International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol. 1, No. 2 (2005), pp. 183–197

The Building Block Basis for Genetic Programming and Variable-length Genetic Algorithms

Riccardo Poli[1] **and Christopher R. Stephens**[2]

[1]Department of Computer Science
University of Essex, UK
rpoli@essex.ac.uk

[2]Department of Computer Science
University of Essex, UK
csteph@essex.ac.uk

Instituto de Ciencias Nucleares
UNAM, Mexico
stephens@nucleares.unam.mx

Abstract: **The Building Block Basis (BBB) has recently been shown to be extremely useful in characterising the dynamics of genetic algorithms operating on fixed-length strings. In this paper we show that there is a natural generalisation of the BBB for variable-length strings and program trees.**

I. Introduction

An Evolutionary Algorithm (EA) can be viewed as a dynamical system wherein one wishes to solve for the population *state* at time t given the population state at some initial time, say $t = 0$. The population state is often stated in terms of a frequency vector, $\mathbf{P}(t)$, whose components $P_I(t)$, give the proportion of the population in state I at time t. Formally, one specifies a fitness landscape, $\{f_I\}$, a set of genetic operators, $\{\mathcal{O}_\alpha\}$, and an initial population, $\mathbf{P}(0)$; then, there exists an evolution operator, $\mathcal{H}(t) \equiv \mathcal{H}(t, \{f_I\}, \{\mathcal{O}_\alpha\}, \mathbf{P}(0))$, such that $\mathbf{P}(t) = \mathcal{H}(t, \{f_I\}, \{\mathcal{O}_\alpha\}, \mathbf{P}(0))$.
The states of the system are elements of a space - the *configuration space*, $\mathcal{C}$ - which may be finite or infinite dimensional. The nature of this space depends on how one parametrises the states of the system. For instance, for the familiar case of fixed-length binary strings of length ℓ, the states of a string may be placed in a one-to-one correspondence with the vertices of an ℓ-dimensional hypercube, ℓ-dimensional in the sense that it may be embedded in $\mathbb{R}^\ell$. $\mathcal{C}$ in this case is the set of vertices of the hypercube, which has dimension $|\mathcal{C}| = 2^\ell$ and which coincides with the *search space* Ω. If, however, the population includes more than one string, as is usual, the state space $\mathcal{C}$ for an EA is the space of all possible populations (note that in this more general case the notion of search and configuration space do not necessarily coincide). For n strings, $\mathcal{C}$ naturally has dimension $|\mathcal{C}| = 2^{n\ell}$. This representation, however, is somewhat over-specified. A more useful representation in Evolutionary Computation (EC) is that based on population vectors, which indicate, for each point in the search space, how many, or what proportion of, individuals sample that point. The population vector is a vector in $\mathcal{C}$ with 2^ℓ components, $(P_1, P_2, \ldots, P_{2^\ell})$. Thus, representing the state of the system via a frequency vector, in the dynamical systems point of view, one is interested in the time evolution of $\mathbf{P}(t)$ in $\mathcal{C}$. If it was necessary to identify individual strings - even if they were of the same genotype - then the frequency vector representation would not be adequate and appeal would be made to a representation that specifies the state of every individual in the population.
Now, $\mathcal{C}$, as a vector space or a simplex, will admit sets of basis vectors with respect to which the components of a vector, such as $\mathbf{P}$, may be defined. Linear transformations between one basis set and another may then also be defined. Such basis transformations are of particular relevance in dynamical systems, as it may well be that in a specific basis the dynamics looks particularly simple. This occurs very commonly in mechanical systems. A simple example of this would be a particle constrained to move on a sphere embedded in $\mathbb{R}^3$. In this case spherical polar coordinates (r, θ, ϕ) are much more natural than Cartesian coordinates (x, y, z), as the constraint that the particle moves on the sphere - $r = \text{constant}$ - is

much more naturally written in this coordinate system. Another simple mechanical example is that of a set of coupled harmonic oscillators, where the coordinates of the positions of the oscillators are coupled between neighbours and hence the dynamics looks complicated. Passing to a description in terms of the eigenmodes of the system however, gives a set of equations wherein the new coordinates on $\mathcal{C}$ are uncoupled, and therefore simple. Such simplifications are important in that, when they are found, they inevitably illuminate what are the appropriate effective degrees of freedom of the dynamical system, thereby aiding in both the qualitative understanding, as well as the quantitative analysis, of the system.

Simplifications of the dynamics in an appropriate basis are not the exclusive reserve of physical, mechanical systems, but also occur in the dynamics of EAs. For instance, it is well known that the dynamics of mutation for binary fixed-length strings looks simpler in the Walsh basis, as in that basis the mutation operator is diagonal ("simple"), the appropriate effective degrees of freedom being the Walsh modes [4, 3, 22, 23]. This simplification also occurs for higher cardinality alphabets, and is equivalent to a Fourier analysis of the system. Similarly, selection looks "simple" - diagonal and time independent - in the string basis.

The most complicated operator is recombination. Nevertheless, also in this case, for fixed-length strings, there exists a preferred basis within which crossover looks simplest - the Building Block basis (BBB) [16, 2]. The BBB is dual to the Taylor basis, as studied in [24], and has already been found useful in concrete calculations [6], as well as being interestingly related to geometric quantities in the theory of information [21]. In this basis the natural effective degrees of freedom are schemata, the Building Blocks of a particular string. For a given crossover mask there is a unique pair of conjugate Building Blocks that join together to form the string. For example, for $\ell = 3$: to construct a string 111 with a crossover mask $m = 001$, where $m_i = 0$ signifies take the ith bit of the offspring from the ith bit of the first parent, while $m_i = 1$ signifies take the bit from the second parent, there is one and only one Building Block combination that builds 111 with this mask - $11*$ and its conjugate $**1$. Furthermore, the linear coordinate transformation that allows one to pass to the BBB can be simply generated for arbitrary ℓ by taking the ℓ-fold tensor product of the transformation for $\ell = 1$. In fact, the whole machinery of recombination can be simply built up from the $\ell = 1$ case [2, 1]. The resulting form of the dynamical equations for a fixed-length GA with mutation, selection and homologous recombination is then seen to be identical to that found previously by coarse graining methods that have given rise to so-called "exact schema theorems" [14, 20, 15, 13, 19, 12, 18]. These, in their turn, have allowed for a reconciliation of previously, seemingly antagonistic formulations of the dynamics of GAs. Such coarse grained formulations have further been extended to variable-length linear representations and tree representations [7, 9, 8, 5, 10, 11, 17], thus leading to a unification of the theory underlying GAs and GP.

Given that coarse grained formulations have led to a great number of advances in the theory of GAs and GP, and that in the case of fixed-length strings a natural coarse graining can be implemented via a coordinate transformation, it is natural to ask if such basis transformations also exist in the more complicated cases of variable-length strings and trees. In particular: do there exist analogs of the BBB for variable-length strings and trees? The answer to this question is yes and forms the subject of this paper.

In section II we discuss briefly the spaces we will consider, i.e., the search space and the space of configurations of the objects under discussion, be they fixed- or variable-length strings or trees. In section III we formalise these notions, and note the recursive and tensorial nature of these spaces, showing how the search spaces for strings or trees of up to a certain size can be generated from the search spaces for simpler objects. This is quite transparent in the linear case where the search space for length ℓ strings can be built up as an ℓ-fold tensor product of the search space for length-one strings. As the search space for variable-length strings can be built up as a tensor sum of the search spaces for fixed-length strings of different sizes, the extension to this case is almost immediate. Trees, naturally, are somewhat more subtle. In this case, we exhibit a natural recursion relation built up in terms of tensor products and sums that relates the search space for programs of up to depth-d to that of programs of up to depth-$(d-1)$. Of course, as linear structures can be generated from the more general tree representation, this recursion relation also applies to strings and yields the already established relations for the linear case.

