of finding, or approximating, a function $\phi : \mathbb{R}^m \to \mathbb{R}$, also called target function, such that:

$$\forall i = 1, 2, \ldots, n : \ \phi(\overrightarrow{x_i}) = t_i.$$

GP is typically used to solve symbolic regression problems using a set of primitive functions $\mathcal{F}$ that are mathematical functions, like for instance the arithmetic functions or others, and a set of terminal symbols $\mathcal{T}$ that contain at least m different real valued variables, and may also contain any set of numeric constants. In this way, a GP individual (or program) P can be seen as a function that, for each input vector $\overrightarrow{x_i}$ returns the scalar value $P(\overrightarrow{x_i})$. In symbolic regression, to measure the fitness of an individual P any distance metric (or error) between the vector $[P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), \ldots, P(\overrightarrow{x_n})]$ and the vector $[t_1, t_2, \ldots, t_n]$ can be used. Just as a matter of example, one may use, for instance, the mean Euclidean distance, or root mean square error, and in this case the fitness $f(P)$ of a GP individual P is:

$$f(P) = \sqrt{\frac{\sum_{i=1}^{n} (P(\overrightarrow{x_i}) - t_i)^2}{n}}, \tag{1}$$

or one may use the Manhattan distance, or absolute error, and in this case the fitness $f(P)$ of a GP individual P is:

$$f(P) = \sum_{i=1}^{n} |P(\overrightarrow{x_i}) - t_i|. \tag{2}$$

Using any error measure as fitness, a symbolic regression problem can be seen as a minimization problem, where the optimal fitness value is equal to zero (in fact, any individual with an error equal to zero behaves on the input data exactly like the target function ϕ).

Vectors $\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}$ are usually called input data, input vectors, training instances or *fitness cases*, while the values $t_1, t_2, \ldots, t_n$ are usually identified as the corresponding *target values*, or expected output values. $\mathbf{X}$ is usually called *dataset.* Finally, the values $P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), \ldots, P(\overrightarrow{x_n})$ are usually called the *output values* of individual P on the input data.

Example. Let $\mathbf{X} = \{[3, 12, 1], [5, 4, 2]\}$ and let $\overrightarrow{t} = [27, 13]$. For instance, GP individuals may be coded using as primitive functions the set of arithmetic operators

$\mathcal{F} = \{+, -, *\}$ and as terminals a set of three real valued variables (since the input vectors have cardinality equal to 3) $\mathcal{T} = \{k_1, k_2, k_3\}$. In this way, the search space contains all the trees that can be built by composing the symbols in $\mathcal{F}$ and $\mathcal{T}$ (with the only exception that usually a maximum possible depth is imposed to the trees in order to avoid having an infinite search space). Using, for instance, the absolute error, one may calculate the fitness of an individual like, for instance:

$$P(k1, k2, k3) = k3 * (k1 - k2).$$

To do that, one has to first calculate the output values of P on the input data. In other words, one has to calculate $P(3, 12, 1)$ (obtained by replacing the values of the first input vector in the dataset $\mathbf{X}$ with k_1, k_2 and k_3 respectively in the expression of P) and $P(5, 4, 2)$ (obtained by replacing the values of the second input vector in the dataset $\mathbf{X}$ with k_1, k_2 and k_3 in P). So, the fitness of P is:

$$\begin{aligned} f(P) &= |P(3, 12, 1) - 27| + |P(5, 4, 2) - 13| = \\ &= |(1 * (3 - 12)) - 27| + |(2 * (5 - 4)) - 13| = \\ &= |9 - 27| + |2 - 13| = 18 + 11 = 47. \end{aligned}$$

It is not difficult to realize that, in this example, a global optimum, i.e. an individual that has a fitness equal to zero, is:

$$P_{opt}(k_1, k_2, k_3) = k_1 + k_2 + k_2.$$

From this, we can see that GP individuals do not have to necessarily use all the variables in $\mathcal{T}$ (for instance, P_{opt} is not using k_3).

In this simple example only the binary operators of sum, subtraction and multiplication have been used. When division is also used, it is typical to "protect" it in some way from failure in case the denominator is equal to zero. The oldest and more popular method to protect division is to replace it with an operator that is equal to the division if the denominator is different from zero and that returns a constant value otherwise [25]. Nevertheless, several more sophisticated methods have been introduced [22].

Generalization. Symbolic regression can be seen as a supervised machine learning problem (supervised because the target values $t_1, t_2, \ldots, t_n$ are known for every vector of input data in the dataset). As such, the objective is usually finding a model of the data that may work not only for the information that is known (the data in the dataset), but that also works for other, unseen data. In this sense, we say that the model should be *general* and the computational method that generates the model (GP in our case) should have *generalization ability*.

Just as a matter of example, in the next section, as a case study of our experiments, we study a real-life application which is a symbolic regression problem, where the various input vectors $\overrightarrow{x_i}$ are vectors of molecular descriptors, i.e. vectors of numbers that univocally represent a molecular compound that is a candidate new drug. For each one of these molecular compounds, the corresponding target value represents the toxicity of that compound. In this situation, the objective of the problem is finding a function (data model) that, when applied to the vector of molecular descriptors of a particular compound, returns as output an approximated value of the toxicity of that compound. But, what would be the point of having a function that works correctly in returning the toxicity *just* for the compounds that we have in the dataset? In the end, for *those* compounds, we *already* know the value of the toxicity: it is their target value, which is in the dataset.

The objective is to discover the relationship between input data (molecular descriptors) and expected outputs (values of the toxicity), "coding" that relationship in the function (data model) that GP, or another computational method, is supposed to find. In this way, applying this function to *other* molecular compounds, that are not included in the dataset, and for which we *don't know* the value of the toxicity, we should be able to have a faithful estimate of it.

Several ways have been defined so far to test the generalization ability of GP. One of the most simple and intuitive ones consists in partitioning the dataset into two subsets, and using only one of them, called *training* set, to generate the data model. The remaining portion of the dataset, called *test* set, is then used only to verify that the model that has been found is general enough to faithfully approximate the target also on data that have not been used to generate it. In several cases, as also in the experiments presented in the next section, the training set is chosen by selecting randomly, with uniform distribution, 70 % of the instances of the dataset, while the remaining 30 % form the test set.

4.3 Semantics in Genetic Programming

As discussed in the previous section, let $\mathbf{X} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}\}$ be the set of input data, or fitness cases, of a symbolic regression problem, and $\overrightarrow{t} = [t_1, t_2, \ldots, t_n]$ the vector of the respective expected output or target values (in other words, for each $i = 1, 2, \ldots, n$, t_i is the expected output corresponding to input $\overrightarrow{x_i}$) [40]. Let P be a GP individual (or program). As discussed in the previous section, P can be seen as a function that, for each input vector $\overrightarrow{x_i}$ returns the scalar value $P(\overrightarrow{x_i})$. Following [32], the *semantics* of P is given by the vector

$$\overrightarrow{s_P} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), \ldots, P(\overrightarrow{x_n})]$$

In other words, from now on, we indicate with the term semantics the vector of the output values of a GP individual on the input data.

This vector can be represented as a point in a n-dimensional space, that we call *semantic space*. Remark that the target vector $\overrightarrow{t}$ itself is a point in the semantic space and, in symbolic regression, it is known. What is not known is the tree structure of a GP individual that has $\overrightarrow{t}$ as its own semantics.

