

# Free Lunches for Function and Program Induction

Riccardo Poli  
School of Computer Science  
and Electronic Engineering  
University of Essex, UK  
rpoli@essex.ac.uk

Mario Graff  
School of Computer Science  
and Electronic Engineering  
University of Essex, UK  
mgraft@essex.ac.uk

Nicholas Freitag McPhee  
Division of Science and  
Mathematics  
University of Minnesota,  
Morris, USA  
mcphee@morris.umn.edu

## ABSTRACT

In this paper we prove that for a variety of practical problems and representations, there is a free lunch for search algorithms that specialise in the task of finding functions or programs that solve problems, such as genetic programming. In other words, not all such algorithms are equally good under all possible performance measures. We focus in particular on the case where the objective is to discover functions that fit sets of data-points – a task that we will call symbolic regression. We show under what conditions there is a free lunch for symbolic regression, highlighting that these are extremely restrictive.

## Categories and Subject Descriptors

I.2.2 [Artificial Intelligence]: Automatic Programming

## General Terms

Algorithms, Performance

## Keywords

Genetic Programming, Theory, No-free Lunch

## 1. INTRODUCTION

Informally speaking, the no-free-lunch theory (NFL) originally proposed by Wolpert and Macready [10] states that, when evaluated over all possible problems, all algorithms are equally good or bad irrespective of our evaluation criteria. In the last decade there have been a variety of results which have refined and specialised NFL (see [9] for a comprehensive recent review).

One such result states that if one selects a set of fitness functions that are *closed under permutation* (more on this below) then the expected performance of any search algorithm over that set of problems is constant, i.e., it does not depend on the algorithm we choose [6] nor the chosen performance measure.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FOGA '09, January 9–11, 2009, Orlando, Florida, USA.  
Copyright 2009 ACM 978-1-60558-414-0/09/01 ...\$5.00.

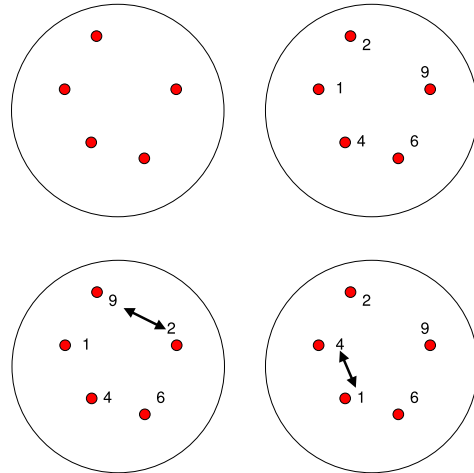


Figure 1: A sample search space (top left), a problem (i.e., an assignment of fitness to the elements of the search space) (top right), and two sample permutations of the fitness function (bottom).

What does it mean for a set of functions to be closed under permutation? A fitness function can be seen as an assignment of fitness values to the elements of the search space, as exemplified graphically in Figure 1(top). A permutation of a fitness function is simply a rearrangement of the fitness values originally allocated to the objects in the search space. Examples of permutations are shown at the bottom of Figure 1. If we enumerate the elements of a search space according to some scheme, we can then represent a fitness function as a vector that stores the fitness associated to each point of the space. For example, if we enumerate the five points in Figure 1(top left) in clockwise order and starting from the top, the fitness function in Figure 1(top right) can be represented as indicated in the leftmost vector in Figure 2. In this case, permuting a fitness function simply means shuffling the elements of the vector representing it. Some permutations of our sample fitness function are shown in Figure 2. A set of problems/fitness functions is closed under permutation, if for every function in the set all possible shuffles of that function are also in the set. If all the possible permutations (including the identity permutation) of the elements of the leftmost vector in Figure 2 were considered, then the resulting set of 120 fitness functions would be closed under permutation.

f1	2	f1	1	f1	4	f1	6	f1	9	... (120) ....
f2	1	f2	2	f2	1	f2	1	f2	1	
f3	4	f3	4	f3	2	f3	4	f3	4	
f4	6	f4	6	f4	6	f4	2	f4	6	
f5	9	f5	9	f5	9	f5	9	f5	2	

**Figure 2: A fitness function seen as a vector and its permutations.**

In formulae, [6] states that if  $\mathcal{F}$  is a set of fitness functions closed under permutation, we have that

$$\sum_{f \in \mathcal{F}} P(f, a_1) = \sum_{g \in \mathcal{F}} P(g, a_2) \quad (1)$$

for any pair of pair of (non-resampling) search algorithms  $a_1$  and  $a_2$  and for any performance measure  $P$ . Furthermore, [6] showed that the connection between closure and NFL is an “if and only if” one. That is, it is also the case that two arbitrary algorithms will have identical performance over a set of functions only if that set of functions is closed under permutation.

An even more general version of NFL is that presented by Igel and Toussaint in [2]. When the set  $\mathcal{F}$  is obtained by permuting a particular function  $f$ , all of the functions in  $\mathcal{F}$  present the same histogram of fitness values. This histogram is obtained by applying  $f$  to all possible elements of its domain and recording the co-domain values returned by the function. The set  $\mathcal{F}$  can be closed under permutation if and only if it includes all the functions which present a particular fitness histogram. Naturally, it is possible for a set  $\mathcal{F}$  to include functions with more than one fitness histogram. For the set to be closed under permutation it must include all functions with each such histogram. Igel and Toussaint [2] proved that NFL applies to a set if and only if all the functions with the same histogram are drawn from the set with identical probability, although different probabilities can be associated to different histograms. A similar result was proved by Streeter [7].

We should note that if the algorithms that are to be compared are known *a priori*, then it is possible to construct sets of functions on which NFL holds but which are not closed under permutation [8]. However, in this paper we will concentrate on the “strong” version of NFL, i.e., where we don’t have any knowledge nor make any assumptions about either the algorithms or the performance measures being used.

Among the many extensions of NFL to a variety of domains, Woodward and Neil [12] have made some progress in assessing the applicability of NFL to the search spaces explored by genetic programming (GP). In particular, they argued that there is a free lunch in a search space whenever there is a non-uniform many to one genotype-phenotype mapping, and that the mapping from syntax to functionality in GP is one such mapping. The reason why NFL would not normally be applicable to search in program spaces is that there are many more programs than functionalities and that not all functionalities are equally likely. As an example, Woodward [11] considered the search space generated by the primitive set  $\{a, b, +\}$  where  $a$  and  $b$  are terminals and  $+$  is the ordinary addition operation. Assuming only trees of up to 3 nodes are allowed, the search space contains

the programs  $a, b, a + a, a + b, b + a$  and  $b + b$ . However, irrespective of the set of test cases, the functionality of  $a + b$  is always identical to the functionality of  $b + a$ . Therefore, if we interpret syntax trees as genotypes and functionalities as phenotypes, the GP genotype-phenotype mapping is many-to-one and non-uniform, which invalidates NFL. This is because an algorithm that always visited  $a + b$  or  $b + a$  last in its search would perform better than an algorithm that visited, say,  $a$  last if the adopted performance measure is the number of fitness evaluations required to find the optimum.

Beyond this interesting counterexample, to show that, in general, not all functionalities are equally likely in program search spaces, Woodward and Neil [12] referred to Langdon’s results on the limiting distribution of functionality (as longer and longer programs are considered the proportion of programs with any particular functionality reaches a limit) [4] and to the universal distribution [3] (informally this states that there are many more programs with a simple functionality than programs with a complex one). However, there is a formal proof of the former result only for the case of register based machines, while, for spaces where programs are represented using syntax trees, there is only a semi-formal argument indicating that a limiting distribution *should* exist [4]. The latter result instead applies to Turing complete languages, i.e., to programs with memory and loops. Being Turing-complete GP a real rarity (due to the complexities of avoiding non-terminating programs or programs with very long run times), essentially, there is no formal proof of under what conditions NFL holds or, conversely, there can be a free lunch, for real GP applications and mainstream forms of search over program spaces. In this paper we want to rectify this situation. In addition, we want to understand whether a non-uniform many-to-one genotype-phenotype mapping is the only condition under which NFL breaks down when searching program spaces (whether with GP or some other technique).

Before we conclude this brief history of NFL, we should point out that Igel and Toussaint [1] were also able to show that if one considers all possible sets of functions with a given domain and a given co-domain, in most conditions the sets that are closed under permutation represent a tiny fraction of the whole. This would suggest that NFL does only really rarely apply. While this is encouraging, it is not clear what kind of probability distribution characterises realistic problem sets (e.g., it might well be that in reality sets that are closed under permutation occur much more frequently than expected). So, for any given class of problems, it may be very difficult to know whether the class is within the tiny fraction of sets that are closed under permutation. As we will see for the case of program search spaces, there are cases where a problem class is closed under permutation, and others where the problem class is not.