Next, in sections IV and V, we come to the crux of the matter: an analysis of some natural coordinate bases on the search spaces for strings and programs. We first consider the natural basis associated with the evolving objects themselves - strings and trees. In particular, we study a basis - the δ-basis - whose elements are characteristic functions that take value one on one search point and zero elsewhere. Any univariate function on the search space can naturally be written in terms of this basis. We then show that, due to the recursive structure of the search space itself, the δ-basis can be recursively constructed from the δ-basis for single bits or primitives of a given arity. Having established the δ-basis, by exploiting the recursive structure of the search space, we then go on to consider the BBB. Having eased the reader into the notation and concepts by starting with the case of fixed-length strings, here we dive straight into the general case of program trees. As there is a great deal of freedom as to which BBB one works with, we restrict attention here to a particular manifestation of the BBB wherein one arbitrary symbol or primitive is turned into a "don't care" symbol. We show that the coordinate transformation between the δ- and Building Block bases is effected by a transformation matrix that, due to the recursive structure of the search space, can once again be constructed from the transformation matrices that transform

single bits or primitives of a given arity. Finally, in section VI we draw some conclusions.

II. Search Space and Configuration Space

We will begin by considering the different search spaces of interest associated with fixed-length strings, variable-length strings and program trees respectively. Although trees are the most general representation, fixed- and variable-length strings being particular cases where the depth and/or arity of the trees is restricted, they are substantially more complicated than the simpler linear case. Hence, we will begin with the latter. However, we will not dwell on the case of fixed-length strings, as they have been extensively treated elsewhere but, rather, use them for illustration, as they are the least complicated and most easily understandable example.

Consider the case of fixed-length strings of length ℓ, where the ith locus is associated with an alphabet of cardinality $\mathcal{A}_i$. The search space, which we denote with Ω_ℓ^f, where the superscript ^{f} emphasises the fixed-length nature of its elements, has dimension $|\Omega_\ell^f| = \prod_{i=1}^{\ell} \mathcal{A}_i$. For example, for the standard fixed binary alphabet, $|\Omega_\ell^f| = 2^\ell$. The configuration space for n fixed-length strings is then $\mathcal{C} = (\Omega_\ell^f)^{\otimes n}$, which represents the n-fold tensor product of Ω_ℓ^f. As stated in the introduction, given that a representation in terms of frequency vectors is most useful and of interest in EC, we will concentrate on the states of the system as specified by the frequencies.

There are many different mathematical structures one may associate with Ω_ℓ^f. For instance, the space of real univariate functions over Ω_ℓ^f, $\mathcal{F}_{\Omega_\ell^f}$, is a real vector space, where a vector, $\mathbf{v}$, on $\mathcal{F}_{\Omega_\ell^f}$ has $|\Omega_\ell^f|$ components. Of particular interest in this paper are the different coordinate bases that span this vector space. Note that a vector on this space is an invariant object, i.e., it is basis independent. What do change, of course, are the components of the vector when one passes from one coordinate basis to another. An important example of a function/vector on the search space is the objective or *fitness function* $f : \Omega_\ell^f \to \mathbb{R}^+$, which associates points in the search space to non-negative fitness values. Thus, for $\ell = 2$ and $\mathcal{A} = 2$, Ω_2^f has 4 elements, $\{00, 01, 10, 11\}$, and the fitness function can be written as a row vector, $(f_{00}, f_{01}, f_{10}, f_{11})$. Another natural class of candidates that are of importance in EAs, is that of probability distributions defined over Ω_ℓ^f, such as the selection probability, the probability of generating offspring of a given genotype from a particular genetic operator, etc.[1] These probability distributions are simply functions of the form $f : \Omega_\ell^f \to [0, 1]$.

Probably the most important function for population based algorithms, like GAs and GP, is the composition of the population, which is a function, $\phi : \Omega_\ell^f \to [0, 1]$, that assigns a number (a proportion) to an element of Ω_ℓ^f. In the language of vectors this can be represented elegantly by an *incidence*, frequency or population, vector, $\mathbf{P} = (\phi_1, \phi_2, \ldots, \phi_{|\Omega_\ell^f|})$. The elements of an incidence vector contain integers which indicate how many individuals of each type are currently present in the population, while the frequency vector normalises these numbers by the population size. In an ordinary EA, this vector represents the state of the system, and so, the configuration space $\mathcal{C}$ is the space of all such vectors, i.e., the simplex.

Note that in the above we have indexed the components of the vectors using the integers, with as many integers as there are points in Ω_ℓ^f. This is not strictly necessary. An incidence vector, for example, can be indexed by the elements of the search space themselves. However, in this case, when writing down a vector (for example, for the purpose of doing some linear algebra operation) one needs to decide in which order to list the coordinates. This requires defining a total order on the elements of the search space, or, alternatively, defining a mapping between search space elements and integers (and then using the standard integer indexing for vectors). In the case of fixed-length binary strings, a natural order is the "odometer" ordering ($0\cdots00 < 0\cdots01 < 0\cdots10 < 0\cdots11 < \cdots < 1\cdots11$) and an equivalent natural mapping is the standard binary to decimal conversion ($0\cdots00 \leftrightarrow 0, 0\cdots01 \leftrightarrow 1, 0\cdots10 \leftrightarrow 2, 0\cdots11 \leftrightarrow 3, \cdots$), and, indeed, these have been used widely in the theory of GAs. However, for more complex spaces there may not be an obvious natural order, and so, if one wants to use vectors to represent information about the problem, algorithm or search space, one needs to explicitly define an order. Of course, results - theoretical or experimental - cannot depend on this arbitrary labelling.

If an incidence vector is indexed by the elements of the search space itself, one can interpret it as a function of the form $f : \Omega_\ell^f \to \mathbb{N}$ (or $f : \Omega_\ell^f \to \mathbb{Z}$, depending on the state representation chosen) plus a total order over Ω_ℓ^f or, equivalently, a bijective indexing function $g : \Omega_\ell^f \to \{1, 2, \cdots, |\Omega_\ell^f|\}$. By the composition, $f(g^{-1}(x)) : \{1, 2, \cdots, |\Omega_\ell^f|\} \to \mathbb{R}$ one then obtains an ordinary vector.

If one defines an indexing function g over the search space, then all of the functions mentioned in this section – fitness functions, selection probabilities for the elements of the search space, etc. – can conveniently be represented using ordinary vectors on $\mathbb{R}^{|\Omega_\ell^f|}$.

As well as univariate functions, which naturally translate into vectors on Ω_ℓ^f, we can also introduce multivariate functions. A natural bivariate function, for instance, is the mutation probability from one string to another. Just as univariate functions map naturally into vectors, so bivariate functions naturally map into matrices.[2]

[1] Strictly speaking probabilities do not form a vector space due to the constraint that they must sum to one. In this case, they form a *simplex*; a simplex in an n-dimensional vector space being the convex hull of any $n + 1$ points that do not lie in any hyperplane of dimension $< n$.