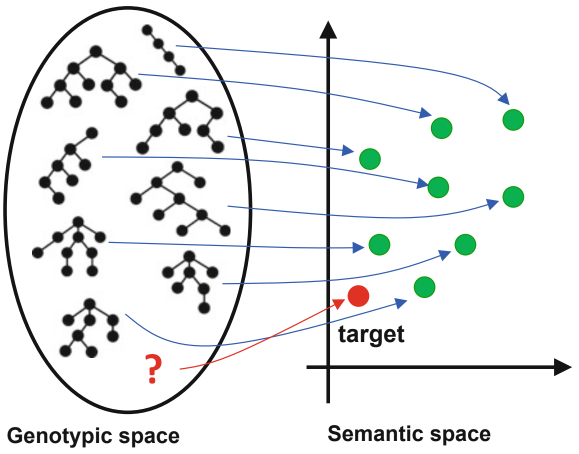

Fig. 6 When we work with GP, we can imagine the existence of two spaces: the genotypic space, in which individuals are represented by their structures, and the semantic space, in which individuals are represented by points, that are their semantics. In figure, the semantic space is represented in 2D, which corresponds to the unrealistic case in which only two training instances exist

Basically, when working with GP, one may imagine the existence of two different spaces: one, that we call genotypic or syntactic space, in which GP individuals are represented by tree structures (or, according to the different considered variant of GP, linear structures, graphs, grammars, etc., as presented in Sect. 4.1), and one, that we call semantic space, in which GP individuals are represented by their semantics, and thus by points. The situation is exemplified in Fig. 6, where, once again, for simplicity, the semantic space is represented in two dimensions, which corresponds to the toy case in which only two training instances exist. Figure 6 also shows that to each tree in the genotypic space corresponds a point in the semantic space. This correspondence is surjective and, in general, not bijective, since different trees can have the same semantics. Once again, it is interesting to point out, as it is visible in the figure, that, since we are working with supervised learning problems, the target (i.e. vector $\overrightarrow{t}$) is a known point in the semantic space (it is in the dataset, it is part of the definition of the problem). What is unknown is its corresponding tree(s) in the genotypic space.

Once we have the semantics of a GP individual, it is immediate to see that the fitness of that individual can be calculate as a distance between the semantics and the target, using any metric. For instance, Eq. (1) is the mean Euclidean distance between $\overrightarrow{s_P} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), \ldots, P(\overrightarrow{x_n})]$ and $\overrightarrow{t} = [t_1, t_2, \ldots, t_n]$, while Eq. (2) is the Manhattan distance between $\overrightarrow{s_P}$ and $\overrightarrow{t}$.

Example. Let us consider, once again, the problem of the example of Sect. 4.2. In that example, we studied a symbolic regression problem in which the set of input data was $\mathbf{X} = \{[3, 12, 1], [5, 4, 2]\}$ and the vector of the corresponding target values was $\overrightarrow{t} = [27, 13]$. In that example, we have considered the set of primitive functions $\mathcal{F} = \{+, -, *\}$ and a set of terminals composed by three real valued variables $\mathcal{T} = \{k_1, k_2, k_3\}$. Also, in that example, we have studied an individual, whose expression in infix notation is $P(k1, k2, k3) = k3 * (k1 - k2)$. What is the semantic of this individual?

The semantic of $P(k1, k2, k3) = k3 * (k1 - k2)$, in this case, is a vector of dimension equal to 2, because there are 2 training instances (the vectors contained in dataset **X**), where the first component is equal to the evaluation of P on the first input vector and the second component is equal to the evaluation of P on the second input vector. In other words, the first component is obtained by replacing the values 3, 12 and 1 respectively to k_1, k_2 and k_3, and evaluating P. Analogously, the second component is obtained by replacing the values 5, 4 and 2 to k_1, k_2 and k_3. So:

$$\overrightarrow{s_P} = [P(3, 12, 1), P(5, 4, 2)] = [1 * (12 - 3), 2 * (5 - 4)] = [9, 2]$$

Using, for instance, the absolute error between semantics and target (as in the example of Sect. 4.2), we have that the fitness of P is:

$$f(P) = d([9, 2], [27, 13]) = |9 - 27| + |2 - 13| = 18 + 11 = 29$$

where, in this case, d is the Manhattan distance. Comparing this calculation with the one of the example of Sect. 4.2 to calculate the fitness of P, we can clearly see that these two calculations are completely identical.

Similarity Between the Semantic Space of Symbolic Regression and the CONO Problem. Once you have understood what the semantics of a GP individual is, and the fact that the semantics can be interpreted as a point in an n dimensional Cartesian space, please compare Fig. 6 to Fig. 4.

Figure 6 represents the two spaces that can be identified when solving a symbolic regression problem with GP (i.e. the genotypic and the semantic space). Figure 4 represents the space of solutions of the CONO problem, discussed in the example of Sect. 3.4. As you can clearly see, the semantic space of a symbolic regression problem is *exactly the same* as the space of solutions of the CONO problem.

But, as we have seen in Sect. 3.4, the CONO problem has a very interesting property! Let us repeat this property here:

IF box mutation is the operator used to explore the search space
THEN The CONO problem has a unimodal fitness landscape

As we can now foresee, this fact has a very important consequence on the solution of symbolic regression problems using GP:

IF we define a GP operator that works like box mutation on the semantic space
THEN **any** *symbolic regression problem has a unimodal fitness landscape*

Let us repeat this fact again, but with different words:

if we were able to define, on the syntax of the GP individuals (i.e. on their tree structure)[5] *an operator that has, on the semantic space, the same effect as box mutation, then the fitness landscape is unimodal for any symbolic regression problem.*

[5]How could it be otherwise? GP is working with a population of trees, so the genetic operators can only act on them!

In other terms, if we were able to define such an operator, then we would be able to match any symbolic regression problem into an instance of the CONO problem that uses box mutation, in the sense that we would be able to perform exactly the same actions in a space (the semantic space of GP) that is identical. As such, we would inherit the same fitness landscape properties from the CONO problem, i.e. a unimodal fitness landscape.

It is worth stressing here one point:

> *This property would hold for any symbolic regression problem, independently on how large or complex the data of the problem are.*

Not having any locally sub-optimal solution, actually *any* symbolic regression problem could be *easy* to optimize for GP (at least on the training set, were fitness is calculated), including problems characterized by huge amounts of data. This fact would clearly foster GP as a very promising method for facing the new challenges of *Big Data* [18].

At this point, I would like beginner readers, like for instance students, to stop for a second and reflect on the importance and impact that this would have. For years, one of the main justifications that researchers have given to the limitations of GP was the fact that it was extremely difficult to study its fitness landscapes, because of the extreme complexity of the genotype/phenotype mapping, and because of the complexity of the neighborhoods induced by GP crossover and mutation. Introducing these new operators, we would not have to worry about this anymore! At least for symbolic regression (but we will see at the end of this chapter that similar operators can be found also for different types of problems), we would have the certainty that the fitness landscape is unimodal, and thus extremely simple.

Furthermore: how many machine learning methods do you know in which the error surface is guaranteed to be unimodal for any possible application? Saying that this GP system would be the first machine learning system to induce unimodal error surfaces would probably be inappropriate; it is not impossible that other machine learning systems with this characteristic may have been defined so far. But still, it is absolutely legitimate to say that this characteristic is quite rare in machine learning, and should give a clear advantage to GP, compared to other systems, at least in terms of evolvability.

All we have to do for obtaining such an important result is to define an operator that, acting on trees, has an effect on the semantic space that is equivalent to box mutation. In other words, if we mutate a GP individual P_1, obtaining a new individual P_2, the semantics of P_2 should be like the semantics of P_1 except for a perturbation of its coordinates, whose magnitude is included in a predefined range. This is the objective of geometric semantic mutation, that is defined in the continuation.