In most cases it is also difficult to know whether the probability distribution over problems satisfies the symmetries described in [2] and summarised above. A case where the original formulation of NFL is guaranteed to break down even if the set of functions considered is closed under permutation is the case of infinite sets of functions (the original NFL and all its refinements are limited to finite sets of functions) as shown by Streeter [7]. In the case of infinite sets the original NFL formulation does not hold because it relies on the assumption that all functions are equally likely. Naturally, with infinite sets it is impossible to sample a set

uniformly at random. Streeter found that NFL is still applicable if the probability distribution over functions from the set is such that all functions that are permutations of a particular function have identical probability of being drawn from the set. This means that the probability of picking a particular function  $f$  from the set can only depend on the domain of  $f$  and the histogram of fitness values of  $f$  obtained when applying  $f$  to all possible elements of its domain. As Streeter pointed out, while there may be an argument for assuming that all functions have equal probability of being the case in finite search spaces (this essentially amounts to saying that we have no information on the problem), there is no evidence to suggest that real-world problem distributions will behave (in relation to permutations of problems) as required for NFL to be applicable.

One may wonder if this result is applicable to GP, in the sense that GP is often said to explore the space of all possible recursive compositions of the primitives in its primitive set, which is, of course, an infinite space. However, we should be careful not to confuse the exploration of infinite search spaces, which in principle GP does,<sup>1</sup> with the application of GP to an infinite sets of fitness functions (or the probabilistic drawing of problems for GP from an infinite set). In this paper, we will characterise the space of problems that GP can face, and we will show it is, in practice, always finite.

The paper is organised as follows. In Section 2 we will look at the typical form of program induction fitness functions, and will show that in the case of symbolic regression there is a natural geometric interpretation for fitness functions. We will then make use of this interpretation in Section 3 to provide conditions under which NFL holds for search in program spaces. These conditions are very constraining, and in Section 4 we will then show both graphically and analytically that under very mild assumptions on the class of symbolic regression problems, NFL does not apply. We consider possible extensions of the work in Section 5. Finally, in Section 6 we draw some conclusions.

## 2. THE GEOMETRY OF SYMBOLIC REGRESSION AND PROGRAM INDUCTION

The quality (fitness, hereafter) of a program  $p$  in GP or some other program induction technique is often the result of evaluating the behaviour of  $p$  in a number of test environments (also known as “fitness cases” in the GP literature or as “examples” in the machine learning literature), assessing by how much such a behaviour deviates from a corresponding target behaviour, adding up the results of that assessment and then, optionally, performing some monotonic transformation of the sum [5]. That is

$$f(p) = h \left( \sum_{i=1}^n g(p(x_i), t(x_i)) \right) \quad (2)$$

where  $f$  is the fitness function,  $\{x_i\}$  is a set of fitness cases of cardinality  $n$ ,  $g$  is a function which evaluates the degree to which the behaviour of  $p$  (i.e., its outputs or its side effects

<sup>1</sup>In practice GP always searches a finite search space, since there are always explicit (software) or implicit (computer hardware) limitations on the size of programs that can be considered.

or both) matches a target behaviour  $t$  on each fitness case and  $h$  is a monotonic transformation (more on this below).

Of course what exactly is meant by a fitness case,  $x_i$ , the target behaviour,  $t(x_i)$ , or the behaviour of a program,  $p(x_i)$ , depends very much on the application. For example, in a robotic control application, each  $x_i$  might represent the initial state of the robot and its environment, the behaviour of a program might be the sequence of robot/environment states produced by the execution of the program for a certain number of time steps, and the target behaviour might be expressed as the target state of the robot/environment at the end of program execution or a set of states we want the robot/environment to visit. Similarly what the function  $g$  does to compute the degree to which actual and desired behaviours match depends on the application.

There are many cases, however, where users of a program induction system are only interested in program outputs and programs have no side effects, i.e., they are functions that map inputs to outputs. In this case the fitness cases  $x_i$ 's are either scalars or vectors representing program inputs. Similarly, the output produced by a program is either a scalar (a very typical case in GP) or a vector of program-outputs (a typical case in artificial neural networks). In the case of scalar outputs, almost invariably the function  $g$  takes the form  $g(a, b) = |a - b|^k$  for  $k = 1$  or  $k = 2$ . When  $k = 1$  typically  $h$  is the identity function, so Equation (2) computes the sum of absolute errors, or it is scaling/normalisation function (e.g., fitness may be the mean absolute error). When  $k = 2$ ,  $h$  may be the square root function (in which case Equation (2) is the sum of squared errors), possibly with some normalisation (e.g., to produce a root mean squared error). If outputs and targets are vectors, very often  $g(a, b) = \|a - b\|^2$ . Note that in all these examples  $g$  is a sort of error measure which, therefore, we want to minimise. In some applications, however, the bigger  $g$  the better. These are also covered by our arguments.<sup>2</sup>

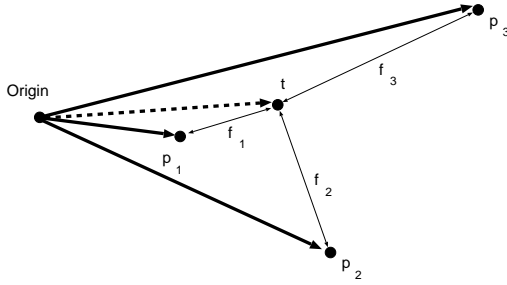
For simplicity, in the paper we will mostly concentrate on the case of vector inputs but scalar outputs, although the theory can easily be extended to more general cases. So, we are interested in finding functions without side effects that map points in  $\mathbb{R}^m$  to points in  $\mathbb{R}$ ; a case that we will term *symbolic regression* hereafter.

Let us consider a finite space of programs  $\Omega = \{p_i\}_{i=1}^r$ , such as, for example, the space of all possible programs with at most a certain size (or depth, if a syntax-tree representation for programs is used), where  $r = |\Omega|$  is the cardinality of  $\Omega$ . In these conditions, a fitness function  $f$  over  $\Omega$  can be represented as a vector  $\mathbf{f} = (f_1, \dots, f_r)$  where  $f_i = f(p_i)$ . Using this vector representation for fitness functions, we can write Equation (2) as

$$\mathbf{f} = \left[ h \left( \sum_{i=1}^n g(p_1(x_i), t(x_i)) \right), \dots, h \left( \sum_{i=1}^n g(p_r(x_i), t(x_i)) \right) \right]. \quad (3)$$

As we can see from Equation (3), if  $n$  and the set of fitness cases  $\{x_i\}$  are fixed *a priori*, then a fitness function is fully determined by the value of the (ordered) set of target

<sup>2</sup>The fitness function  $f$  is often an error measure which needs to be minimised. In GP, the function  $h$  is often used also to turn a minimisation problem into a maximisation one or *vice versa* and to normalise the fitness values in the range  $[0,1]$ , through offsetting and squashing transformations. These details are unimportant for our treatment.



**Figure 3:** In symbolic regression, fitness (to be minimised) is the distance between the target behaviour  $\mathbf{t}$  and the behaviour exhibited by each program,  $p_1$ ,  $p_2$ , etc.

behaviours,  $\mathbf{T} = (t_1, t_2, \dots, t_n)$  where  $t_i = t(x_i)$ , is fixed. In symbolic regression we typically have that the elements  $t_i$  are scalars representing the desired output for each fitness case. In this case  $\mathbf{T}$  can be treated as an ordinary vector,  $\mathbf{t} = (t_1, t_2, \dots, t_n)$  in  $\mathbb{R}^n$ .

It is important to note that in most cases Equation (2), i.e., the fitness associated to a program  $p$ , can be interpreted as the distance between the vector  $\mathbf{t} \in \mathbb{R}^n$  and the vector  $\mathbf{p} = (p(x_1), p(x_2), \dots, p(x_n)) \in \mathbb{R}^n$ . That is

$$f(\mathbf{p}) = d(\mathbf{p}, \mathbf{t}), \quad (4)$$

for some function  $d$  which satisfies the axioms of a metric. This is the case, for example, if  $f$  is the sum of absolute errors, in which case it corresponds to the city-block distance. Also, if  $f$  is the root mean squared error, it is also a distance (being proportional to the Euclidean distance). In other cases, a simple transformation of  $f$  is a distance. For example, when  $f$  is the total sum of squared errors,  $\sqrt{f}$  is a distance. So, although in the remainder of the paper we will often focus on the case where  $f(\mathbf{p}) = d(\mathbf{p}, \mathbf{t})$ , most results can be extended to the more general case where  $f$  is an invertible function of a distance, as is the case of the total sum of squared errors where  $\sqrt{f(\mathbf{p})} = d(\mathbf{p}, \mathbf{t})$ .

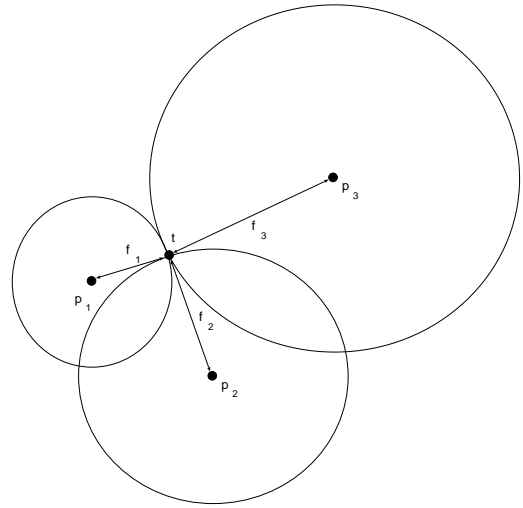
Figure 3 shows a set of programs (represented as vectors in  $\mathbb{R}^n$ ), their fitnesses and their relationship with the target vector  $\mathbf{t}$ .<sup>3</sup> Note that whenever we represent programs using their behaviour vector we are essentially focusing on the phenotype-to-fitness mapping, thereby complementing the analysis of Woodward and Neil [12] summarised in Section 1.