[2] More naturally they map into $(1, 1)$ tensors. Similarly, recombination is naturally written in terms of a tri-variate function - the probability that two

Finally, we sometimes need to talk about subsets of elements of the search space, and whether or not a point belongs to a set. *Schemata* are special types of subsets of the search space, which are typically represented using patterns of symbols (e.g. 10**1, which represents all strings of length 5 starting with 10 and ending with 1, or  which represents all programs where any two terminal symbols are added together). In the case of strings/trees or schemata, another interesting function on Ω_ℓ^f is the *characteristic* or membership function - $g : \Omega_\ell^f \to \{0, 1\}$, which returns 1 when applied to an element that belongs to the set of interest, and 0 otherwise.

Turning now to the case of variable-length strings, we denote the search space $\Omega^v_{\ell_{\max}}$, where $\ell_{\max}$ is the maximum string length considered[3] and v refers to the variable length of the elements in the search space. In this case, the dimension of $\Omega^v_{\ell_{\max}}$ is $|\Omega^v_{\ell_{\max}}| = \sum_{\ell=1}^{\ell_{\max}} \prod_{i=1}^{\ell} \mathcal{A}_{\ell i}$, where $\mathcal{A}_{\ell i}$ is the cardinality of the alphabet associated with the ith locus of strings of length ℓ. For a fixed alphabet of cardinality $\mathcal{A}$, this simplifies to $|\Omega^v_{\ell_{\max}}| = \mathcal{A}(\mathcal{A}^{\ell_{\max}} - 1)/(\mathcal{A} - 1)$. Just as for fixed-length strings one may naturally introduce multivariate functions, vectors, schemata etc. As for fixed-length strings, if, for a population of variable-length strings of size n, one is interested in the individual state of every string identified, as distinct from any other, i.e. not just identified by genotype, then the natural configuration space $\mathcal{C}$ is of dimension $\left|\left(\Omega^v_{\ell_{\max}}\right)^{\otimes n}\right|$. If we are satisfied with the frequency or incidence vector representation then we have a much smaller configuration space.

Finally, for trees, we denote the search space by Ω_d^t, where d refers to the maximum tree depth. $|\Omega_d^t|$ is then the number of programs of depth up to d, which we denote n_d. This number is not as simple as in the linear case but, as we will see in the next section, satisfies the recursion relation

$$n_0 = |\mathcal{P}_0| \qquad n_d = \sum_{a=0}^{a_{\max}} |\mathcal{P}_a| \times (n_{d-1})^a \tag{1}$$

where $|\mathcal{P}_a|$ is the number of primitives of arity a, $a_{\max}$ being the maximum arity. As with the linear case one can define the space of univariate functions over Ω_d^t, $\mathcal{F}_{\Omega_d^t}$, in order to define, for example, the fitness function for programs. In the same way, one may generally define vectors, including the incidence and frequency vectors, more general multivariate functions and schemata. Additionally, if we wish to distinguish program trees above and beyond their genotypic representation then, for a population of n programs, the natural configuration space is $\mathcal{C} = (\Omega_d^t)^{\otimes n}$. Once again though, normally we will be happy with the much smaller space of frequency/population vectors. As mentioned, linear strings are just a special case of trees, where the maximum arity is 1 and the relation between d and $\ell_{\max}$ is $d+1 = \ell_{\max}$. Equation (1) reduces to $|\Omega_\ell^v|$ for variable-length strings in this case, while the value for fixed-length strings can be found by considering $(n_d - n_{d-1})$.

parents J and K give rise to an offspring I - that maps into a $(2, 1)$ tensor. See [2] for a discussion of this.

[3] One can, of course, consider the limit $\ell_{\max} \to \infty$

III. The Recursive Structure of the Search Space

In the previous section, we defined abstractly the search spaces for fixed-length strings, variable-length strings and trees, showing how one could naturally associate uni- and multivariate functions (equivalent to vectors and tensors), schemata etc., with the space. In this section, we wish to show that the search spaces themselves all have natural recursive structures, where the space is built up from simpler, lower-dimensional objects. As will be shown later, this recursive structure greatly simplifies the analysis of these spaces and manifests itself in the appearance of tensor products and sums.

To illustrate this recursive structure, we first consider the case of fixed-length strings. In this case, Ω_ℓ^f has a natural recursive structure based on the concept of the direct (tensor) product, wherein Ω_ℓ^f can be generated from $\Omega_1^f(i)$, the search space associated with the ith bit, which is of dimension $|\Omega_1^f(i)| = \mathcal{A}_i$. Thus,

$$\begin{aligned} \Omega_\ell^f &= \Omega_1^f(1) \otimes \Omega_1^f(2) \otimes \ldots \otimes \Omega_1^f(\ell) \\ &\equiv \bigotimes_{i=1}^{\ell} \Omega_1^f(i) \\ |\Omega_\ell^f| &= \prod_{i=1}^{\ell} |\Omega_1^f(i)| \end{aligned} \tag{2}$$

where $\otimes$ and $\bigotimes$ represent a tensor product of the spaces. In the simpler case of a fixed cardinality alphabet, we can write $\Omega_\ell^f = (\Omega_1^f)^{\otimes \ell}$, where $\otimes \ell$ as a superscript represents the ℓ-fold tensor product, in this case, of Ω_1^f, the search space associated with one bit.

For variable-length strings the story is similar. In this case, there is first a natural decomposition of $\Omega^v_{\ell_{\max}}$ into the form

$$\Omega^v_{\ell_{\max}} = \bigoplus_{\ell=1}^{\ell_{\max}} \Omega_\ell^f$$

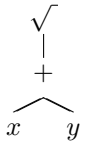
where $\bigoplus$ represents a tensor sum of the spaces. Thus, the search space of variable-length strings is just the tensor (direct) sum of the search spaces for fixed-length strings of different lengths. One can then use (2) to find

$$\Omega^v_{\ell_{\max}} = \bigoplus_{\ell=1}^{\ell_{\max}} \bigotimes_{i=1}^{\ell} \Omega_1^f(i)$$

Thus, armed only with knowledge of the one-bit search spaces we can easily generate the search spaces for fixed-

or variable-length strings. Obviously, the fixed alphabet case is especially simple.

Passing now to the case of trees, we divide the primitive set $\mathcal{P}$ into subsets $\mathcal{P}_a$, with elements p_{ai}, for $i = 1, \cdots, |\mathcal{P}_a|$, of primitives of arity a, for $a = 0, 1, \cdots a_{\max}$, where $a_{\max}$ is the maximum arity of the primitives in $\mathcal{P}$. So, $\mathcal{P} = \bigcup_a \mathcal{P}_a$. The elements of the search space can be represented as trees, with nodes labelled with primitives in such a way that labels have the appropriate arity for the structure of the tree, e.g.



Alternatively, and equivalently, the elements of the search space can be seen as sequences of primitives, such as $\surd + x\, y$, of appropriate arity so as to form a valid syntax tree. Formally, a sequence is valid if it is part of a language with the following grammar:

$$
\begin{array}{rcl}
E & \rightarrow & \mathcal{P}_0 \\
E & \rightarrow & \mathcal{P}_1 E \\
E & \rightarrow & \mathcal{P}_2 E E \\
 & \vdots & \\
E & \rightarrow & \mathcal{P}_{a_{\max}} \overbrace{E \cdots E}^{a_{\max}} \\
\mathcal{P}_0 & \rightarrow & p_{01} \,|\, p_{02} \,|\, \cdots \,|\, p_{0|\mathcal{P}_0|} \\
\mathcal{P}_1 & \rightarrow & p_{11} \,|\, p_{12} \,|\, \cdots \,|\, p_{1|\mathcal{P}_1|} \\
 & \vdots & \\
\mathcal{P}_{a_{\max}} & \rightarrow & p_{a_{\max}1} \,|\, p_{a_{\max}2} \,|\, \cdots \,|\, p_{a_{\max}|\mathcal{P}_{a_{\max}}|}
\end{array}
$$

Note that it must be the case that $|\mathcal{P}_0| > 0$, that is, we must have at least one terminal symbol. In this grammar we used the prefix notation typical of Lisp and other languages, but we did not include any syntactic sugar, such as brackets. We will, however, in the following, occasionally represent programs using brackets, since this makes the correspondence between expressions (programs) and their syntax tree clearer. So, for example, $- + x\, y\, x$ would be written as $(-\,(+\,x\,y)\,x)$.