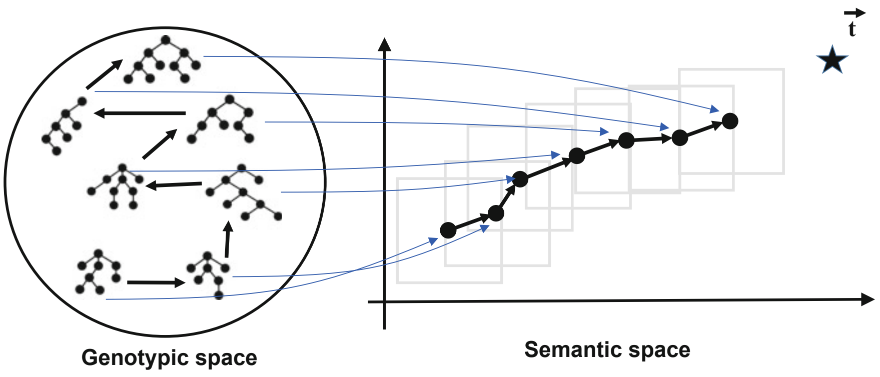

Fig. 7 A simple graphical representation, in the genotypic space, of a chain of solutions $\mathcal{C} = \{s_1, s_2, \ldots, s_h\}$ where, for each $i = 1, 2, \ldots, h-1$, s_{i+1} is a neighbor of s_i, and the corresponding points in the semantic space. The known vector of target values is represented in the semantic space by a star. The similarity between the semantic space in this plot and Fig. 5 should convince the reader that GP with geometric semantic mutation is actually mimicking the CONO problem in the semantic space, thus inheriting its unimodal fitness landscape

4.4 Geometric Semantic Mutation

The objective of Geometric Semantic Mutation (GSM) is the one of generating a transformation on the syntax of GP individuals that has the same effect on their semantics as box mutation. The situation is exemplified in Fig. 7. More in particular, Fig. 7 represents a chain of possible individuals that could be generated by applying GSM several times, with their corresponding semantics. Given that the semantics of the individuals generated by mutation can be any point inside a box of a given side centered in the semantics itself, GSM has always the possibility of creating an individual whose semantics is closer to the target (represented by a star symbol in Fig. 7). As a direct consequence, the fitness landscape has no locally suboptimal solutions.

By comparing the semantic space of Fig. 7 with Fig. 5, it should not be difficult to see that if we use GSM, we *are* actually performing a CONO problem with box mutation (discussed in Sect. 3.4) on the semantic space.

The definition of GSM, as given in [32], is:

Definition 1 Geometric Semantic Mutation (GSM). Given a parent function $P : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step ms returns the real function $P_M = P + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions.

It is not difficult to have an intuition of the fact that GSM has the same effect as box mutation on the semantic space. In fact, one should consider that each element of the semantic vector of P_M is a "weak" perturbation of the corresponding element in P's semantics. We informally define this perturbation as "weak" because it is given by a random expression centred in zero (the difference between two random

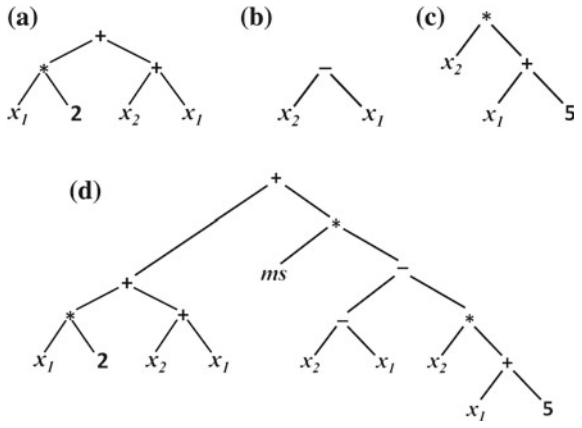

Fig. 8 A GP individual P that has to be mutated (plot **a**), two random tree T_{R1} and T_{R2} (plots **b** and **c** respectively) and the individual P_M generated by geometric semantic mutation (plot **d**)

trees). Nevertheless, by changing parameter ms, we are able to tune the "step" of the mutation, and thus the importance of this perturbation. Figure 8 gives a visual representation of the tree generated by GSM on a simple example. The tree P, that is mutated, is represented in Fig. 8a. The two used random trees T_{R1} and T_{R2} are shown in Fig. 8b and c respectively. Finally, Fig. 8d shows the resulting tree P_M generated by GSM.

In practice, and in order to make the perturbation even weaker, it is often useful to limit the codomain of the possible outputs of T_{R1} and T_{R2} into a given predefined range. This allows us to better "control" what mutation can do. The typical situation is that T_{R1} and T_{R2} are forced to assume values in [0, 1]. This can be done easily, for instance, by "wrapping" T_{R1} and T_{R2} inside a logistic function. In other words, random trees are generated and the logistic is applied to their output before plugging them into P_M. Even though this approach has been the object of criticism [17], it has allowed us to obtain the excellent experimental results that are presented later in this chapter.

Before continuing, it is extremely important to stop for a while and convince oneself about the importance of having random *trees/expressions* (like T_{R1} and T_{R2}) in the definition of GSM, instead of just having random "numbers". The motivation is the following: when we mutate an individual, we want to perturb each one of the coordinates of its semantics by a *different* amount. It is easy to understand why this is important by considering a simple numeric example.

Let the semantic of an individual P be, say, $s_P = [5.6, 9.2]$, and let the target vector be $\overrightarrow{t} = [9, 10]$ (we are again considering the unrealistic bi-dimensional case for simplicity). If we used just random numbers for applying the perturbation, then we would be able to only mutate each one of the coordinates by the same amount. So, if we mutate by, say, 0.5, we obtain a new individual P_M whose semantics is [6.1, 9.7]. Even though P_M has a semantics that is closer to the target than P, it should not be difficult to convince oneself that if we iterate mutation, even in case we change the entity of the perturbation at each step, we will never have any chance of reaching the target. The only possibility that we have to reach the target is to mutate the first coordinate *more* (i.e. of a larger amount) than the second one, simply because the first coordinate of s_P is further away from the corresponding coordinate of $\overrightarrow{t}$ than the second coordinate. So, we need a way of doing a perturbation that has to possess the following properties: (1) it has to be random; (2) it has to be likely to be different for each coordinate; (3) it does not have to use any information from the dataset.

With point (3), we mean that the algorithm that makes the perturbation cannot have a form like:

$$\text{“}\textbf{if } (\overrightarrow{x_i} = \ldots)\textbf{then} \text{ perturbation} = \ldots\text{''},$$

because in this case the system would clearly overfit training data.

Under these hypothesis, the only way we could imagine of doing the perturbation was to sum to the value calculated by individual P the value calculated by a random expression. A random expression, in fact, is likely to have different output values for the different fitness cases.

Last but not least, the fact that the difference between two random expressions is used ($T_{R1} - T_{R2}$) instead of just one random expression can be justified as follows. Especially in the final part of a GP run, it may happen that some of the coordinates have already been approximated in a satisfactory way, while it is not the case for others. In such a situation, it would be useful to have the possibility of modifying *some* coordinates and *not* modifying (or, which is the same, modifying by an amount equal to zero) others. The difference between two random expressions is a random expression *centered in zero*. This means that its output value is likely to be equal to zero, or close to zero, more than to be equal to any other value. In other words, by using the difference between two random expressions, we are imposing that some coordinates may have a perturbation that is likely to be equal, or at least as close as possible, to zero.

4.5 Geometric Semantic Crossover

GP practitioners may be surprised to notice that, so far, basically only mutation has been considered, practically ignoring crossover. On the other hand, crossover is known as the most powerful genetic operator, at least in standard GP. This section intends to fill this gap, by presenting a Geometric Semantic Crossover (GSC) that can behave, on the semantic space, as geometric crossover of GAs (defined in [30]) in continuous optimization.

Geometric GAs crossover works by generating one offspring o that has, for each coordinate i, a linear combination of the corresponding coordinates of the parents p and q, with coefficients included in [0, 1], whose sum is equal to 1. In other words, for all $i = 1, 2, \ldots, n$:

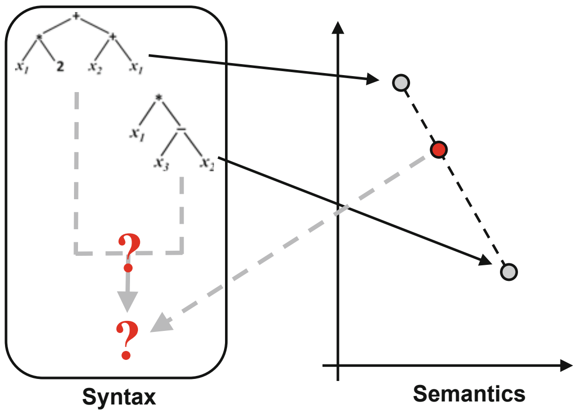

Fig. 9 Graphical representation of geometric semantic crossover, in the simple case of bi-dimensional semantic space. The offspring generated by this crossover has a semantics that stands in the segment joining the semantics of the parents

$$o_i = (a_i * p_i) + (b_i * q_i)$$

where $a_i \in [0, 1]$, $b_i \in [0, 1]$ and $a_i + b_i = 1$.