Using distances, Equation (3) can be rewritten more concisely as

$$\mathbf{f} = (d(\mathbf{p}_1, \mathbf{t}), d(\mathbf{p}_2, \mathbf{t}), \dots, d(\mathbf{p}_r, \mathbf{t})). \quad (5)$$

Note that if we know the fitness  $f$  of a program  $p$ , we know that the target behaviour  $\mathbf{t}$  that generated that fitness must be on the surface of a sphere centred on  $\mathbf{p}$  (the vector representation of  $p$ ) and of radius  $f$ . So, for every valid symbolic regression fitness function, the target behaviour is at the intersection of the spheres centred on the behaviour of each program in the search space. Naturally, the notion of sphere and whether the intersection is a point or a whole set (segment) depends on the distance metric  $d$  which represents our fitness function (see Figures 4 and 5).

<sup>3</sup>In the rest of the paper we will represent vectors as points whenever possible to avoid cluttering the figures.



**Figure 4:** If  $g$  measures the squared difference between two numbers, a valid fitness function requires the spheres centred on each program behaviour and with radius given by their corresponding fitness to intersect in one point: the target vector  $\mathbf{t}$ .

In the case of symbolic regression (and many of its generalisations) the application of Equation (2) to all programs  $p$  in the search space  $\Omega$  produces a system of equations which correspond to a set of *geometric constraints* on the target behaviour  $t$ . However, it is clear that the locus of all  $t$  that satisfy Equation (2) for all  $p \in \Omega$  can be defined even in the most general program induction conditions, including cases where programs with side effects are explored, although in such cases a different geometric interpretation may be required.<sup>4</sup> To see this, we rewrite Equation (3) as

$$\mathbf{f} = [h(\mathbf{1} \cdot g(\mathbf{p}_1, \mathbf{T})), \dots, h(\mathbf{1} \cdot g(\mathbf{p}_r, \mathbf{T}))] \quad (6)$$

where  $\mathbf{1}$  is a vector whose components are all 1,  $\cdot$  represents scalar product,  $\mathbf{p}_i = (p_i(x_1) \dots p_i(x_n))$ ,  $\mathbf{T}$  is the ordered set  $\{t(x_1), \dots, t(x_n)\}$  and we extend the function  $g$  to act on its two arguments component wise, i.e.,  $g(\mathbf{p}_i, \mathbf{T}) = (g(p_i(x_1), t(x_1)) \dots g(p_i(x_n), t(x_n)))$ . So, every assignment of fitness to the programs in the search space produces a set of constraints on the ordered set  $\mathbf{T}$  (irrespective of what exactly its components represent). In particular, if  $f_1, \dots, f_r$  are the components of the vector  $\mathbf{f}$ , then  $\mathbf{T}$  must satisfy the system of equations

$$\begin{cases} h^{-1}(f_1) = \mathbf{1} \cdot g(\mathbf{p}_1, \mathbf{T}), \\ \vdots \\ h^{-1}(f_r) = \mathbf{1} \cdot g(\mathbf{p}_r, \mathbf{T}). \end{cases} \quad (7)$$

This has a geometric interpretation, albeit not in terms of distances. Because  $\mathbf{1} \cdot g(\mathbf{p}_i, \mathbf{T})$  represents the length of the

<sup>4</sup>We should note that a geometric interpretation for program/function induction fitness functions makes it particularly easy to interpret NFL and free-lunch results. So, although such an interpretation is not necessary, in the rest of the paper we will often restrict our attention to symbolic regression with fitness functions that are distance measures, which leads to the simplest geometric interpretation for our results.

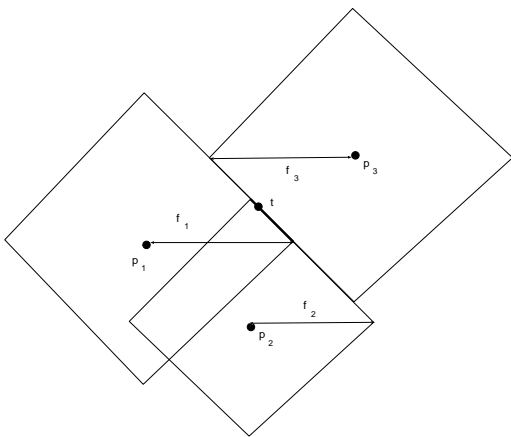


Figure 5: If  $g$  measures the absolute difference between two numbers, then a valid fitness function requires diamonds centred on each program behaviour and with edge-length given by  $\sqrt{2}$  times the corresponding fitness to intersect in one segment. Any target vector  $\mathbf{t}$  on that segment satisfies the constraints.

projection of the vector  $g(\mathbf{p}_i, \mathbf{T})$  along the vector  $\mathbf{1}$ , Equation (7) effectively defines  $n$  parallel planes and requires  $\mathbf{T}$  to be such that each  $g(\mathbf{p}_i, \mathbf{T})$  belongs to one such plane. The situation is illustrated in Figure 6.

We should note at this point that whenever the size,  $n$ , of the set of fitness cases is finite, and the representation for each target behaviour  $t(x_i)$  is finite (as is the case for anything that can be represented in a digital computer), then the space of possible fitness functions of the form in Equation (2) or Equation (6) is finite even if the number of programs in the search space,  $r$ , is infinite. This is because if at most  $k$  bits are necessary to represent each  $t(x_i)$ , then  $n \times k$  bits are sufficient to represent the set  $\mathbf{T}$ . So, there can be at most  $2^{n \times k}$  different fitness functions for program induction. In the particular case of symbolic regression fitness functions, since the target behaviours  $t(x_i)$  are simple real values, then  $k$  is typically either 32 or 64 depending on the chosen representation for floating point numbers. Thus, although the space of program induction fitness functions may be vast, that space is finite whenever the number of test examples is finite. So, Streeter’s result [7] about the applicability or otherwise of NFL to infinite sets of fitness functions (see Section 1) cannot really apply to the space of programs.

### 3. NO FREE LUNCH FOR PROGRAM AND FUNCTION INDUCTION

As we have seen symbolic regression fitness functions take the vector form in Equation (5). That is, the elements of a vector  $\mathbf{f}$  representing a symbolic regression problem are not independent degrees of freedom: they are the result of a distance measurement. An important question is whether or not imposing this constraint on the class of fitness functions has implications for the applicability of the no-free lunch theorem. As we will see below, the answer to this question is: “yes, there are important consequences”. In particular, we will show that under mild conditions the constraint

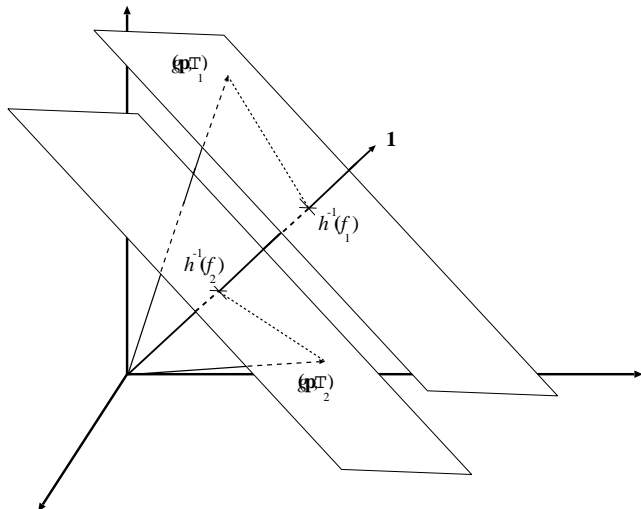


Figure 6: Geometric interpretation of program induction fitness functions which are functions of the sum of errors or fitness contributions over a set of fitness cases as in Equation 2.

implies that the set of GP fitness functions is not closed under permutation, and, therefore, NFL does not apply. So, the search for superior search algorithms is meaningful when dealing with program search spaces, for at least some performance measures. To start with, however, we will state under what conditions NFL applies to a set of symbolic regression problems.

As we indicated above, [6] showed that two arbitrary algorithms have identical performance (irrespective of the chosen performance measure) over a set of functions only if that set of functions is closed under permutation. That is, using the notation introduced in the previous section, in order for NFL to apply to a set of functions  $\mathcal{F}$ , for every  $\mathbf{f} \in \mathcal{F}$  there must be an  $\tilde{\mathbf{f}} \in \mathcal{F}$  such that  $\tilde{f}_i = f_{\sigma(i)}$  where  $\sigma$  is a permutation list (i.e., a permutation of the vector  $(1, \dots, r)$ ).<sup>5</sup> The following theorem connects NFL’s permutations with the true degrees of freedom of symbolic regression fitness functions.