The space Ω_d^t of program trees that can be constructed using the primitives in $\mathcal{P}$, and are of depth up to d, is built using the following recursion: [4]

$$\Omega_0^t = \mathcal{P}_0 \qquad \Omega_d^t = \bigoplus_{a=0}^{a_{\max}} \mathcal{P}_a \otimes (\Omega_{d-1}^t)^{\otimes a}$$

Let $n_d = |\Omega_d^t|$ be the number of different programs of depth at most d (where a program including a single terminal has depth 0) that can be constructed using primitives from $\mathcal{P}$. Clearly we have:

$$n_0 = |\mathcal{P}_0| \qquad n_d = \sum_{a=0}^{a_{\max}} |\mathcal{P}_a| \times (n_{d-1})^a$$

Example 1 *Search space for the case of trees with primitives* $\mathcal{P} = \{x, y, \surd, +, \times\}$. We have $a_{\max} = 2$, $\mathcal{P}_0 = \{x, y\}$, $\mathcal{P}_1 = \{\surd\}$ and $\mathcal{P}_2 = \{+, \times\}$. Then, using prefix notation, $\Omega_0^t = \{x, y\}$ and

$$
\begin{array}{rl}
\Omega_1^t & \\
= & \bigoplus_{a=0}^{2} \mathcal{P}_a \otimes (\Omega_0^t)^{\otimes a} \\
= & \mathcal{P}_0 \oplus \mathcal{P}_1 \otimes \Omega_0^t \oplus \mathcal{P}_2 \otimes \Omega_0^t \otimes \Omega_0^t \\
= & \{x, y\} \oplus \{\surd\} \otimes \{x, y\} \oplus \{+, \times\} \otimes \{x, y\} \\
 & \otimes \{x, y\} \\
\Omega_2^t & \\
= & \bigoplus_{a=0}^{2} \mathcal{P}_a \otimes (\Omega_1^t)^{\otimes a} \\
= & \{x, y\} \oplus \{\surd\} \otimes \Omega_1^t \oplus \{+, \times\} \otimes \Omega_1^t \otimes \Omega_1^t \\
= & \{x, y\} \\
\oplus & \{\surd\} \otimes \{x, y\} \\
\oplus & \{\surd\} \otimes \{\surd\} \otimes \{x, y\} \\
\oplus & \{\surd\} \otimes \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{\surd\} \otimes \{x, y\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{x, y\} \otimes \{\surd\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{\surd\} \otimes \{x, y\} \otimes \{\surd\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \otimes \{\surd\} \\
 & \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{x, y\} \otimes \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{\surd\} \otimes \{x, y\} \otimes \{+, \times\} \otimes \{x, y\} \\
 & \otimes \{x, y\} \\
\oplus & \{+, \times\} \otimes \{+, \times\} \otimes \{x, y\} \otimes \{x, y\} \otimes \{+, \times\} \\
 & \otimes \{x, y\} \otimes \{x, y\}
\end{array}
$$

where, in the last step, we have distributed all sums over products. Clearly, there are 13 subspaces that are summed up tensorially. It is easy to see that the total number of programs, n_2, is $2+2+2+8+8+8+32+8+8+32+32+32+128 = 302$, which can be more easily calculated using the recursion previously introduced obtaining

$$
\begin{array}{rcl}
n_0 & = & 2 \\
n_1 & = & 2 + 1 \times (n_0) + 2 \times (n_0)^2 = 12 \\
n_2 & = & 2 + 1 \times (n_1) + 2 \times (n_1)^2 = 302
\end{array}
$$

□

[4] In this and the previous formulas $\otimes$ can also be interpreted as the Cartesian product and $\oplus$ as the set union operation.

The example above suggests that an alternative but equivalent characterisation of Ω_d^t based on tree shapes is possible. Let s_d be the number of different shapes of depth at most d that can be constructed using primitives from $\mathcal{P}$. This is given by the recursion

$$s_0 = 1 \qquad s_d = \sum_{a=0}^{a_{\max}} (s_{d-1})^a \times \delta(|\mathcal{P}_a| > 0)$$

Let us then enumerate all shapes of up to depth d, and let S_i be the ith shape in the set. Let $n(S, a)$ be a function that returns the number of nodes of arity a in shape S, and $\mathbf{ar}(x)$ a function that returns the arity of node x in a shape. We can then write

$$\Omega_d^t = \bigoplus_{i=1}^{s_d} \bigotimes_{j \in S_i} \mathcal{P}_{\mathbf{ar}(j)}$$

where by $j \in S_i$ we mean that the tensor product ranges over the nodes, denoted by j, in shape S_i. This gives us another way of computing n_d:

$$n_d = \sum_{i=1}^{s_d} \prod_{a=0}^{a_{\max}} |\mathcal{P}_a|^{n(S_i,a)}$$

Note that the semantics of the primitives is totally irrelevant as far as the definition of the search space is concerned — it only matters during fitness evaluation. The primitives are simply labels for the nodes of the trees in the search space. The only thing that really matters is their arity. In this sense the arity of a primitive should not be thought of as the number of arguments required by the primitive (seen as a function), but rather as a description of how the primitive seen as a node is meant to connect to other primitives (nodes). Also, we can even have primitives with the same name and different arities (like the unary minus sign and the binary minus sign in standard algebra), as long as there is a way of determining which is which (i.e. from their arity or from their position in a tree).

Therefore, the primitive set $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_2$ with $\mathcal{P}_0 = \{0, 1\}$, and $\mathcal{P}_2 = \{0, 1\}$ is a perfectly valid set. The difference between the primitives in $\mathcal{P}_0$ and the primitives in $\mathcal{P}_2$ is that the primitives in $\mathcal{P}_0$ are followed by *no* other primitive, while the primitives in $\mathcal{P}_2$ are followed by *two* arbitrary sequences of primitives (subtrees). In this example Ω_d^t is the space of all binary trees of depth up to d with binary-labelled nodes.

Of particular interest for us is the case where only zero-ary and unary primitives are allowed, i.e. $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_1$. In this situation $\Omega_d^t \equiv \Omega_{\ell_{\max}}^v$ is the space of variable-length linear structures of length at most $\ell_{\max} = d + 1$. These can represent programs (e.g. in assembler code) or variable length strings, as in the case $\mathcal{P}_0 = \{0, 1\}$ and $\mathcal{P}_1 = \{0, 1\}$, where Ω_{d+1}^v is the space of variable-length bit strings of size up to $d + 1$. The difference $\Omega_d^t \ominus \Omega_{d-1}^t \equiv \Omega_\ell^f$ then represents the space of linear structures of length exactly $\ell = d + 1$. If $|\mathcal{P}_0| = |\mathcal{P}_1| = \mathcal{A}$, as is the case for a search space of variable-length strings drawn from an alphabet of cardinality $\mathcal{A}$, we have $n_d = \sum_{k=0}^{d} \mathcal{A}^{k+1} = (\mathcal{A}^{d+1} - 1)\mathcal{A}/(\mathcal{A} - 1)$.