Under this hypothesis, the offspring can geometrically be represented as a point that stands in the segment joining the parents.

This is the behaviour that GSC must have on the semantic space. The situation is exemplified in Fig. 9. The objective of GSC is to generate the tree structure of an individual whose semantics stands in the segment joining the semantics of the parents. Following [32], GSC is defined as follows:

Definition 2 Geometric Semantic Crossover (GSC). Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$.

It is not difficult to see from this definition that T_{XO} has a semantics that is a linear combination of the semantics of T_1 and T_2, with random coefficients included in [0, 1] and whose sum is 1. The fact that we are using a random expression T_R instead of a random number can be interpreted analogously to what we have done for explaining the use of random expressions in GSM. Furthermore, it is worth mentioning that in Definition 2 the fitness function is supposed to be the Manhattan distance; if Euclidean distance is used, then T_R should be a random constant instead. The interested user is referred to [32] for an explanation of this concept. Figure 10 gives a visual representation of the tree generated by GSC on a simple example.

The fact that the semantics of the offspring T_{XO} stands in the segment joining the semantics of the parents T_1 and T_2 has a very interesting consequence: the offspring generated by GSC cannot be worse than the worst of its parents. To convince oneself of this property, one could simply draw a point in the semantic space of Fig. 9, interpreting this point as the target $\vec{t}$. One should easily see that, independently from the position where that point was drawn, it is impossible that the distance of the semantics of T_{XO} is larger than both distances of both parents T_1 and T_2 to that point. For a deeper discussion of this property, the reader is referred to [30, 32].

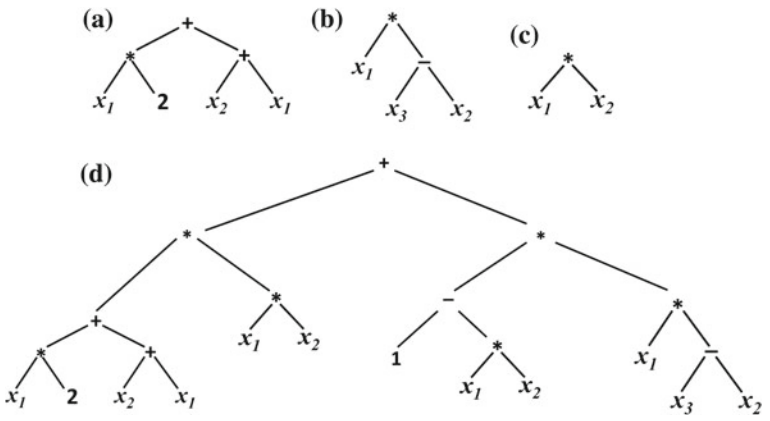

Fig. 10 Two parents T_1 and T_2 (plots **a** and **b** respectively), one random tree T_R (plot **c**) and the offspring T_{XO} of the crossover between T_1 and T_2 using T_R (plot **d**)

5 Drawback and New Implementation

Looking at their definition (and at Figs. 8 and 10), it is not hard to see that geometric semantic operators create offspring that contain the complete structure of the parents, plus one or more random trees and some additional arithmetic operators: the size of the offspring is thus clearly much larger than the size of their parents. The rapid growth of the individuals in the population, shown by Moraglio et al. [32], makes these operators unusable in practice: after a few generations the population becomes unmanageable because the fitness evaluation process becomes unbearably slow.

The solution suggested in [32] consists in performing an automatic simplification step after each generation in which the individuals are replaced by (hopefully smaller) semantically equivalent ones. However, this additional step adds to the computational cost of GP and is only a partial solution to the progressive size growth. Last but not least, depending on the particular language used to code individuals and the used primitives, automatic simplification can be a very hard task.

Here I present a novel implementation of GP using these operators that overcomes this limitation, making them efficient without performing any simplification step. This implementation was first presented in [6, 49]. Although the algorithm is described assuming the representation of the individuals is tree based, the implementation fits any other type of representation.

In a first step, we create an initial population of (typically random) individuals, exactly as in standard GP. We store these individuals in a table (that we call P from now on) as shown in Fig. 11a, and we evaluate them. To store the evaluations we create a table (that we call V from now on) containing, for each individual in P, the values resulting from its evaluation on each fitness case (in other words, it contains the semantics of that individual). Hence, with a population of n individuals and a training set of k fitness cases, table V will be made of n rows and k columns. Then,

(a)

Id	Individual
T_1	$x_1 + x_2x_3$
T_2	$x_3 - x_2x_4$
T_3	$x_3 + x_4 - 2x_1$
T_4	x_1x_3
T_5	$x_1 - x_3$

(b)

Id	Individual
R_1	$x_1 + x_2 - 2x_4$
R_2	$x_2 - x_1$
R_3	$x_1 + x_4 - 3x_3$
R_4	$x_2 - x_3 - x_4$
R_5	$2x_1$

(c)

Id	Operator	Entry
T_6	crossover	$\langle \text{ID}(T_1), \text{ID}(T_4), \text{ID}(R_1) \rangle$
T_7	crossover	$\langle \text{ID}(T_4), \text{ID}(T_5), \text{ID}(R_2) \rangle$
T_8	crossover	$\langle \text{ID}(T_3), \text{ID}(T_5), \text{ID}(R_3) \rangle$
T_9	crossover	$\langle \text{ID}(T_1), \text{ID}(T_5), \text{ID}(R_4) \rangle$
T_{10}	crossover	$\langle \text{ID}(T_3), \text{ID}(T_4), \text{ID}(R_5) \rangle$

Fig. 11 Illustration of the example described in Sect. 5. **a** The initial population P; **b** The random trees used by crossover; **c** The representation in memory of the new population P'

for every generation, a new empty table V' is created. Whenever a new individual T must be generated by crossover between selected parents T_1 and T_2, T is represented by a triplet $T = \langle \text{ID}(T_1), \text{ID}(T_2), \text{ID}(R) \rangle$, where R is a random tree and, for any tree τ, ID(τ) is a *reference* (or memory pointer)[6] to τ (using a C-like notation). This triplet is stored in an appropriate structure (that we call $\mathcal{M}$ from now on) that also contains the name of the operator used, as shown in Fig. 11c. The random tree R is created, stored in P, and evaluated in each fitness case to reveal its semantics. The values of the semantics of T are also easily obtained, by calculating $(T_1 \cdot R) + ((1 - R) \cdot T_2)$ for each fitness case, according to the definition of geometric semantic crossover, and stored in V'. Analogously, whenever a new individual T must be obtained by applying mutation to an individual T_1, T is represented by a triplet $T = \langle \text{ID}(T_1), \text{ID}(R_1), \text{ID}(R_2) \rangle$ (stored in $\mathcal{M}$), where R_1 and R_2 are two random trees (newly created, stored in P and evaluated for their semantics). The semantics of T is calculated as $T_1 + ms \cdot (R_1 - R_2)$ for each fitness case, according to the definition of geometric semantic mutation, and stored in V'. In the end of each generation, table V' is copied into V and erased. All the rows of P and $\mathcal{M}$ referring to individuals that are not ancestors[7] of the new population can also be erased. Note that, while $\mathcal{M}$ grows at every generation, by keeping the semantics of the individuals separated we are able to use a table V whose size is independent from the number of generations.

Summarizing, this algorithm is based on the idea that, when semantic operators are used, an individual can be fully described by its semantics (which makes the syntactic component much less important than in standard GP), a concept discussed in depth in [32]. Therefore, at every generation we update table V with the semantics of the new individuals, and save the information needed to build their syntactic structures without explicitly building them.