**THEOREM 1.** *Let  $\mathcal{F} = \{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m\}$  be a set of fitness functions of the form in Equation (5) and let  $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_m\}$  be the set of target vectors associated to the functions in  $\mathcal{F}$ , with  $\mathbf{t}_i$  being the vector generating  $\mathbf{f}_i$  for all  $i$ . The set  $\mathcal{F}$  is closed under permutation (and NFL applies to it) if and only if for all target vectors  $\mathbf{t} \in \mathcal{T}$  and for all permutations  $\sigma$  of  $(1, 2, \dots, r)$  there exists a target vector  $\tilde{\mathbf{t}} \in \mathcal{T}$  such that*

$$d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) = d(\mathbf{p}_j, \tilde{\mathbf{t}}) \quad (8)$$

for all  $j = 1, 2, \dots, r$ .

**PROOF.** We proceed by reductio ab absurdum.

<sup>5</sup>Since we consider the set of test case inputs  $x_i$  fixed, here we treat a function  $f$  and its vectorial representation  $\mathbf{f}$  as equivalent. So, we will sometime say that  $\mathbf{f}$  is a function. Also, we will often think of a set of functions  $\mathcal{F}$  as a set of vectors and we will write things such as  $\mathbf{f} \in \mathcal{F}$ , instead of the more appropriate  $f \in \mathcal{F}$ .

**IF:** Let us assume  $\mathcal{F}$  is closed under permutation, but  $\exists \mathbf{t} \in \mathcal{T}$  and  $\exists \sigma$  such that  $\forall \tilde{\mathbf{t}} \in \mathcal{T}, \exists j_{\tilde{\mathbf{t}}} \in \{1, \dots, r\}$  for which  $d(\mathbf{p}_{\sigma(j_{\tilde{\mathbf{t}})}}, \mathbf{t}) \neq d(\mathbf{p}_{j_{\tilde{\mathbf{t}}}}, \tilde{\mathbf{t}})$  (note the dependency of  $j$  on  $\tilde{\mathbf{t}}$ ). Let  $\mathbf{f}$  be the fitness function associated to  $\mathbf{t}$ . If we permute  $\mathbf{f}$  with  $\sigma$  we obtain a fitness function  $\tilde{\mathbf{f}}$ . There are two possibilities: either  $\tilde{\mathbf{f}}$  is a symbolic regression fitness function and so a target vector generating it exists, or it isn't.

If  $\tilde{\mathbf{f}}$  is not a symbolic regression fitness function, then surely it cannot be a member of  $\mathcal{F}$ , but then this implies that it is possible to permute an element of the set and obtain an element outside it. So,  $\mathcal{F}$  is not closed under permutation, which is a contradiction.

If, instead,  $\tilde{\mathbf{f}}$  is a symbolic regression fitness function, then let  $\tilde{\mathbf{t}}$  be a target vector associated to it. So,  $\mathbf{f}_j = d(\mathbf{p}_j, \tilde{\mathbf{t}})$ . Because  $f_{\sigma(j)} = \tilde{f}_j$  and  $f_{\sigma(j)} = d(\mathbf{p}_{\sigma(j)}, \mathbf{t})$ , then  $d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) = d(\mathbf{p}_j, \tilde{\mathbf{t}})$  for all  $j$ . However, earlier we assumed that for our particular choice of  $\mathbf{t}$  and  $\sigma$  for every element  $\tilde{\mathbf{t}} \in \mathcal{T}$  there is always a  $j$  for which  $d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) \neq d(\mathbf{p}_j, \tilde{\mathbf{t}})$ . Since  $\tilde{\mathbf{t}}$  does not, it must be the case that  $\tilde{\mathbf{t}} \notin \mathcal{T}$ . As a consequence  $\tilde{\mathbf{f}} \notin \mathcal{F}$ . So,  $\mathcal{F}$  is not closed under permutation, which is a contradiction.

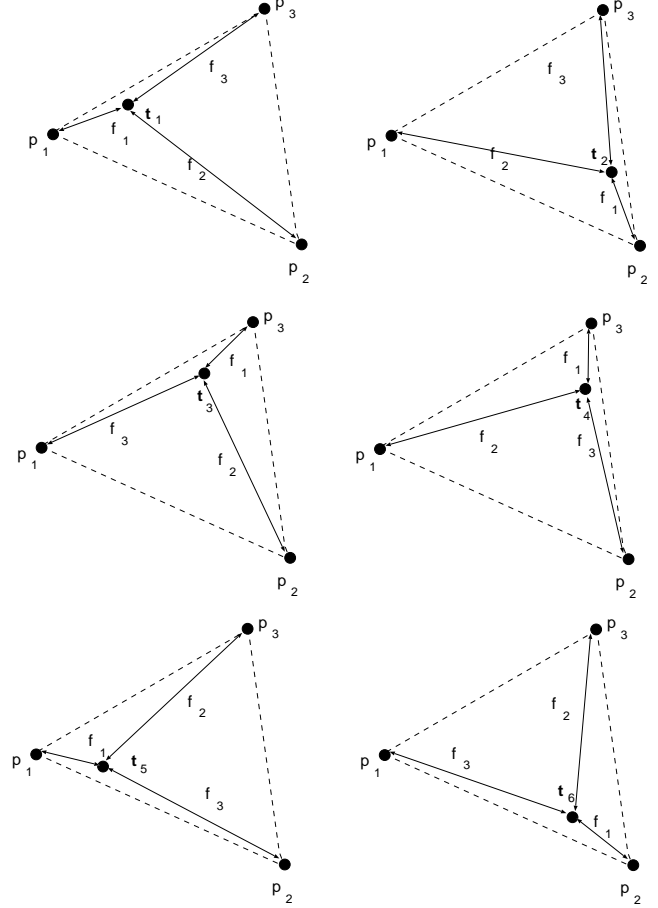
**ONLY IF:** Let us assume that  $\forall \mathbf{t} \in \mathcal{T}$  and  $\forall \sigma, \exists \tilde{\mathbf{t}} \in \mathcal{T}$  such that  $d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) = d(\mathbf{p}_j, \tilde{\mathbf{t}})$  for all  $j$ , but that  $\exists \mathbf{f} \in \mathcal{F}$  and  $\exists \sigma$  such that  $\tilde{f}_j = f_{\sigma(j)}$  but  $\tilde{\mathbf{f}} \notin \mathcal{F}$ . Let  $\mathbf{t}$  be the target vector associated to this particular  $\mathbf{f}$ .

Because of our assumption, even when we consider the permutation  $\sigma$  that generates  $\tilde{\mathbf{f}}$  from  $\mathbf{f}$ , there must be some  $\tilde{\mathbf{t}} \in \mathcal{T}$  for which  $d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) = d(\mathbf{p}_j, \tilde{\mathbf{t}})$  for all  $j$ . Since  $\tilde{\mathbf{t}} \in \mathcal{T}$  it must have a function,  $\hat{\mathbf{f}}$ , associated to it and  $\hat{\mathbf{f}} \in \mathcal{F}$ . Furthermore, if we interpret the relationship  $d(\mathbf{p}_{\sigma(j)}, \mathbf{t}) = d(\mathbf{p}_j, \tilde{\mathbf{t}})$  in terms of fitnesses, it is clear that  $\hat{f}_j = f_{\sigma(j)}$  for all  $j$ . As a consequence,  $\hat{\mathbf{f}} \equiv \tilde{\mathbf{f}}$  and so  $\tilde{\mathbf{f}} \in \mathcal{F}$ , which is a contradiction.  $\square$

From a geometrical point of view, the target vector  $\mathbf{t} \in \mathcal{T}$  associated to a function  $\mathbf{f} \in \mathcal{F}$  must be at the intersection of the spheres centred on programs  $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r$  and having radii  $f_1, f_2, \dots, f_r$ , respectively (see discussion at the end of Section 2). Permuting the elements of  $\mathbf{f}$  via a permutation  $\sigma$  to obtain a new fitness function corresponds to shuffling (according to  $\sigma$ ) the radii of the spheres centred on  $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r$ . Note these centres remain fixed since they represent the behaviour of the programs in the search space. After the shuffling some spheres may have had their radius decreased, some increased, and some unchanged. If any radius has changed then  $\mathbf{t}$  can no longer be the intersection of the spheres. However, we must be able to find a new intersection  $\tilde{\mathbf{t}} \in \mathcal{T}$  or else the new fitness function we have generated is not a symbolic regression fitness function and therefore cannot be a member of  $\mathcal{F}$ , which would imply that the set is not closed under permutation.

One might wonder, at this point, how difficult it would be to find a set of fitness functions that satisfy Theorem 1. As shown in Figure 7, by making use of symmetries, it is easy to create an artificial program space and an associated set of symbolic regression problems which are closed under permutation. So, situations where NFL applies to symbolic regression are possible.