If one orders the sets $\mathcal{P}_i$ that make up the primitive set, it is possible to identify their elements using integers. The chosen order is not important, as long as one is consistent. For example, if we have a primitives set $\mathcal{P}_1 = \{C, T, G, A\}$ we can identify C with 0, T with 1, etc., but any other assignment would also work. Similarly, if we reconsider the previous example, where $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2$ with $\mathcal{P}_0 = \{x, y\}$, $\mathcal{P}_1 = \{\surd\}$ and $\mathcal{P}_2 = \{+, \times\}$, we can index the primitives as follows: $x \leftrightarrow 0$, $y \leftrightarrow 1$, $\surd \leftrightarrow 0$, $+ \leftrightarrow 0$ and $\times \leftrightarrow 1$. Note that we are reusing the same integer index for different primitives. This is fine as long as we have a way of telling to which arity group ($\mathcal{P}_i$) an index refers to.[5]

In effect, this indexing operation creates a isomorphism between the original search space and the space of trees with integer-labelled nodes. The simplest case is, of course, when only primitives of arity 0 and 1 are used. In this case, the search space is isomorphic to the space of variable-length sequences of integers $(x_0, ...x_{d-1}, x_d)$ with $x_i \in \{0, \cdots, |\mathcal{P}_1|\}$ for $0 \le i < d$ and $x_d \in \{0, \cdots, |\mathcal{P}_0|\}$. So, if the search is further limited to sequences of primitives of length ℓ, the search space is isomorphic to an ℓ-dimensional cubic lattice (a hypercube in the case $\mathcal{P}_0 = \mathcal{P}_1 = \{0, 1\}$ of binary strings).

IV. The δ-basis

As we have emphasised, the states of populations of strings or programs are naturally represented by particular functions (incidence or frequency vectors) on the search space. We are therefore led to look for natural bases in which these functions (and others) may be expanded. Let us denote by $\mathcal{F}_\Omega$ the space of all univariate functions on our search space, Ω, whatever it may be, to $\mathbb{R}$. We want to introduce a basis for this space.

A natural basis for $\mathcal{F}_\Omega$ is formed from the *characteristic* functions of Ω itself, which are $|\Omega|$ unit amplitude δ-like functions, each with support on a single element of Ω. We call this basis of characteristic functions the *δ-basis*[2], $B_\delta(\Omega)$. If we index this basis using $v \in \Omega$, the functions in the δ-basis have the form

$$\delta_v(x) = \delta(x = v),$$

where $\delta(\texttt{expr})$ is a function that returns 1 if expression `expr` is true, and 0 otherwise.

The expansion of an arbitrary function f in the δ-basis is particularly simple, the coefficient of a basis element (delta function) being the value of f at the corresponding point of Ω,

$$f = \sum_v f(v)\delta_v\,, \tag{3}$$

[5] Clearly for primitive sets that are subsets of the space of integers, such as $\mathcal{P}_1 = \{0, 1\}$, there is a natural order, which we may want to respect when indexing their elements.

where v ranges over all elements of the search space and δ_v has support only on v.
To obtain a more explicit representation of $B_\delta(\Omega)$ we need to consider a specific search space. Once again, for the purposes of illustration we begin with fixed-length strings. In this case the δ-basis consists of $|\Omega_\ell^f|$ characteristic functions, each one associated with a point in Ω_ℓ^f. The points of Ω_ℓ^f may be placed on an ℓ-dimensional lattice with $\mathcal{A}_i$ points in the ith direction. If the coordinates of a point v on the lattice are $(v_1, v_2, \ldots, v_\ell)$, then $\delta_v = \delta^1_{v_1}\delta^2_{v_2}\ldots\delta^\ell_{v_\ell}$ where $\delta^i_{v_i}$ is the characteristic function for the ith locus with coordinate v_i in the ith direction. An explicit representation of $\delta^i_{v_i}$ is [1]

$$\begin{aligned}\delta^i_{v_i} &\equiv \mathcal{N}_i x_i(x_i-1)\ldots\widehat{(x_i-v_i)}\ldots(x_i-\mathcal{A}+1)\\ \mathcal{N}_i^{-1} &\equiv (-1)^{\mathcal{A}-v_i-1}v_i!(\mathcal{A}-v_i-1)!\end{aligned}$$

(hats denote omission). Notice that $\delta^i_{v_i}$ is the characteristic function of the $x_i = v_i$ hyperplane. Thus, we may write the δ-basis as

$$B_\delta(\Omega_\ell^f) = \{\delta_{v(1)}, \delta_{v(2)}, \ldots, \delta_{v(|\Omega_\ell^f|)}\}$$

where $\delta_{v(k)} = \delta^1_{v_1(k)}\delta^2_{v_2(k)}\ldots\delta^\ell_{v_\ell(k)}$ is the characteristic function for the vertex represented by the vector $v(k)$.
Example 2 *δ and vertex bases for $\ell = 2$, $\mathcal{A} = 3$ strings* The δ-basis is

$$\begin{aligned}B_\delta &= \{\delta_{00}, \delta_{01}, \ldots, \delta_{22}\}\\ &= \{\delta^1_0\delta^2_0, \delta^1_0\delta^2_1, \ldots, \delta^1_2\delta^2_2\}\\ &= \{(x_1-1)(x_1-2)(x_2-1)(x_2-2)/4,\\ &\quad -(x_1-1)(x_1-2)x_2(x_2-2)/2, \ldots,\\ &\quad x_1(x_1-1)x_2(x_2-1)/4\}\,. \qquad (4)\end{aligned}$$

□

In section III we saw that the search space could be written as a tensor product, $\Omega_\ell^f = \bigotimes_{i=1}^\ell \Omega_1^f(i)$. We can thus use the rule for constructing bases on tensor product spaces to construct the basis for Ω_ℓ^f from the bases for $\Omega_1^f(i)$. For example, if the bases for spaces V and W are $B_V = \{v_1, \ldots, v_n\}$ and $B_W = \{w_1, \ldots, w_m\}$ respectively, then the basis $B_{V\otimes W}$ for the tensor product $V \otimes W$ is taken to be

$$B_{V\otimes W} = \{v_1\otimes w_1, \ldots, v_1\otimes w_m, v_2\otimes w_1, \ldots, v_n\otimes w_m\}\,,$$