In terms of computational time, it is worth emphasizing that the process of updating table V is very efficient as it does not require the evaluation of the entire

[6]Simple references to *lookup table* entries can be used in the implementation instead of real memory pointers (see [6, 49]). This makes the implementation possible also in programming languages that do not allow direct manipulation of memory pointers, like for instance Java or MatLab.

[7]The term "ancestors" here is a bit abused to designate not only the parents but also the random trees used to build an individual by crossover or mutation.

trees. Indeed, evaluating each individual requires (except for the initial generation) a constant time, which is independent from the size of the individual itself. In terms of memory, tables P and $\mathcal{M}$ grow during the run. However, table P adds a maximum of $2 \times n$ rows per generation (if all new individuals are created by mutation) and table $\mathcal{M}$ (which contains only memory pointers) adds a maximum of n rows per generation. Even if we never erase the "ex-ancestors" from these tables (and never reuse random trees, which is also possible), we can manage them efficiently for several thousands of generations.

Let us briefly consider the cost in terms of time and space of evolving a population of n individuals for g generations. At every generation, we need $O(n)$ space to store the new individuals. Thus, we need $O(ng)$ space in total. Since we need to do only $O(1)$ operations for any new individual (since the fitness can be computed using the fitness of the parents), the time complexity is also $O(ng)$. Thus, we have a linear space and time complexity with respect to population size and number of generations.

The final step of the algorithm is performed after the end of the last generation. In order to reconstruct the individuals, we may need to "unwind" our compact representation and make the syntax of the individuals explicit. Therefore, despite performing the evolutionary search very efficiently, in the end we may not avoid dealing with the large trees that characterize the standard implementation of geometric semantic operators. However, most probably we will only be interested in the best individual found, so this unwinding (and recommended simplification) process may be required only once, and it is done offline after the run is finished. This greatly contrasts with the solution proposed by Moraglio et al. of building and simplifying *every* tree in the population at each generation online with the search process. If we are not interested in the form of the optimal solution, we can avoid the "unwinding phase" and we can evaluate an unseen input with a time complexity equal to $O(ng)$. In this case the individual is used as a "black-box" which, in some cases, may be sufficient.

Excluding the time needed to build and simplify the best individual, the proposed implementation allowed us to evolve populations for thousands of generations with a considerable speed up with respect to standard GP.

Example. Let us consider the simple initial population P shown in Table (a) of Fig. 11 and the simple pool of random trees that are added to P as needed, shown in Table (b). For simplicity, we will generate all the individuals in the new population (that we call P' from now on) using only crossover, which will require only this small amount of random trees. Besides the representation of the individuals in infix notation, these tables contain an identifier (Id) for each individual ($T_1, \ldots, T_5$ and $R_1, \ldots, R_5$). These identifiers will be used to represent the different individuals, and the individuals created for the new population will be represented by the identifiers $T_6, \ldots, T_{10}$.

The individuals of the new population P' are simply represented by the set of entries exhibited in Table (c) of Fig. 11. This table contains, for each new individual, a *reference* to the ancestors that have been used to generate it and the name of the operator used to generate it (either "crossover" or "mutation"). For example, the individual T_6 is generated by the crossover of T_1 and T_4 and using the random tree R_1.

Let us assume that now we want to reconstruct the genotype of one of the individuals in P', for example T_{10}. The tables in Fig. 11 contain all the information needed to do that. In particular, from table (c) we learn that T_{10} is obtained by crossover between T_3 and T_4, using random tree R_5. Thus, from the definition of geometric semantic crossover, we know that it will have the following structure: $(T_3 \cdot R_5) + ((1 - R_5) \cdot T_4)$. The remaining Tables (a) and (b), that contain the syntactic structure of T_3, T_4, and R_5, provide us with the rest of the information we need to completely reconstruct the syntactic structure of T_{10}, which is:

$$((x_3 + x_4 - 2\,x_1) \cdot (2\,x_1)) + ((1 - (2\,x_1)) \cdot (x_1\,x_3))$$

and upon simplification becomes:

$$-x_1\,(4\,x_1 - 3\,x_3 - 2\,x_4 + 2\,x_1\,x_3).$$

6 Some Experimental Results

The literature reports on various results obtained on several different applicative domains using the implementation of GSGP presented above. In this section, a subset of those results is briefly discussed. The objective of this section is only to give the reader an idea of the quality of the results that can be obtained using GSGP. We do not intend in any way to be exhaustive about the results that have been obtained. For a deeper review of the literature, the reader is referred to [50] and the references cited in Sect. 7.

All the applications presented in this section are real-life symbolic regression problems. Table 1 summarizes the main characteristics of each one of them. As the table shows, we are using 6 different problems. The first three of them (%F, %PPB and LD50) are problems in pharmacokinetics and they consist in the prediction of a pharmacokinetic parameter (bioavailability, protein binding level and toxicity respectively) as a function of some molecular descriptors of a potential new drug. The PEC dataset has the objective of predicting energy consumption in one day as a function of several different types of data relative to the previous days. The PARK dataset contains data about a set of patients of the Parkinson's disease and the target value is a measure of the severity of the disease, according to a standard measure. Finally, the CONC dataset has the objective of predicting concrete strength as a function of a set of parameters that characterize a concrete mixture. For different reasons, all these problems are considered important in their respective applicative domains, and they have been studied so far using several different computational intelligence methods.

The results that have been obtained concerning a comparison between GSGP and standard GP (ST-GP from now on) on these problems are presented in Fig. 12 (%F, %PPB and LD50 datasets) and Fig. 13 (PEC, PARK and CONC datasets).

Table 1 The main characteristics of the test problems used in the experiments presented in this chapter

Dataset name (ID)	# Features	# Instances	Objective
%F	241	359	Predicting the value of human oral bioavailability of a candidate new drug as a function of its molecular descriptors
%PPB	626	131	Predicting the value of the plasma protein binding level of a candidate new drug as a function of its molecular descriptors
LD50	626	234	Predicting the value of the toxicity of a candidate new drug as a function of its molecular descriptors
PEC	45	240	Predicting the value of energy consumption in one day as a function of a set of meteorologic data, and other kinds of data, concerning that day
PARK	18	42	Predicting the value of Parkinsons disease severity as a function of a set of patients data
CONC	8	1028	Predicting the value of concrete strength as a function of a set of feature of concrete mixtures

The plots in these figures report, for each problem, the results obtained on the training set (leftmost plot) and on the test set (rightmost plot). A detailed discussion of the parameters used in both GP systems in these experiments is beyond the objective of this chapter, but they can be found in the relative references quoted, for each one of the applications, in Sect. 7. What we can generally observe from Figs. 12 and 13 is that GSGP is able to consistently outperform ST-GP both on the training set and on the test set for all the considered problems. A statistical study of these results is also beyond the scope of the chapter, but the respective references contain such a statistical study and indicate that all these results are statistically significant.

6.1 *Optimization and Generalization Ability of GSGP. A Discussion*

The good results that GSGP has obtained on training data were expected: the geometric semantic operators induce an unimodal fitness landscape, which facilitates evolvability. On the other hand, on a first analysis, it has been a surprise to observe the excellent results that have been obtained on test data. These results even appeared a bit counterintuitive at a first sight: we were expecting that the good evolvability on training data would entail an overfitting of those data. However, in order to give an interpretation to the excellent generalization ability shown by GSGP, one feature of geometric semantic operators was realized that was not so obvious previously. Namely:

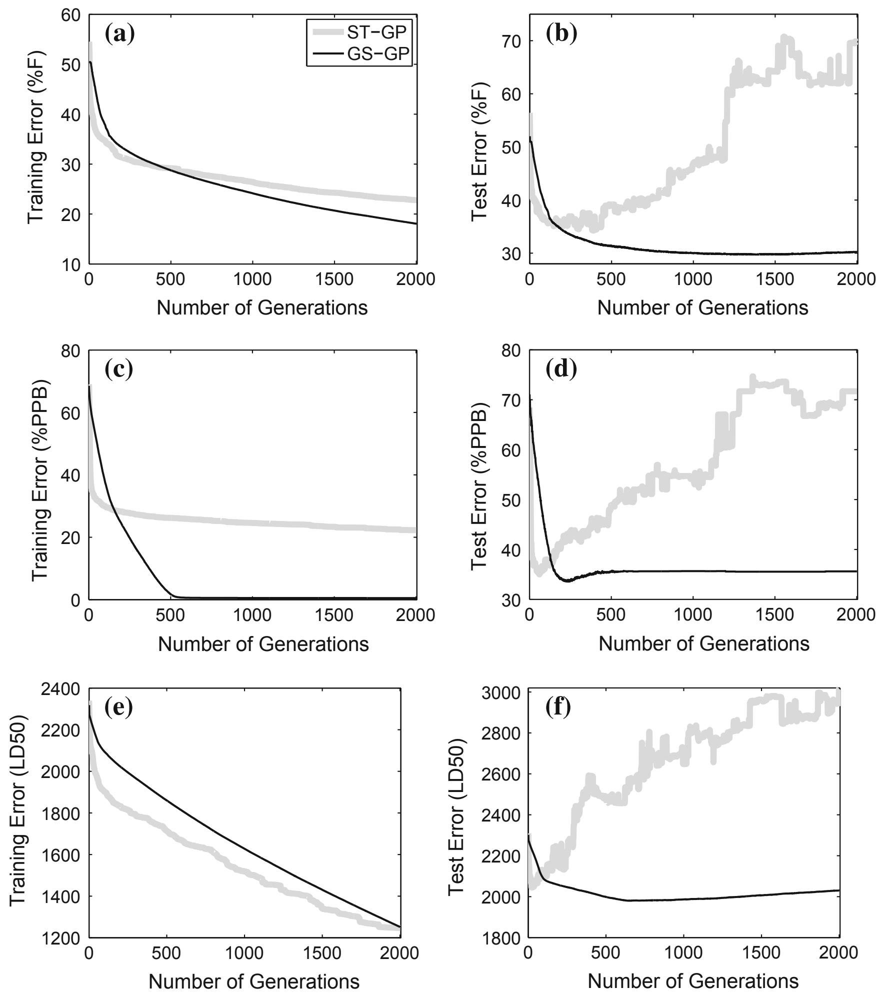

Fig. 12 Experimental comparison between standard GP (ST-GP) and Geometric Semantic GP (GS-GP). **a** %F problem, results on the training set; **b** %F problem, results on the test set; **c** %PPB problem, results on the training set; **d** %PPB problem, results on the test set; **e** LD50 problem, results on the training set; **f** LD50 problem, results on the test set

the geometric properties of geometric semantic operators hold independently of the data on which individuals *are evaluated, and thus they hold also on test data!*

In other words, for instance geometric semantic crossover produces an offspring that stands between the parents also in the semantic space induced by test data. As a direct implication, following exactly the same argument as Moraglio et al. [32], each offspring is, in the worst case, not worse than the worst of its parents on the test set. Analogously, as it happens for training data, geometric semantic mutation produces

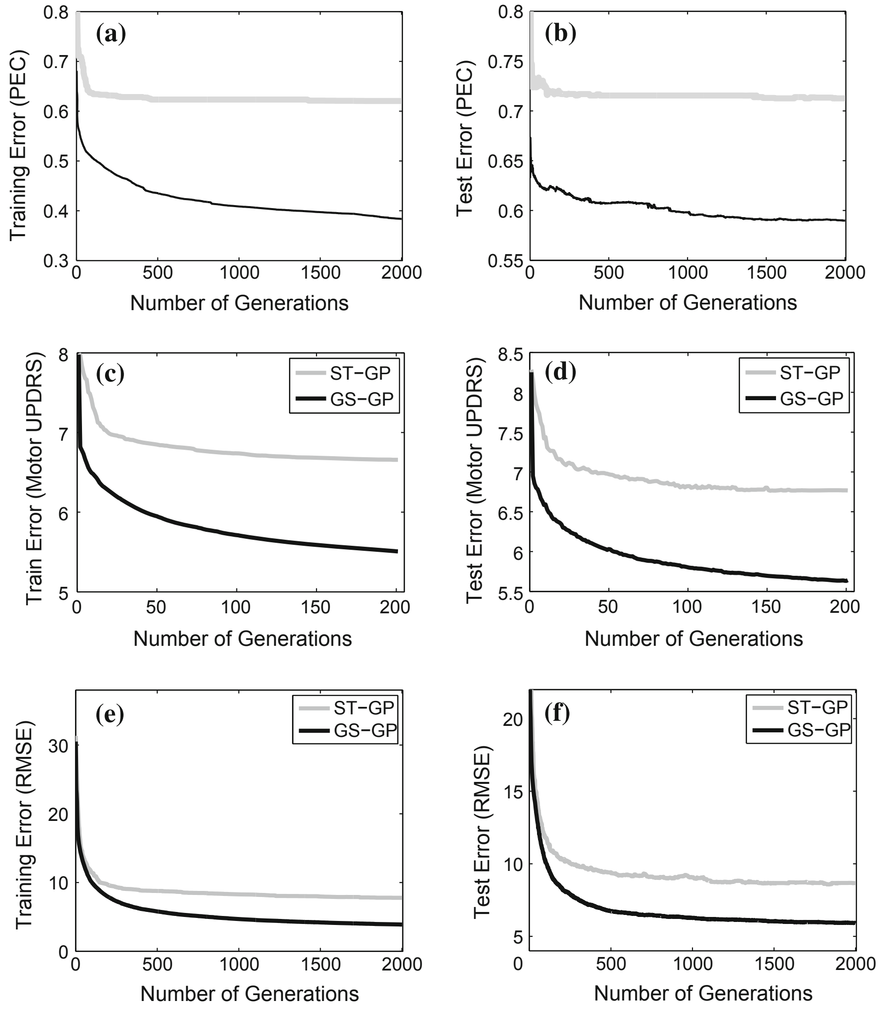

Fig. 13 Experimental comparison between standard GP (ST-GP) and Geometric Semantic GP (GS-GP). **a**: PEC problem, results on the training set; **b**: PEC problem, results on the test set. **c**: PARK problem, results on the training set; **d**: PARK problem, results on the test set. **e**: CONC problem, results on the training set; **f**: CONC problem, results on the test set

an offspring that is a “weak” perturbation of its parent also in the semantic space induced by test data (and the maximum possible perturbation is, again, expressed by the *ms* step).

The immediate consequence for the behaviour of GSGP on test data is that, while geometric semantic operators do not guarantee an improvement in test fitness each time they are applied, they at least guarantee that the possible worsening of the test fitness is bounded (by the test fitness of the worst parent for crossover, and by *ms*

Table 2 An experimental comparison between GSGP (last line), ST-GP (second to last line) and other machine learning strategies on the CONC dataset. The leftmost column contains the name of the method, the middle one the results obtained on the training set at termination, and the rightmost column contains the results obtained on the test set. Root Mean Square Error (RMSE) results are reported

Method	Train	Test
Linear regression [52]	10.567	10.007
Square regression [43]	17.245	15.913
Isotonic regression [21]	13.497	13.387
Radial basis function network [20]	16.778	16.094
SVM polynomial kernel (1°) [42]	10.853	10.260
SVM polynomial kernel (2°)	7.830	7.614
SVM polynomial kernel (3°)	6.323	6.796
SVM polynomial kernel (4°)	5.567	6.664
SVM polynomial kernel (5°)	4.938	6.792
Artificial neural networks [20]	7.396	7.512
Standard GP	7.792	8.67
Geometric semantic GP	3.897	5.926

for mutation). In other words, *geometric semantic operators help control overfitting*. Of course, overfitting may still happen for GSGP (as it happens, slight but visible, for instance in plots (d) and (f) of Fig. 12, reporting the results on %PPB and LD50 respectively), but there are no big “jumps” in test fitness like the ones observed for ST-GP.

It is worth remarking that, without the novel implementation presented in Sect. 5, that allowed us to use GSGP on these complex real-life problems, this interesting property would probably have remained unnoticed.