Equation (8) is a mathematical statement of the geometric requirements for  $\tilde{\mathbf{t}}$  to exist. The left-hand side represents the fitness that was originally associated to program  $j$  and that, because of the permutation/shuffling, must now be assigned to program  $\sigma(j)$ . The right-hand side represents the



**Figure 7:** Three programs  $p_1, p_2, p_3$  whose behaviours  $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$  over a test set of  $n = 2$  fitness cases are the corners of an equilateral triangle in  $\mathbb{R}^2$ . A fitness function  $\mathbf{f}$  induced by a target vector  $\mathbf{t}_1$  assigns fitnesses  $f_1, f_2$  and  $f_3$  to such programs (top left). Then, because of geometric symmetries, any permutation of such fitness assignments to the three programs is also induced by a target vector (see  $\mathbf{t}_2, \dots, \mathbf{t}_6$  in top right and remaining panels).

constraint that  $\tilde{\mathbf{t}}$  must be on the surface of the new sphere. Because we require this condition to be verified for every  $j$ , we have a system of equations where  $\tilde{\mathbf{t}}$  is the unknown.

As we have seen in Section 2, a similar system of equations, namely Equation (7), is obtained if we consider fitness functions of the more general form in Equation (2). In the proof of Theorem 1 the fact that the function  $d$  is a distance is never used. Nothing would prevent the function  $d$  from having a form such as  $h(\mathbf{1} \cdot g(\mathbf{p}_i, \mathbf{T}))$ . It is then clear that we can generalise Theorem 1 to fitness functions of the form in Equations (2) and (6) obtaining

**THEOREM 2.** Let  $\mathcal{F} = \{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m\}$  be a set of fitness functions of the form in Equation (6) and let  $\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_m\}$  be the set of target behaviours associated to the functions in  $\mathcal{F}$ , with  $\mathbf{T}_i$  being the behaviour generating  $\mathbf{f}_i$  for all  $i$ . The set  $\mathcal{F}$  is closed under permutation (and NFL applies to it) if and only if for all target behaviours  $\mathbf{T} \in \mathcal{T}$  and for all permutations  $\sigma$  of  $(1, 2, \dots, r)$  there exists

a target behaviour  $\tilde{\mathbf{T}} \in \mathcal{T}$  such that

$$\mathbf{1} \cdot g(\mathbf{p}_{\sigma(j)}, \mathbf{T}) = \mathbf{1} \cdot g(\mathbf{p}_j, \tilde{\mathbf{T}}) \quad (9)$$

for all  $j = 1, 2, \dots, r$ .

PROOF. The proof is identical to the proof of Theorem 1 with the replacements  $f_i \leftrightarrow h^{-1}(f_i)$  and  $d(a, b) \leftrightarrow \mathbf{1} \cdot g(a, b)$ .  $\square$

A geometric illustration of this theorem is provided in Figure 8.

We should note that Theorems 1 and 2 focus on the set of program behaviours. So, they tell us something about the nature of the phenotype-to-fitness function. They really state under which conditions there can be an NFL for a searcher exploring program behaviours with no resampling. If a search algorithm instead explores the space of syntactic structures representing programs (e.g., sequences of machine code instructions or syntax trees), in the presence of symmetries (such as those highlighted by Woodward and Neil) then the searcher will produce resampling of program behaviours even if it never resampled the same syntactic structure. So, in the presence of a set of behaviours for which NFL holds, this would unavoidably give the ‘‘syntactic’’ searcher lower average performance than an algorithm which never resampled behaviours.

Related to this, we should note that in constructing the example in Figure 7, we assumed program behaviours can be chosen arbitrarily. In reality program behaviours are the expression (e.g., execution) of a corresponding syntactic representation (which we would call a genotype in GP). So, one should really verify if NFL sets such as the ones represented in Figure 7 would be compatible with the ‘‘syntactic’’/‘‘genotypic’’ program space. Such sets do exist. For example, if  $\Omega$  includes the three programs  $p_1 = 0$ ,  $p_2 = (x + 1)/2$  and  $p_3 = ((1 - \sqrt{3}) * x + (1 + \sqrt{3}))/4$  and the fitness cases are  $x = -1$  and  $x = +1$ , then we are in a situation similar to the one depicted in Figure 7. However, it is apparent how artificial this situation is. So, in general, finding a set of actual programs (as opposed to behaviours) and fitness cases for which NFL holds for symbolic regression, and, more generally, program/function induction, may be much harder than the figure may suggest.

## 4. FREE LUNCHES FOR SYMBOLIC REGRESSION AND PROGRAM INDUCTION

Theorems 1 and 2 impose requirements which involve all possible permutations of fitness functions. Naturally, even if we focused on one specific permutation of  $\mathbf{f}$ , instead of all permutations, we still have the question of whether or not a vector  $\tilde{\mathbf{t}}$  satisfying the system of equations (8) or a set  $\tilde{\mathbf{T}}$  satisfying equations (9) can exist in general.

In the case of  $\tilde{\mathbf{t}}$  the problem is principally that we have  $r$  constraint equations ( $j \in \{1, \dots, r\}$ ), but only  $n$  variables ( $\tilde{t}_1, \dots, \tilde{t}_n$ ). Therefore, in general the problem of finding a vector  $\tilde{\mathbf{t}}$  satisfying Equations (8) should be expected to be over-constrained, and we should not be able to build a set of symbolic-regression-type of problems which is closed under permutation.

The situation is similar for  $\tilde{\mathbf{T}}$ . The target behaviour  $\tilde{\mathbf{T}}$  must be expressed through a set of degrees of freedom. What exactly these are depends on the application, but there must

be such a set. If  $n$  is the number of degrees of freedom characterising  $\tilde{\mathbf{T}}$ , since we have  $r$  constraint equations (Equations (9)), finding a  $\tilde{\mathbf{T}}$  which satisfies them all may be an over-constrained problem if  $r > n$ . Given the immense sizes of typical program search spaces this condition is easily met.

Informally speaking, this implies that, in general, *there is a free lunch for search in program spaces*. We will devote the rest of this section to illustrating this for a variety of practical situations.

Schumacher *et al.*’s result [6] — that two arbitrary algorithms have identical performance over a set of functions if and only if that set of functions is closed under permutation — was proven by designing a performance measure such that if NFL held over a set of functions  $\mathcal{F}$  which is not closed under permutation, it would then be possible to identify an algorithm whose average performance over  $\mathcal{F}$  is inferior to that of some other algorithm, leading to a contradiction. So, another way to look at this result is the following:

**COROLLARY 3.** *If the set of functions  $\mathcal{F}$  is not closed under permutation, then there exists at least one performance measure  $M$  under which at least one algorithm has better (worse) than average performance at optimising functions from  $\mathcal{F}$ .*

The result is particularly important for what we want to investigate in this paper.

While an analytic approach to finding conditions where NFL does not hold for program induction is possible, it is particularly easy to see what type of situations might lead to sets of symbolic regression fitness functions that are not closed under permutation by considering the geometric interpretation of such fitness functions. This is, of course, best illustrated when target vectors and program behaviours can be represented as 2-D points. As shown in Figure 9, while one can assign any fitness value to program behaviours in the search space, most assignments cannot be induced by a target vector simply because the spheres centred on each program and with radius equal to the fitness of the program may not intersect in one point. So, even if one started the construction of a set of functions from a function which is induced by a target vector, permuting such a function may easily produce something which violates some of the geometric constraints that symbolic regression fitness functions must satisfy.

Let us look at a simple GP example. Let us consider the function set  $\{+, -\}$  and the terminal set  $\{x\}$  and let us restrict the search space  $\Omega$  to programs of up to depth 1. That is  $\Omega = \{x, (+ x x), (- x x)\}$  (where we expressed programs in standard Lisp prefix notation). Let the size of the training set be  $n = 2$ , with  $x_1 = -1$  and  $x_2 = +1$ , and let us assume that the corresponding target values are  $t_1 = 0$  and  $t_2 = 0$  (e.g., the target function might be  $t(x) = 0$ ).

Following standard practice in GP let us assume that  $g(a, b) = |a - b|$  and let us evaluate the fitness of program  $p_1 = x$ . Clearly  $f_1 = 2$  since the function  $p_1$  evaluates to -1 in -1 and to 1 in 1 (so  $\mathbf{p}_1 = (-1, 1)$ ) thereby producing two fitness contributions (errors) of 1 each. Let us now consider  $p_2$ . Because this is equivalent to the function  $2x$ ,  $\mathbf{p}_2 = (-2, 2)$  and we get double the errors than in the previous case, resulting in  $f_2 = 4$ . Finally, the fitness of  $p_3$  is  $f_3 = 0$  since  $p_3 \equiv 0$  everywhere and so  $\mathbf{p}_3 = (0, 0)$ . Thus, we get  $\mathbf{f} = (2, 4, 0)$ . Geometrically, the situation is as depicted in Figure 10(a).