The δ-basis for $\Omega_1^f(i)$ is $B_\delta(\Omega_1^f(i)) = \{\delta^i_{v_i}(0), \delta^i_{v_i}(1), \ldots, \delta^i_{v_i}(\mathcal{A}-1)\}$ Thus, a basis for Ω is $B_\delta(\Omega_1^f(1)\otimes\Omega_1^f(2)\otimes\ldots\otimes\Omega_1^f(\ell))$. If we order the elements of $B_\delta(\Omega_1^f(i))$ into a vector then we can write

$$\begin{aligned}B_\delta(\Omega_\ell^f) &= B_\delta(\Omega_1^f(1))\otimes B_\delta(\Omega_1^f(2))\otimes\ldots\otimes B_\delta(\Omega_1^f(\ell))\\ &\equiv \bigotimes_{i=1}^{\ell} B_\delta(\Omega_\ell^f) \qquad (5)\end{aligned}$$

and so we see how the δ-basis for length-ℓ strings may be constructed from the bases for length-one strings. For a fixed alphabet size $B_\delta(\Omega_\ell^f) = (B_\delta(\Omega_1^f))^{\otimes\ell}$.
The case of variable-length strings follows straightforwardly from the fixed-length case using the fact that $\Omega^v_{\ell_{\max}}$ is a tensor sum of the Ω_ℓ^f for fixed-length strings. Thus,

$$B_\delta(\Omega^v_{\ell_{\max}}) = \bigoplus_{\ell=1}^{\ell_{\max}} B_\delta(\Omega_\ell^f)$$

where $B_\delta(\Omega_\ell^f)$ can be written in terms of the $\ell = 1$ δ-basis using (5), and the basis, $B_{V\oplus W}$, for a tensor sum $V \oplus W$ is taken to be

$$B_{V\oplus W} = \{v_1, v_2, \ldots, v_n, w_1, \ldots, w_m\}\,.$$

We now pass to the more complicated case of programs. In this case, in order to give a more precise description of the structure of the functions in the δ-basis we need to be able to analyse the composition of the elements of the search space – trees – node by node. Since trees have a natural recursive structure, as was seen in section III, we will achieve this by using recursion.
Let v be an element of the search space Ω_d^t. We define the following functions:

- $\mathbf{rt}(v)$ which returns the primitive labelling the root node of v,
- $\mathbf{ch}(v, i)$ which returns the ith subtree of the root of v,
- $\mathbf{ar}(x)$ which returns the arity of the primitive labelling a node.

So, for example, if $v = (-\,(+\,x\,y)\,x)$, then $\mathbf{rt}(v)$ is the primitive "$-$", $\mathbf{ch}(v,1)$ is the subtree $(+\,x\,y)$, $\mathbf{rt}(\mathbf{ch}(v,1))$ is the primitive "$+$" and $\mathbf{ar}(\mathbf{rt}(v)) = 2$.
So, for each $v \in \Omega_d^t$ we have a basis function $\delta_v \in B_\delta(\Omega_d^t)$ recursively defined as follows:

$$\delta_v(x) \;=\; \delta\left(v \overset{r}{=} x\right) \times \prod_{i=1}^{\mathbf{ar}(\mathbf{rt}(v))} \delta_{\mathbf{ch}(v,i)}(\mathbf{ch}(x,i))$$

where the comparison operation $v \overset{r}{=} x$ is defined as $\mathbf{rt}(v) = \mathbf{rt}(x)$ and we use the convention $\prod_{i=1}^0 \cdot = 1$ which prevents an infinite recursion. The recursion terminates prematurely when there is a mismatch of arity or label between two nodes in x and v, in which case $\delta_v(x) = 0$. If, instead the depth-first comparison of x and v is successful, then $\delta_v(x) = 1$.[6]
This formulation is important because it shows how effectively the δ-basis $B_\delta(\Omega_d^t)$ is constructed by combining tensorially $B_\delta(\Omega_{d-1}^t)$ and the bases $B_\delta(\mathcal{P}_a)$ (for $a = 0, 1, \cdots$)

[6] One can easily see this by unrolling the recursion. This produces an expression for $\delta_v(x)$ which is a product of $\delta(\mathtt{expr})$-type expressions (each one checking that a particular node in the two trees matches) and 1's (these derive from the terminating condition $\prod_{i=1}^0 \cdot = 1$). For example, $\delta_{(v_1v_2v_3)}((x_1x_2x_3)) = \delta(v_1 = x_1)\times\delta(v_2 = x_2)\times 1\times\delta(v_3 = x_3)\times 1$.

of characteristic functions over the primitive set. That is

$$B_\delta(\Omega_d^t) = \bigoplus_{a=0}^{a_{\max}} B_\delta(\mathcal{P}_a) \otimes B_\delta(\Omega_{d-1}^t)^{\otimes a}$$

If one could unroll the recursion, one would see that $B_\delta(\Omega_d^t)$ is a tensor sum with as many terms as there are possible program shapes. Each term in the sum is a tensor product containing as factors the δ-bases for the subsets $\mathcal{P}_a$ of the primitive set.[7]

As before, we can also recover the linear case as a special version of the tree case. For instance, for variable-length linear structures with $|\mathcal{P}_1| = |\mathcal{P}_0|$, we can unroll the recursion in the definition of $\delta_v(x)$, obtaining the following equivalent definition for the elements of the δ basis:

$$\delta_v(x) \quad = \quad \delta(|v| = |x|) \prod_{i=1}^{|v|} \delta(v_i = x_i)$$

where $|v|$ and $|x|$ are the number of primitives in v and x, respectively, and subscripts are used to index the primitives from the root node (leftmost primitive) to the leaf node (rightmost primitive). Likewise, we can unroll the recursive tensorial definition of $B_\delta(\Omega_d^t)$ obtaining:

$$\begin{aligned} B_\delta(\Omega_d^t) &\equiv B_\delta(\Omega_{d+1}^v) \\ &= \Big(1 \oplus B_\delta(\mathcal{P}_1) \oplus (B_\delta(\mathcal{P}_1))^{\otimes 2} \\ &\quad \oplus \ldots \oplus (B_\delta(\mathcal{P}_1))^{\otimes d}\Big) \otimes B_\delta(\mathcal{P}_0) \\ &= \left(\bigoplus_{i=0}^{d} (B_\delta(\mathcal{P}_1))^{\otimes i}\right) \otimes B_\delta(\mathcal{P}_0) \end{aligned}$$

where we should remember the subtle difference between the size (the number of primitives) $|x|$ of a linear structure x and its depth $d(x)$, when the structure is seen as a tree, i.e., that $d(x) = |x| - 1$.

As mentioned earlier, if we index the elements of the primitive set, in the linear case, the search space is isomorphic to the space of variable-length sequences of integers $(x_0, ...x_{d-1}, x_d)$ with $x_i \in \{0, \cdots, |\mathcal{P}_1|\}$ for $0 \le i < d$ and $x_d \in \{0, \cdots, |\mathcal{P}_0|\}$. In this case, we can explicitly represent characteristic functions using polynomials. To illustrate this, let us consider the special case $\mathcal{P}_0 = \mathcal{P}_1 = \{0, 1, \cdots \mathcal{A}-1\}$, and $d = \ell - 1$. That is, the search space is the space of variable-length strings of up to length ℓ built from an alphabet of cardinality $\mathcal{A}$.

We can see in the following examples how our recursion formula for trees, restricted to the arity-one case, yields the same basis as the variable-length δ-basis written down earlier.

[7] Bases for $\mathcal{F}_\Omega$ are *sets* of functions. When we say that we create them combining, via tensor sums and tensor products, other, lower-dimensional, bases we assume that the bases are represented as $(1, 0)$ tensors. Alternatively, one can treat them as sets, but in this case we must interpret $\otimes$ as Cartesian product and $\oplus$ as the set union.