Table 2 also reports an experimental comparison between GSGP and a wide set of non-evolutionary machine learning state-of-the-art methods on the CONC dataset. Analogous results can be found in literature also for all the other studied problems, and they show that GSGP is able to outperform all the other techniques both on training and test data.

Before terminating this section, it is worth repeating that the objective of this section (and of this chapter in general) is not a deep discussion of the experimental results that have been obtained using GSGP so far. Indeed, this section deliberately misses many details about the experimental setting and the statistical validation of these results. On the other hand, the objective of the section is showing that GSGP can be considered a new machine learning method, able to compete with the state-of-the-art methods on complex real-life applications. Given the young age of GSGP and the amount of research that is still needed in this field, according to the author’s aim, these results should stimulate the reader to deepen her knowledge on the subject, reading, for instance, the several references that contain details on these and other experimental studies. The next section is conceived with the objective of advising the reader on bibliographic references that can be useful to deepen one's competence on GSGP.

7 How to Deepen the Subject

Because of the vast set of studies that have appeared so far on GSGP and because of the obvious space limitations of this publication, this section cannot contain an exhaustive list of references on the subject. On the other hand, the bibliographic references cited in this section have to be interpreted as the material that I personally advice to read in order to deepen one's competencies on GSGP. One may want to consult the survey in [51] for a deeper discussion of the state of the art in semantic-based GP.

The first step for deepening one's knowledge on GSGP and go beyond this introductory chapter is, probably, reading the original paper by Moraglio and coauthors [32], in which geometric semantic operators were first introduced. This is a real "must" if one wants to become an expert in GSGP. It contains several details about these operators that were not discussed here and that are very useful to deeply understand these operators and their philosophy. Nevertheless, the reader should be warned that reference [32] is much more formal and technical (and thus, I believe, more difficult to understand for beginners) than this chapter. It is, thus, in my opinion, appropriate to make sure that one has fully understood this chapter before going on and reading [32].

Among several other pieces of information that can be found in [32], one may discover that versions of the geometric semantic operators exist also for domains that are different from symbolic regression (the only one that was discussed in this chapter). For instance, geometric semantic operators have been defined also for boolean problems and for classification [32]. The number of applications that have appeared so far on these domains is incomparably smaller than the number of existing applications in symbolic regression. For this reason, applying geometric semantic operators on boolean problems and classification problems is still a hot topic, and new research in this direction is definitely in demand.

Other references that should be part of the know how of any researcher in GSGP are the papers published on the subject by Moraglio and coauthors later than [32]. For instance, the following references are strongly advised:

- Reference [33] contains a definition of a new geometric semantic mutation that generates individuals of exactly the same size as the parents. Nevertheless, this mutation is specific for the restricted set of basis functions. The paper also presents a runtime analysis of this system and, as several others, contributes to the idea that mutation is the leading genetic operator in GSGP.
- Reference [28] defines and analyses a mutation-based GSGP for the class of classification tree learning problems.

- Reference [34] presents a runtime analysis of GSGP that uses only mutation for boolean problems.
- Reference [31] presents an implementation of GSGP that, similarly to the one discussed in this chapter, is based on tracing the ancestry of individuals instead of storing them. However, the implementation in [31] does not explicitly build and maintain a data structure, but uses higher-order functions and memorization to achieve the same effect, leaving the burden of book-keeping to the compiler.
- Additionally, one may want to read Moraglio's PhD thesis [30], that proposes a unified vision of EAs based on the concept of geometry.

The work of Krawiec and coauthors also represents an excellent reading material in order to deepen one's knowledge on GSGP. More in particular:

- One should definitely read the book [26], containing a complete analysis of semantic-based studies in GP.

Also, I advise reading of:

- Reference [37], in which properties are discussed of guarantees of progress for the search operators of GSGP.
- Reference [39], where a semantic backpropagation is proposed for designing GP search operators.
- Reference [36], which contains a deep study of different metrics, and their different properties, in GSGP.
- Reference [38], where several kinds of geometric semantic crossovers are discussed and compared.

One of the most promising recent trends is to hybridize GSGP with local search strategies, with the objective of speeding up the training optimization, without compromising generalization ability. The interested reader is referred to [11], where a hybrid version of GSGP and local search is presented and applied to some of the real life applications used as case studies in this chapter.

Last but not least, the reader may be interested in understanding more about the real-life applications that have been solved so far by GSGP. Those applications are:

- Prediction of the relative position of computer tomography slices [10].
- Prediction of the proteins tertiary structure [13].
- Analysis of reviews of Amazon's products [13].
- Forecasting of energy consumption [8, 9, 12].
- Electoral redistricting [5].
- Prediction of pharmacokinetic parameters [47, 51].
- Prediction of the unified Parkinson's disease rating scale assessment [15].
- Prediction of high performance concrete strength [14].
- Anticoagulation level prediction in pharmacogenetics [4].
- Classification of land cover/land use [7].
- Prediction of forest aboveground biomass [44].
- Prediction of vessels' trajectories at sea for maritime safety and security [48]

8 Conclusions and Future Work

Geometric semantic operators represent one of the hottest topic in genetic programming. Even though, in their four years of existence, they have already allowed us to obtain interesting applicative results, they are still far from being accepted as a new state-of-the-art technology by the vast machine learning community. Improvements are needed to reach this goal, and consequently a lot of research is still in demand. The collaboration of young and motivated researchers is crucial for the future success of geometric semantic genetic programming, and this is the fundamental motivation for the existence of this chapter.

This chapter contained a basic, and hopefully easy to understand, introduction to geometric semantic genetic operators, and the focus was on explaining as clearly as possible the reason why these operators induce a unimodal fitness landscape, characterized by the total absence of locally suboptimal solutions. This is the characteristic that should give geometric semantic genetic programming a competitive advantage and that makes geometric semantic genetic programming so exciting and rather "novel" compared to several existing machine learning systems. The objective of this chapter is to stimulate the readers to deepen the subject and possibly also to invest some of their time contributing to this young and promising research field.

Many are the possible directions that can be taken to contribute to the field, but in my personal vision they can be mainly classified into two possible tracks of activity: defining new geometric semantic operators and improving the existing ones. New genetic operators should have the same interesting properties that allow the existing operators to induce unimodal fitness landscapes, but should also not (or at least not always) generate offspring that are larger than their parents. This could allow us to generate models that are also easy to visualize and interpret, a very important requirement in several applicative domains. Concerning the second possible track, one may consider investigating methods of automatic simplification of expressions in order to make the models generated by geometric semantic genetic programming smaller, and thus more compact and more readable. In addition, the use of hardware and software high performance computing technologies should be another possible area to investigate, with the objective of incrementing the power of geometric semantic genetic programming. For instance, it is not difficult to imagine an implementation of geometric semantic genetic programming on graphical processing unit; in fact, geometric semantic genetic programming is mainly a vectorial system, given that the concept of semantics here is coded by a vector of values.

Acknowledgments My heartfelt acknowledgment goes to Sara Silva and Mauro Castelli, who shared with me the work on geometric semantic genetic programming and much much more! I also dearly thank Leonardo Trujillo and Oliver Schütze for inviting me to the NEO 2015 workshop. It has been an unforgettable experience.