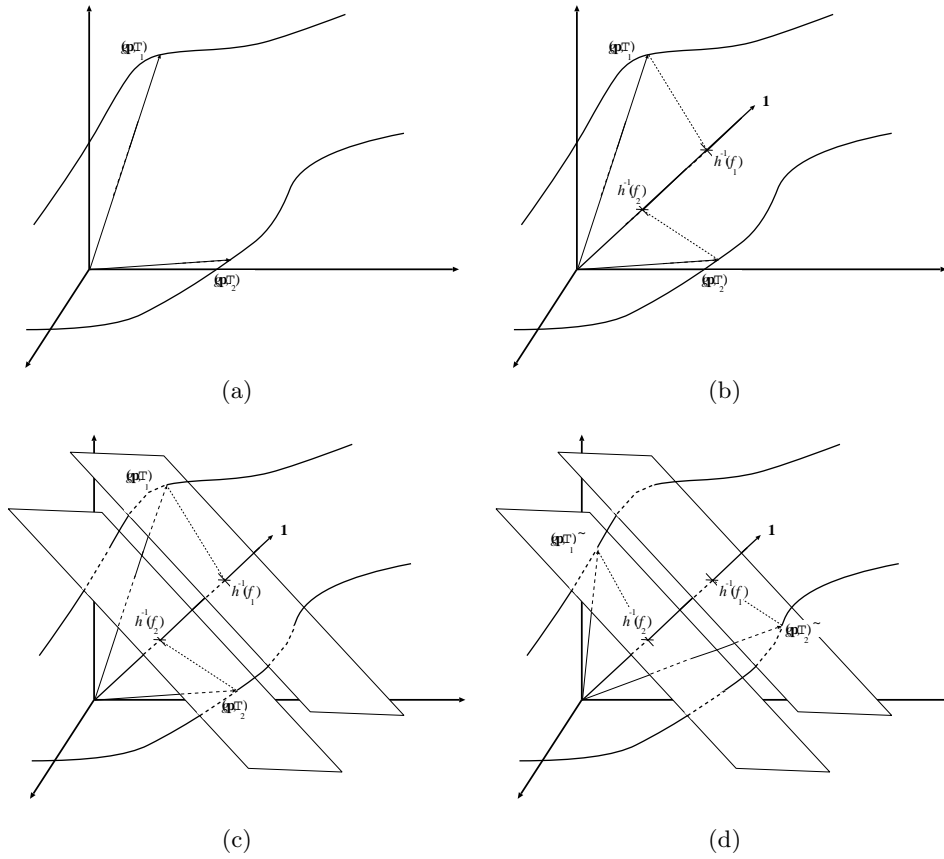


Figure 8: Illustration of Theorem 2. Let  $p_1$  and  $p_2$  be two program behaviours: (a) as the target behaviour  $T$  is varied, the vectors  $g(p_1, T)$  and  $g(p_2, T)$  produce two curves in  $\mathbb{R}^n$ ; (b) for a given  $T$  the fitness of the programs  $p_1$  and  $p_2$  is determined by the length of the projection of the vectors  $g(p_1, T)$  and  $g(p_2, T)$  onto the vector  $\mathbf{1}$  via the function  $h$ ; (c) to each fitness value corresponds a hyperplane; (d) Theorem 2 requires that for each set of hyperplanes it is possible to find a target behaviour  $\tilde{T}$  such that  $g(p_1, \tilde{T})$  belongs to the hyperplane originally associated with  $g(p_2, T)$  while  $g(p_2, \tilde{T})$  belongs to the hyperplane originally associated with  $g(p_1, T)$ .

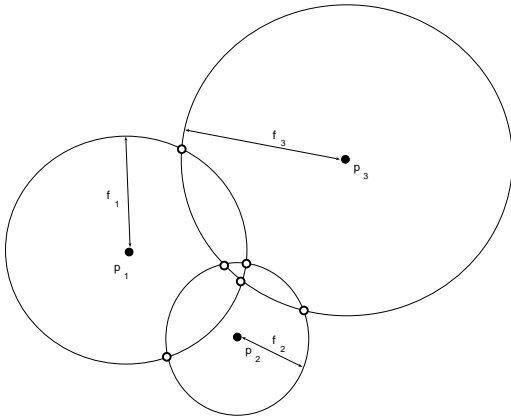


Figure 9: Certain assignments of fitnesses to program behaviours are incompatible with symbolic regression, in the sense that their fitness cannot represent the error between the functionality of the programs and a target functionality. So, there cannot be a point where all spheres meet. In these cases one cannot find a target vector  $\mathbf{t}$  that could generate such a fitness function.

Let us now consider the following permutation of this function:  $\tilde{\mathbf{f}} = (0, 2, 4)$ . Is there values of  $\tilde{t}_1$  and  $\tilde{t}_2$  that can induce this new function? Because this function requires  $f_1 = 0$ , it must be the case that  $\tilde{t}_1 = -1$  and  $\tilde{t}_2 = +1$ , or else the error for  $p_1$  would be bigger than 0. But do these values correctly induce the remaining entries in  $\tilde{\mathbf{f}}$ ? With  $p_2$ , i.e.,  $2x$ , we see that these target values produce the correct result,  $\tilde{f}_2 = 2$ . However, when we test  $p_3$  which is 0 everywhere, we find that  $\tilde{\mathbf{t}}$  induces a value of fitness  $\tilde{f}_3 = 2$  and not the required 4. The situation is depicted in Figure 10(b). Therefore, it is impossible to build a set of symbolic regression fitness functions for  $\Omega$  what is closed under permutation and includes the function  $\mathbf{f} = (2, 4, 0)$  induced by  $\mathbf{t} = (0, 0)$ .

As illustrated in Figure 11 the situation does not change if we consider a fitness function based on adding up squared errors instead of absolute errors.

A geometric arrangement similar to the one depicted in Figure 11 also represents the even simpler situation where the three programs  $p_1$ ,  $p_2$  and  $p_3$  are constant functions, e.g.,  $p_k = k$ . If the target function  $t$  is also a constant function, the three programs will have fitnesses which cannot be permuted in all possible ways and still produce symbolic regression fitness functions.

These examples are useful since they show that a free

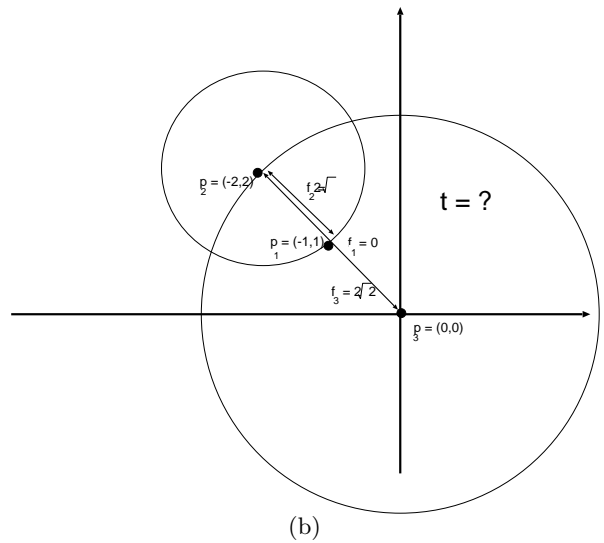
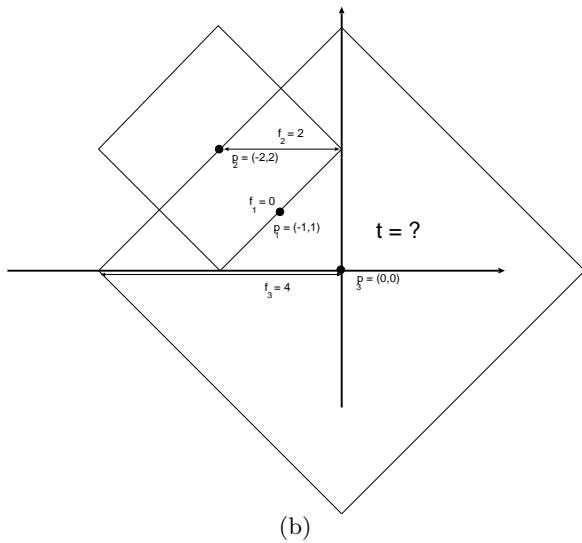
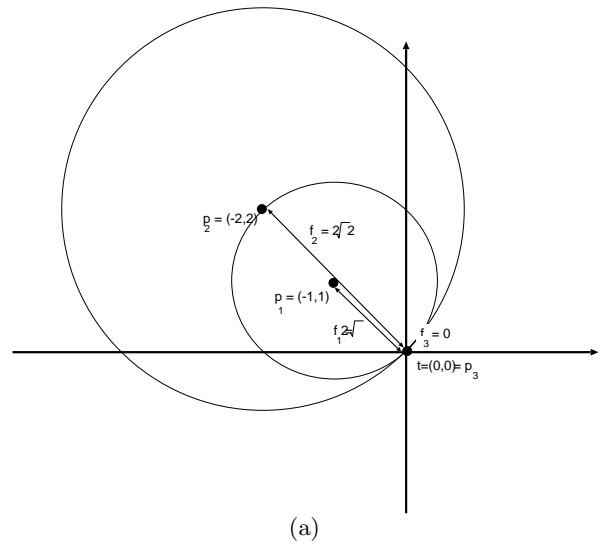
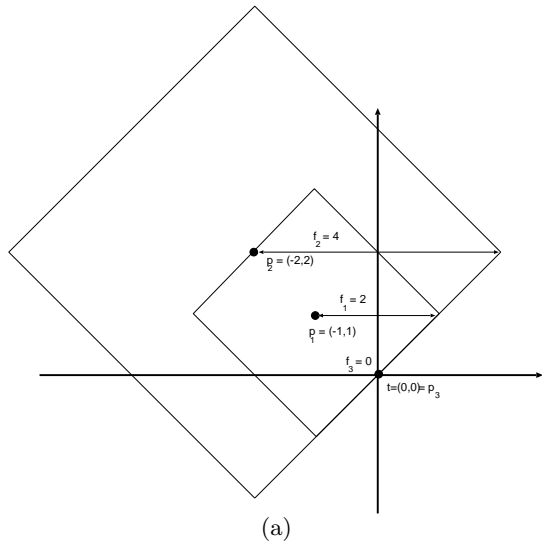


Figure 10: A search space and a city-block-type symbolic regression fitness function (a), which, however, cannot be permuted and still remain a valid symbolic regression fitness function (b).