Example 3 *δ-basis for variable-length binary strings.* The case $\mathcal{A} = 2$ for binary strings is particularly simple, as in this case everything can be built from two characteristic functions: $\delta(1 = x_i) = x_i$ and $\delta(0 = x_i) = \bar{x}_i$, where $\bar{x}_i = (e - x_i)$ is the bit complement of x_i, e here being understood as the function (over the primitive set) that always returns 1. In this case, if we use the standard ("odometer"-like) order for binary strings, and we take $\ell = 2$, we have

$$\begin{aligned} & B_\delta^v(\Omega_2^v) \\ &= B_\delta(\{0,1\}) \oplus B_\delta(\{0,1\})^{\otimes 2} \\ &= \{\bar{x}_1^{(1)}, x_1^{(1)}\} \oplus \{\bar{x}_1^{(2)}, x_1^{(2)}\} \otimes \{\bar{x}_2^{(2)}, x_2^{(2)}\} \\ &= \{\bar{x}_1^{(1)}, x_1^{(1)}\} \oplus \{\bar{x}_1^{(2)}\bar{x}_2^{(2)}, \bar{x}_1^{(2)}x_2^{(2)}, x_1^{(2)}\bar{x}_2^{(2)}, x_1^{(2)}x_2^{(2)}\} \\ &= \{\bar{x}_1^{(1)}, x_1^{(1)}, \bar{x}_1^{(2)}\bar{x}_2^{(2)}, \bar{x}_1^{(2)}x_2^{(2)}, x_1^{(2)}\bar{x}_2^{(2)}, x_1^{(2)}x_2^{(2)}\} \end{aligned}$$

where superscripts of the form $^{(i)}$ have been used to represent length-class. □

Example 4 *δ basis for variable-length linear structures with $\mathcal{A} = 3$ and $\ell = 2$* This time the δ-basis, in the odometer ordering, is given by

$$\begin{aligned} & B_\delta(\Omega_2^v) \\ &= \{(x_1^{(1)} - 1)(x_1^{(1)} - 2)/2, -x_1^{(1)}(x_1^{(1)} - 2), \\ &\quad x_1^{(1)}(x_1^{(1)} - 1)/2, \\ &\quad (x_1^{(2)} - 1)(x_1^{(2)} - 2)(x_2^{(2)} - 1)(x_2^{(2)} - 2)/4, \\ &\quad -(x_1^{(2)} - 1)(x_1^{(2)} - 2)x_2^{(2)}(x_2^{(2)} - 2)/2, \\ &\quad \ldots, x_1^{(2)}(x_1^{(2)} - 1)x_2^{(2)}(x_2^{(2)} - 1)/4\}\,. \end{aligned}$$

□

We have considered here the δ-basis as being the natural basis for expanding any univariate function on the search space or, equivalently, any vector. We have mentioned that also of interest, when one considers mutation and recombination, are multivariate functions, which in their turn have a geometrical interpretation as tensors. We emphasise, without going into further detail, that the δ-basis can also serve to build up bases for these higher order tensors too. For instance, for a $(0, 2)$ tensor the natural basis would be $B_\delta(\Omega) \otimes B_\delta(\Omega)$. For details the reader may consult [1].

V. Building Block Basis

As we have seen, the δ-basis for $\mathcal{F}_\Omega$ is formed from the characteristic functions of single elements (singletons) of Ω. We know that there is a benefit in defining a basis using characteristic functions for subsets of Ω other than singletons in the case of fixed-length strings. The *Building Block Basis* (BBB), which we will denote as $B_\beta(\Omega)$, does exactly this. One may therefore wonder if these benefits extend to the more general case of variable-length strings and trees.

While in the previous sections we have proceeded from the particular (fixed- and variable-length strings) to the general (trees), in this section we will develop the theory directly for the space of trees and we will then specialise to the simpler cases to better exemplify it.

In the BBB the subsets of the search space are *schemata* which are syntactically represented using either trees or sentences from the following grammar:

$$\begin{aligned} E &\rightarrow \mathcal{P}_0 \,|\, \mathcal{P}_1 E \,|\, \cdots \,|\, \mathcal{P}_{a_{\max}} E \cdots E \\ \mathcal{P}_0 &\rightarrow p_{01} \,|\, p_{02} \,|\, \cdots \,|\, p_{0|\mathcal{P}_0|} \,|\, *_0 \\ \mathcal{P}_1 &\rightarrow p_{11} \,|\, p_{12} \,|\, \cdots \,|\, p_{1|\mathcal{P}_1|} \,|\, *_1 \\ &\vdots \\ \mathcal{P}_{a_{\max}} &\rightarrow p_{a_{\max}1} \,|\, p_{a_{\max}2} \,|\, \cdots \,|\, p_{a_{\max}|\mathcal{P}_{a_{\max}}|} \,|\, *_{\max} \end{aligned}$$

Note that this is exactly the same syntax as for the elements of the search space, but we have extended the lexicon with the symbols "$*_a$" (for $a = 0, 1, \cdots, a_{\max}$). These are interpreted as "don't care" symbols that stand for exactly one primitive of arity a. So, semantically a schema $H = h_1 h_2 ...$ is the set of all programs that match its syntactic representation (i.e., all programs with the same structure as $h_1 h_2 ...$ and with exactly the same primitives as $h_1 h_2 ...$ for all the non-"don't care" symbols). An alternative interpretation for the symbols $*_a$ is to consider them as the sets of all valid primitives of arity a, that is $*_a = \mathcal{P}_a$.

Although the number of possible schemata of up to depth d is much smaller than the number of elements of the powerset (the set of all subsets) of Ω_d^t, this number is substantially bigger than n_d. So, clearly, a basis for $\mathcal{F}_{\Omega_d^t}$ could not contain the characteristic functions of all possible schemata. Indeed, the BBB selects only one specific subset of schemata. In the simplest case, the set is constructed as follows:

- For each subset of a given arity, $\mathcal{P}_a$, of the primitive set, where $a = 0, \cdots, a_{\max}$, we choose a primitive p_a, and we replace it with the symbol $*_a$.
- We generate all valid trees/sentences in this new language. We call this set $\mathcal{H}_d^t$. Some trees in $\mathcal{H}_d^t$ will be ordinary elements of the search space (e.g., programs), but most will represent schemata (sets of elements).
- $B_\beta(\Omega_d^t)$ is the set of characteristic functions for the elements $\mathcal{H}_d^t$ (interpreted as subsets of Ω_d^t).

There is complete freedom as to which primitive to replace with a "don't care" symbol in each $\mathcal{P}_a$. So, there are $\prod_{a=0}^{a_{\max}} |\mathcal{P}_a|$ different BBBs that can be constructed in this way. Furthermore, in the most general situation, one can apply a different renaming convention for each tree shape and, even, for each node in each shape class, obtaining, in all cases a valid BBB. BBBs built in this more general way offer advantages when writing the evolution equations for selecto-recombinative EAs, in that it allows the "expansion" of the BBB around any particular search space element of interest. In the following, however, for simplicity of exposition we will ignore this aspect of the BBB-construction procedure.

Let us see how we can express more precisely the structure of the characteristic functions in the BBB. For each $v \in \mathcal{H}_d^t$ we have a basis function $\beta_v \in B_\beta(\Omega_d^t)$ recursively defined as follows:

$$\beta_v(x) = \delta(x \overset{r}{\in} v) \times \prod_{i=1}^{\mathbf{ar}(\mathbf{rt}(v))} \beta_{\mathbf{ch}(v,i)}(\mathbf{ch}(x,i))$$

where

$$\begin{aligned} \delta(x \overset{r}{\in} v) &= \delta(v \overset{r}{=} x) \\ &+ \delta(v \overset{r}{\neq} x) \times \delta\left(\mathbf{rt}(v) = *_{\mathbf{ar}(\mathbf{rt}(x))}\right) \end{aligned}$$

which returns 1 if either the root of x and root of v are identical (non-$*$) symbols, or if the root of v is a $*$ symbol and the root of x is a primitive with the same arity as that $*$ symbol. The function $\delta(x \overset{r}{\in} v)$ returns 0 otherwise. So, an alternative way of expressing it is the following

$$\delta(x \overset{r}{\in} v) = \sum_{p \in \mathbf{rt}(v)} \delta\left(\mathbf{rt}(x) = p\right)$$

where, with a minor notational stretch, we treat the $*$ symbols returned by $\mathbf{rt}(v)$ as sets of primitives and the ordinary primitives returned by $\mathbf{rt}(v)$ as singletons (containing such primitives).