References

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. Wiley, New York (1989)
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics). Princeton University Press, Princeton (2007)
3. Back, T., et al. (eds.): Handbook of Evolutionary Computation, 1st edn. IOP Publishing Ltd., Bristol (1997)
4. Castelli, M., Castaldi, D., Giordani, I., Silva, S., Vanneschi, L., Archetti, F., Maccagnola, D.: An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics. In: Correia, L., et al. (eds.) Progress in Artificial Intelligence. Lecture Notes in Computer Science, vol. 8154, pp. 78–89. Springer, Berlin (2013)
5. Castelli, M., Henriques, R., Vanneschi, L.: A geometric semantic genetic programming system for the electoral redistricting problem. Neurocomputing **154**, 200–207 (2015)
6. Castelli, M., Silva, S., Vanneschi, L.: A C++ framework for geometric semantic genetic programming. Genet. Program. Evol. Mach. 1–9 (2014)
7. Castelli, M., Silva, S., Vanneschi, L., Cabral, A., Vasconcelos, M., Catarino, L., Carreiras, J.: Land cover/land use multiclass classification using gp with geometric semantic operators. In: Esparcia-Alczar, A. (ed.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 7835, pp. 334–343. Springer, Berlin (2013)
8. Castelli, M., Trujillo, L., Vanneschi, L.: Energy consumption forecasting using semantic-based genetic programming with local search optimizer. Comput. Intell. Neurosci. Article ID 971908, 8 p. (2015). http://dx.doi.org/10.1155/2015/971908
9. Castelli, M., Trujillo, L., Vanneschi, L., Popovic, A.: Prediction of energy performance of residential buildings: A genetic programming approach. Energy Build. **102**, 67–74 (2015)
10. Castelli, M., Trujillo, L., Vanneschi, L., Popovic, A.: Prediction of relative position of CT slices using a computational intelligence system. Appl. Soft Comput. (2015, in press)
11. Castelli, M., Trujillo, L., Vanneschi, L., Silva, S., Z-Flores, E., Legrand, P.: Geometric semantic genetic programming with local search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15, pp. 999–1006. ACM, New York, NY, USA (2015)
12. Castelli, M., Vanneschi, L., Felice, M.D.: Forecasting short-term electricity consumption using a semantics-based genetic programming framework: The south italy case. Energy Econ. **47**, 37–41 (2015)
13. Castelli, M., Vanneschi, L., Manzoni, L., Popovic, A.: Semantic genetic programming for fast and accurate data knowledge discovery. Swarm Evol. Comput. (2015, in press)
14. Castelli, M., Vanneschi, L., Silva, S.: Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. Expert Syst. Appl. **40**(17), 6856–6862 (2013)
15. Castelli, M., Vanneschi, L., Silva, S.: Prediction of the unified parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. Expert Syst. Appl. **41**(10), 4608–4616 (2014)
16. Darwin, C.: On the Origin of Species by Means of Natural Selection. Murray, London (1859) or the Preservation of Favored Races in the Struggle for Life
17. Dick, G.: Improving geometric semantic genetic programming with safe tree initialisation. In: Machado, P., et al. (eds.) 18th European Conference on Genetic Programming. LNCS, vol. 9025, pp. 28–40. Springer, Copenhagen, 8–10 April 2015
18. Fan, W., Bifet, A.: Mining big data: current status, and forecast to the future. SIGKDD Explor. Newsl. **14**(2), 1–5 (2013)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
20. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall, Upper Saddle River (1999)

21. Hoffmann, L.: Multivariate Isotonic Regression and Its Algorithms. Wichita State University, College of Liberal Arts and Sciences, Department of Mathematics and Statistics (2009)
22. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Genetic Programming, Proceedings of EuroGP'2003. LNCS, vol. 2610, pp. 70–82. Springer (2003)
23. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann Publishers Inc., San Francisco (2001)
24. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
25. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
26. Krawiec, K.: Behavioral Program Synthesis with Genetic Programming. Studies in Computational Intelligence, vol. 618. Springer, Berlin (2016)
27. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Berlin (2002)
28. Mambrini, A., Manzoni, L., Moraglio, A.: Theory-laden design of mutation-based geometric semantic genetic programming for learning classification trees. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 416–423 (2013)
29. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York (1990)
30. Moraglio, A.: Towards a Geometric Unification of Evolutionary Algorithms. Ph.D. thesis, Department of Computer Science, University of Essex, UK (2007)
31. Moraglio, A.: An efficient implementation of GSGP using higher-order functions and memoization. In: Johnson, C., et al. (eds.) Semantic Methods in Genetic Programming, Ljubljana, Slovenia, 13 Sept. 2014. Workshop at Parallel Problem Solving from Nature 2014 conference (2014)
32. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Parallel Problem Solving from Nature, PPSN XII (part 1). Lecture Notes in Computer Science, vol. 7491, pp. 21–31. Springer (2012)
33. Moraglio, A., Mambrini, A.: Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In: Blum, C., et al. (eds.) Proceedings of the 15th annual international conference on Genetic and Evolutionary Computation. GECCO '13, pp. 989–996. ACM, New York, NY, USA (2013)
34. Moraglio, A., Mambrini, A., Manzoni, L.: Runtime analysis of mutation-based geometric semantic genetic programming on boolean functions. In: Neumann, F., De Jong, K. (eds.) Foundations of Genetic Algorithms, pp. 119–132. ACM, Adelaide, Australia, 16–20 January 2013
35. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. World Scientific, Singapore (2006)
36. Pawlak, T.P., Krawiec, K.: Progress properties and fitness bounds for geometric semantic search operators. Genetic Programming and Evolvable Machines (Online first)
37. Pawlak, T.P., Krawiec, K.: Guarantees of progress for geometric semantic genetic programming. In: Johnson, C., et al. (eds.) Semantic Methods in Genetic Programming, Ljubljana, Slovenia, 13 Sept. 2014. Workshop at Parallel Problem Solving from Nature 2014 conference (2014)
38. Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and comparative analysis of geometric semantic crossovers. Genet. Progr. Evol. Mach. **16**(3), 351–386 (2015)
39. Pawlak, T.P., Wieloch, B., Krawiec, K.: Semantic backpropagation for designing search operators in genetic programming. IEEE Trans. Evol. Comput. **19**(3), 326–340 (2015)
40. Poli, R., Langdon, W.B., Mcphee, N.F.: A field guide to genetic programming (2008)
41. Richter, H., Engelbrecht, A. (eds.): Recent Advances in the Theory and Application of Fitness Landscapes. Emergence. Complexity and Computation, vol. 6. Springer, Berlin (2014)
42. Schölkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Adaptive computation and machine learning. MIT Press (2002)
43. Seber, G., Wild, C.: Nonlinear Regression. Wiley Series in Probability and Statistics. Wiley (2003)

44. Silva, S., Ingalalli, V., Vinga, S., Carreiras, J., Melo, J., Castelli, M., Vanneschi, L., Gonalves, I., Caldas, J.: Prediction of forest aboveground biomass: An exercise on avoiding overfitting. In: Esparcia-Alczar, A. (ed.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 7835, pp. 407–417. Springer, Berlin Heidelberg (2013)
45. Tomassini, M., Vanneschi, L., Collard, P., Clergue, M.: A study of fitness distance correlation as a difficulty measure in genetic programming. Evol. Comput. **13**(2), 213–239 (2005)
46. Vanneschi, L.: Theory and Practice for Efficient Genetic Programming. Ph.D. thesis, Faculty of Sciences, University of Lausanne, Switzerland (2004)
47. Vanneschi, L.: Improving genetic programming for the prediction of pharmacokinetic parameters. Memet. Comput. **6**(4), 255–262 (2014)
48. Vanneschi, L., Castelli, M., Costa, E., Re, A., Vaz, H., Lobo, V., Urbano, P.: Improving maritime awareness with semantic genetic programming and linear scaling: prediction of vessels position based on ais data. In: Mora, A.M., Squillero, G. (eds.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 9028, pp. 732–744. Springer International Publishing (2015)
49. Vanneschi, L., Castelli, M. Manzoni, L., Silva, S.: A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013. LNCS, vol. 7831, pp. 205–216. Springer, Vienna, Austria, 3–5 April 2013
50. Vanneschi, L., Castelli, M., Silva, S.: A survey of semantic methods in genetic programming. Genet. Progr. Evol. Mach. **15**(2), 195–214 (2014)
51. Vanneschi, L., Silva, S., Castelli, M., Manzoni, L.: Geometric semantic genetic programming for real life applications. In: Riolo, R., et al. (eds.) Genetic Programming Theory and Practice XI, Genetic and Evolutionary Computation. Springer US, Computer Science Collection, 2013. Invited article (2013, to appear)
52. Weisberg, S.: Applied Linear Regression. Wiley, Wiley Series in Prob. and Stat (2005)
53. Wright, S.: The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: Jones, D.F. (ed.) Proceedings on the Sixth International Congress on Genetics, vol. 1, pp. 356–366 (1932)