Figure 11: A search space and a valid Euclidean-type symbolic regression fitness function (a), which, however, cannot be permuted and still remain a valid symbolic regression fitness function (b).

lunch is possible for search in program spaces even if there isn't a many-to-one genotype-phenotype mapping. However, a geometric interpretation of the problem can lead to much more general results. For example, it is easy to see that if two programs have identical behaviour, they must have identical fitness or there cannot be any intersection between the spheres centred on them (see Figure 12). With this in mind, it is then trivial to understand the following:

**THEOREM 4.** *Consider a search space which includes at least two programs  $p_1$  and  $p_2$  such that  $\mathbf{p}_1 = \mathbf{p}_2$  (i.e., the two programs give identical outputs for all  $x$  in the training set). Let a set of symbolic regression problems  $\mathcal{F}$  contain a fitness function  $f$  induced by a target vector  $\mathbf{t}$  such that there exist a third program  $p_3$  in the search space with fitness  $f(p_3) \neq f(p_1) = f(p_2)$ . Then  $\mathcal{F}$  cannot be closed under permutation and NFL does not hold.*

**PROOF.** Let  $\alpha = f(p_1) = f(p_2)$  and  $\beta = f(p_3)$  with  $\alpha \neq \beta$ . If  $\mathcal{F}$  was closed under permutation, there would have to be a permutation of  $f, \tilde{f}$ , such that  $\tilde{f}(p_1) = \beta$  and  $\tilde{f}(p_2) = \alpha$ . Let us assume that a vector  $\tilde{\mathbf{t}}$  which induces the function  $\tilde{f}$  exists. From Equation (4) we have

$$\beta = d(\mathbf{p}_1, \tilde{\mathbf{t}}) \quad \text{and} \quad \alpha = d(\mathbf{p}_2, \tilde{\mathbf{t}})$$

However, because we know that  $\mathbf{p}_1 = \mathbf{p}_2$ , we have that  $d(\mathbf{p}_1, \tilde{\mathbf{t}}) = d(\mathbf{p}_2, \tilde{\mathbf{t}})$  and, so,  $\beta = \alpha$ , a contradiction. Hence, a target vector  $\tilde{\mathbf{t}}$  with the required properties does not exist, hence  $\tilde{f}$  is not a symbolic regression problem, and, so,  $\tilde{f} \notin \mathcal{F}$ . This then leads to conclude that  $\mathcal{F}$  cannot be closed under permutation.  $\square$

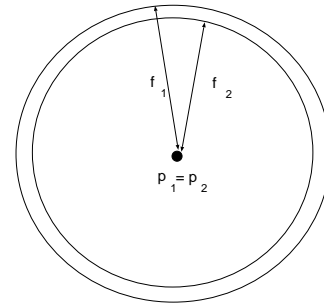
This result investigates the case where the genotype-phenotype map is many-to-one, in the sense that while programs  $p_1$  and  $p_2$  are distinct, their behaviours  $\mathbf{p}_1$  and  $\mathbf{p}_2$  are identical. So, we are considering the same situation as Woodward and Neil [12] (see Section 1). However, we can now see that in the case of symbolic regression fitness functions non-uniformity in the mapping is not required for NFL to break down. NFL breaks down because it is impossible to create a set of problems that is closed under permutation. Note also that our result does not require that the two programs  $p_1$  and  $p_2$  be equivalent, as would be the case, for example, for the programs  $a + b$  and  $b + a$  which were used by Woodward [11] to create a counter-example to NFL in the case of program induction (see Section 1). It simply requires that two programs produce the same outputs to all fitness cases in the particular training set one has chosen.

Continuing with our geometric investigation of NFL, looking at Figure 4 it is clear that a necessary condition for the existence of a target vector  $\mathbf{t}$  which induces a particular fitness function is that the triangular inequality be verified. More precisely:

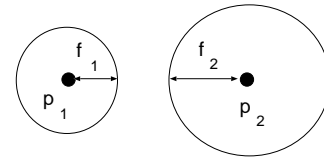
**LEMMA 5.** *If a target vector  $\mathbf{t}$  induces a symbolic regression fitness function  $\mathbf{f}$ , then for every pair of program behaviours  $\mathbf{p}_1$  and  $\mathbf{p}_2$  the distance  $d(\mathbf{p}_1, \mathbf{p}_2)$  between  $\mathbf{p}_1$  and  $\mathbf{p}_2$  (based on the same metric used to measure fitnesses) must not be greater than  $f_1 + f_2$ .*

**PROOF.** Because of the triangular inequality  $d(\mathbf{p}_1, \mathbf{p}_2) \leq d(\mathbf{p}_1, \mathbf{t}) + d(\mathbf{p}_2, \mathbf{t}) = f_1 + f_2$  for any valid symbolic regression fitness function.  $\square$

From this result we can see that another common situation where there is incompatibility between an assignment of fitness to programs and the fitness representing a symbolic



**Figure 12:** If two programs have identical behaviour, they must have identical fitness or there cannot be any intersection between the spheres centred on them.



**Figure 13:** The programs  $p_1$  and  $p_2$  have fitness such that  $f_1 + f_2$  is smaller than their distance. So, by the triangular inequality there cannot be any intersection between the spheres centred on them.

regression problem is the case in which two programs have different behaviours (and so are represented by two distinct points in the  $\mathbb{R}^n$ ), but the sum of their fitnesses is smaller than their distance. The situation is illustrated in Figure 13. This leads to the following result:

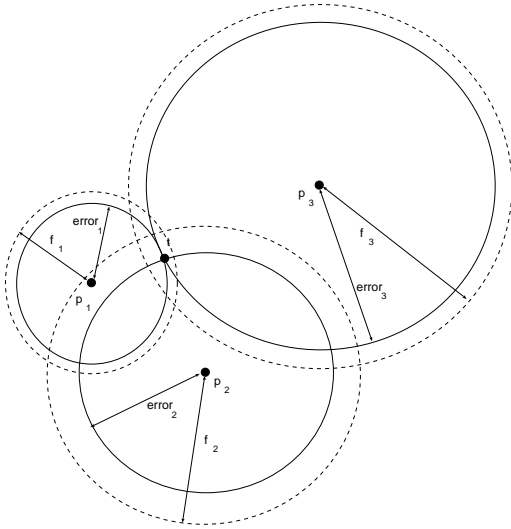
**THEOREM 6.** *Given a set of symbolic regression functions  $\mathcal{F}$ , if there exists any function  $\mathbf{f} \in \mathcal{F}$  and any four program behaviours  $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$  in a search space such that  $d(\mathbf{p}_1, \mathbf{p}_2) > f_3 + f_4$  then the set is not closed under permutation and NFL does not apply.*

**PROOF.** If  $\mathcal{F}$  is closed under permutation, then there must be a permutation of  $\mathbf{f}, \tilde{\mathbf{f}}$ , which assigns the values of fitness originally associated to  $p_3$  and  $p_4$  to  $p_1$  and  $p_2$ , respectively, instead. That is  $\tilde{f}_1 = f_3$  and  $\tilde{f}_2 = f_4$ . As a result  $d(\mathbf{p}_1, \mathbf{p}_2) > f_3 + f_4$  implies  $d(\mathbf{p}_1, \mathbf{p}_2) > \tilde{f}_1 + \tilde{f}_2$ . Thus, by Lemma 5, there cannot be a target vector  $\tilde{\mathbf{t}}$  which induces  $\tilde{\mathbf{f}}$ . So,  $\tilde{\mathbf{f}}$  is not a symbolic regression fitness function and cannot therefore be a member of  $\mathcal{F}$ . So, not all permutations of  $\mathbf{f}$  are in  $\mathcal{F}$ , and the set is not closed under permutation.  $\square$

## 5. EXTENSIONS

In this paper we have mainly concentrated on a set of fitness functions which represent the cumulative error a program makes when evaluating a set of training cases. We have studied in particular detail the symbolic regression case where the fitness satisfies the axioms of distances. Many if not most GP systems use exactly these types of fitness functions. However, the ideas in the paper can easily be extended to other and more general situations.

One important case is the case where the classical symbolic regression fitness measure is combined with some other measure of performance, such as a parsimony pressure term



**Figure 14: Graphical interpretation of fitnesses and distances for symbolic regression fitness functions with constant terms.**

which is added to the fitness of programs and is modulated by the size of the programs. If this is the case the fitness of program  $p$  is obtained as

$$f(p) = h \left( \sum_{i=1}^n g(p(x_i), t(x_i)), c(p) \right) \quad (10)$$

where:  $c(p)$  is a function, which we will call a parsimony term, whose output depends solely on the structure or behaviour of program  $p$ ; the function  $h$  combines the error term with the parsimony term  $c(p)$ ; the other symbols have the same meaning as in Equation (2). Frequently this equation can be expressed using distances as follows

$$f(\mathbf{p}) = d(\mathbf{p}, \mathbf{t}) + c(p). \quad (11)$$

The addition of the offsets  $c(p)$  complicates a geometrical interpretation of the requirements for a fitness function to be a symbolic regression fitness function. Essentially, we have two circles around each  $\mathbf{p}$ : an inner circle of radius  $d(\mathbf{p}, \mathbf{t})$  which contains  $\mathbf{t}$  and represents the true error program  $p$  makes on the training data, and an outer circle of radius  $d(\mathbf{p}, \mathbf{t}) + c(p)$  which represents the fitness assigned to it. A fitness function can then be a symbolic regression fitness function only if all of the inner circles associated to different programs in the search space meet at point  $\mathbf{t}$ . The situation is exemplified in Figure 14.

From an analytic point of view, however, things remain rather simple. All we need for a fitness function to be a valid symbolic regression fitness function induced by a target vector  $\mathbf{t}$  and a penalty function  $c(p)$  is that

$$f_j = d(\mathbf{p}_j, \mathbf{t}) + c_j \quad (12)$$

for  $j = 1, \dots, r$ , where  $\mathbf{c}$  is a vector representing the parsimony coefficients associated to each program  $p_j$ , i.e.,  $c_j = c(p_j)$ . It then seems reasonable to expect that a form of generalisation of Theorem 1 for this class of problems would be straightforward. We can also expect that many of the free-lunch results presented in the previous section could be extended to this case and that, in fact, NFL would be

even less applicable due to the asymmetries introduced by the constant terms.

As a concrete example of one such extension, we provide an extension of Theorem 4 which applies to the case where the fitness is composed by the distance between the behaviour of  $p$  and the behaviour of  $t$  plus some constant that depends on the program  $p$ .

**THEOREM 7.** *Consider a search space which includes at least two programs,  $p_1$  and  $p_2$ , such that  $p_1(x) = p_2(x)$  for all  $x$  in the training set. Let a set of symbolic regression problems with parsimony terms,  $\mathcal{F}$ , contain a fitness function  $f$  induced by a target vector  $\mathbf{t}$  such that there exist a third program  $p_3$  in the search space with fitness  $f(p_3)$  such that  $f(p_3) \neq f(p_1)$  and  $f(p_3) \neq f(p_2)$ . Then  $\mathcal{F}$  cannot be closed under permutation and NFL does not hold.*

**PROOF.** The proof is divided into two cases:

**$\mathbf{c}(\mathbf{p}_1) = \mathbf{c}(\mathbf{p}_2)$ :** Given the equality of these parsimony coefficients and the fact that  $p_1(x) = p_2(x)$  for all  $x$  in the training set, then  $f(p_1) = f(p_2)$ . Let  $\alpha = f(p_1)$  and  $\beta = f(p_3)$  with  $\alpha \neq \beta$ . If  $\mathcal{F}$  was closed under permutation, there would have to be a permutation of  $f$ ,  $\tilde{f}$ , such that  $\tilde{f}(p_1) = \beta$  and  $\tilde{f}(p_2) = \alpha$ . Let us assume that a vector  $\tilde{\mathbf{t}}$  which induces the function  $\tilde{f}$  exists. From Equation (11) we have

$$\beta = d(\mathbf{p}_1, \tilde{\mathbf{t}}) + c(p_1) \quad \text{and} \quad \alpha = d(\mathbf{p}_2, \tilde{\mathbf{t}}) + c(p_2).$$

However, because we  $p_1(x) = p_2(x)$ , it must be the case that  $d(\mathbf{p}_1, \tilde{\mathbf{t}}) = d(\mathbf{p}_2, \tilde{\mathbf{t}})$ . We also know that  $c(p_1) = c(p_2)$  and so  $\alpha = \beta$  which is a contradiction. Hence, a target vector  $\tilde{\mathbf{t}}$  with the required properties does not exist.

**$\mathbf{c}(\mathbf{p}_1) \neq \mathbf{c}(\mathbf{p}_2)$ :** Given the inequality in the parsimony coefficient and the fact that  $p_1(x) = p_2(x)$  for all  $x$  in the training set, we know that  $f(p_1) \neq f(p_2)$ . Let  $\alpha = f(p_1)$  and  $\beta = f(p_2)$  with  $\alpha \neq \beta$ . There must be a permutation of  $f$  where  $\tilde{f}(p_1) = \beta$  and  $\tilde{f}(p_2) = \alpha$ . Let us assume that a vector  $\tilde{\mathbf{t}}$  exists. From Equation (11) we have

$$\beta = d(\mathbf{p}_1, \tilde{\mathbf{t}}) + c(p_1) \quad \text{and} \quad \alpha = d(\mathbf{p}_2, \tilde{\mathbf{t}}) + c(p_2).$$

However, because we know that  $p_1(x) = p_2(x)$  then  $d(\mathbf{p}_1, \tilde{\mathbf{t}}) = d(\mathbf{p}_2, \tilde{\mathbf{t}})$ . Using these equivalences in the previous equations we get  $\beta = \alpha - c(p_2) + c(p_1)$ . By substituting the values  $\alpha = f(p_1)$  and  $\beta = f(p_2)$  and expressing the result in terms of distances using Equation (11), one obtains  $d(\mathbf{p}_2, \tilde{\mathbf{t}}) + c(p_2) = d(\mathbf{p}_1, \tilde{\mathbf{t}}) + c(p_1) - c(p_2) + c(p_1)$ . This equation simplifies to  $2c(p_2) = 2c(p_1)$  which is a contradiction.  $\square$

Naturally the constant  $c(p)$  can represent the evaluation of any property of program  $p$  (not just size). We are also free to set  $c(p) \equiv 0$  for all  $p$ , which shows that Theorem 4 is effectively a corollary of Theorem 7.

We conclude this section with an extension which links the ability to solve problems with free lunches in the most general conditions, i.e., without requiring that fitness functions are of the form in Equation (2):

**THEOREM 8.** *Let  $\mathcal{F}$  be a set of program induction problems and let  $p_1$  be a program in the search space  $\Omega$  such there exists a fitness function  $f_1$  in  $\mathcal{F}$  for which  $f_1(p_1) = 0$  (i.e.,  $p_1$  is a 100% correct solution). If there exists a program  $p_2$  in  $\Omega$  such that there is no fitness function  $g$  in  $\mathcal{F}$  for which  $g(p_2) = 0$ , then  $\mathcal{F}$  cannot be closed under permutation and NFL does not hold.*

PROOF. If there is one program,  $p_1$ , that solves one problem,  $f_1$ , in the set of problems and one that solves none,  $p_2$ , then in at least one permutation of  $f_1$  we will need to assign a fitness of zero to program  $p_2$ , but then this means it solves at least one problem in  $\mathcal{F}$ . Contradiction.  $\square$

## 6. CONCLUSIONS

We have shown that under certain conditions on the behaviour of the programs in the search space, NFL can apply to the domains of program and function induction as well as to symbolic regression, an important class of problems in genetic programming and other induction techniques exploring program spaces. However, we have also found that it is extremely easy to find realistic situations in which a set of induction problems is provably not closed under permutation. This implies, that there is a free lunch for techniques that sample the space of computer programs, particular when the purpose is fitting data-sets.

Naturally, the fact that NFL results do not hold for these problem domains does not imply that the existing methods for searching program and function spaces, such as the many variants of GP and evolutionary neural networks, are average, better than average or worse than average. This may need to be proved by other means. The non-applicability of NFL, however, means that it is worthwhile to try to come up with new and more powerful algorithms for function and program induction and for symbolic regression.

## Acknowledgements

We would like to thank Yossi Borenstein and Chris Stephens for helpful comments. The reviewers are also warmly thanked for their insightful and constructive comments.

## 7. REFERENCES

- [1] C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321, 2003.
- [2] C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3:313–322, 2004.
- [3] W. Kirchner, M. Li, and P. Vitanyi. The miraculous universal distribution. *Mathematical Intelligencer*, 19:7–15, 1997.
- [4] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- [5] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008. (With contributions by J. R. Koza).
- [6] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 565–570. Morgan Kaufmann, 2001.
- [7] M. J. Streeter. Two broad classes of functions for which a no free lunch result does not hold. In *Proc. Genetic and Evolutionary Computation Conference GECCO-2003*, pages 1418–1430. Morgan Kaufmann, 2003.
- [8] D. Whitley and J. Rowe. Focused no free lunch theorems. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 811–818, New York, NY, USA, 2008. ACM.
- [9] D. Whitley and J. P. Watson. Complexity theory and the no free lunch theorem. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 11, pages 317–339. Springer US, 2005.
- [10] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.
- [11] J. Woodward. GA or GP? that is not the question. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1056–1063, Canberra, 8-12 Dec. 2003. IEEE Press.
- [12] J. R. Woodward and J. R. Neil. No free lunch, program induction and combinatorial problems. In C. Ryan, T. Soule, M. Keijzer, E. P. K. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 475–484, Essex, 14-16 Apr. 2003. Springer-Verlag.