Clearly, the functions $\delta(x \overset{r}{\in} v)$ depend only on $\mathbf{rt}(v)$ and $\mathbf{rt}(x)$, not on the actual structure and content of the rest of the trees x and v. So, these are characteristic functions, defined over the primitive set $\mathcal{P}$, i.e., $\delta(x \overset{r}{\in} v) : \mathcal{P} \rightarrow \{0, 1\}$. They are particular in that they return 1 only on a whole $\mathcal{P}_a$ (when $\mathbf{rt}(v) = *_a$) or on just singletons (when $\mathbf{rt}(v)$ is an ordinary primitive of arity a).

By substituting the expression of $\delta(x \overset{r}{\in} v)$ into $\beta_v(x)$ we obtain

$$\begin{aligned} \beta_v(x) &= \sum_{p \in \mathbf{rt}(v)} \delta\left(\mathbf{rt}(x) = p\right) \\ &\times \prod_{i=1}^{\mathbf{ar}(\mathbf{rt}(v))} \beta_{\mathbf{ch}(v,i)}(\mathbf{ch}(x,i)) \end{aligned}$$

which reveals the relationship between the elements of the δ-basis and the characteristic functions in the BBB: $\beta_v(x)$ is the sum of characteristic functions $\delta_w(x)$ of all the $w \in \Omega_d^t$ which "match" (or, more precisely, are contained) in the schema v, i.e.

$$\beta_v(x) = \sum_{w \in v} \delta_w(x)$$

Again, a recursive formulation is important because it shows how effectively the BBB, $B_\beta(\Omega_d^t)$, is constructed by combining tensorially $B_\beta(\Omega_{d-1}^t)$ and the bases $B_\beta(\mathcal{P}_a)$ (for

$a = 0, 1, \cdots$) of characteristic functions over the primitive set. That is

$$B_\beta(\Omega_d) = \bigoplus_{a=0}^{a_{\max}} B_\beta(\mathcal{P}_a) \otimes B_\beta(\Omega_{d-1}^t)^{\otimes a}$$

where the basis $B_\beta(\mathcal{P}_a)$ includes:

- The characteristic functions for the singletons $\{p_{ai}\}$ for all valid elements p_{ai} of $\mathcal{P}_a$, except the element of $\mathcal{P}_a$ that has been replaced by $*_a$. These, of course, are the same for both $B_\beta(\mathcal{P}_a)$ and $B_\delta(\mathcal{P}_a)$.
- The characteristic function for the whole set $\mathcal{P}_a$.

Let us assume, to start with, that the sets $\mathcal{P}_a$ are ordered and that the element of each $\mathcal{P}_a$ we have replaced with $*_a$ is the first element. If we then arrange the elements of $B_\beta(\mathcal{P}_a)$ and $B_\delta(\mathcal{P}_a)$ in two column vectors, $\mathbf{x}_\delta^a$, $\mathbf{x}_\beta^a$ we have,

$$\mathbf{x}_\beta^a = \Lambda_a \mathbf{x}_\delta^a \,, \tag{6}$$

where

$$\Lambda_a \equiv \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

is an invertible $a \times a$ matrix with inverse

$$\Lambda_a^{-1} \equiv \begin{pmatrix} 1 & -1 & -1 & -1 & \cdots & -1 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The structure of Λ_a is simple. It is the identity matrix of size a, in which the 0's in the first row have been turned into 1's. Note that this is exactly the elementary matrix that needs to tensor up to construct the BBB for fixed-length strings from an alphabet with $\mathcal{A}$ symbols [1]. Also, note that, if the element of $\mathcal{P}_a$ replaced with $*_a$ is not the first one but the ith, the structure of Λ_a remains the same, except that now the 0's in the ith row are turned into 1's.

Any δ-basis, including $B_\delta(\mathcal{P}_a)$, is a basis (as it is trivially provable), irrespective of the space it is built on. So, the invertibility of Λ_a guarantees that the set of characteristic functions $B_\beta(\mathcal{P}_a)$ is also a basis for $\mathcal{F}_{\mathcal{P}_a}$.

If the primitive sets $\mathcal{P}_a$ are ordered, we can then extend their order to the space of trees as follows:

- For primitives of the same arity, a, we adopt the order in $\mathcal{P}_a$.
- For primitives of different arity, primitives of lower arity preceed primitives with higher arity, i.e., given $x \in \mathcal{P}_a$ and $y \in \mathcal{P}_b$ with $a \neq b$, $x > y$ if $a > b$.
- For any $x, y \in \Omega_d^t$, if $\mathbf{rt}(x)$ precedes $\mathbf{rt}(y)$ (based on the points above), then x precedes y.
- For any $x, y \in \Omega_d^t$ such that $\mathbf{rt}(x) = \mathbf{rt}(y)$ and $\mathbf{ar}(\mathbf{rt}(x)) = 1$. then x precedes y if $\mathbf{rt}(\mathbf{ch}(x, 1))$ precedes $\mathbf{rt}(\mathbf{ch}(y, 1))$.
- For any $x, y \in \Omega_d^t$ such that $\mathbf{rt}(x) = \mathbf{rt}(y)$ and $\mathbf{ar}(\mathbf{rt}(x)) = 2$, then x precedes y if $\mathbf{rt}(\mathbf{ch}(x, 1))$ precedes $\mathbf{rt}(\mathbf{ch}(y, 1))$ or $\mathbf{rt}(\mathbf{ch}(x, 1)) = \mathbf{rt}(\mathbf{ch}(y, 1))$ and $\mathbf{rt}(\mathbf{ch}(x, 2))$ precedes $\mathbf{rt}(\mathbf{ch}(y, 2))$.
- We proceed similarly to compare trees with roots of arity $a > 2$.

For linear structures this produces the classical odometer ordering.

If we use for $B_\delta(\Omega_d^t)$ the same order as for Ω_d^t, we can now place the elements of $B_\delta(\Omega_d^t)$ in a column vector $\mathbf{x}_\delta^{\Omega_d^t}$. Naturally, we can do exactly the same thing for $\mathcal{H}_d^t$ – the space of schemata in the BBB, and $B_\beta(\Omega_d^t)$, obtaining a column vector $\mathbf{x}_\beta^{\Omega_d^t}$. We want to find the matrix Λ_d^t that performs the basis transformation

$$\mathbf{x}_\beta^{\Omega_d^t} = \Lambda_d^t \mathbf{x}_\delta^{\Omega_d^t}$$

Because of the recursive nature of the search spaces we have

$$\mathbf{x}_\delta^{\Omega_d^t} = \bigoplus_{a=0}^{a_{\max}} \mathbf{x}_\delta^a \otimes \left(\mathbf{x}_\delta^{\Omega_{d-1}^t}\right)^{\otimes a}$$

and

$$\mathbf{x}_\beta^{\Omega_d^t} = \bigoplus_{a=0}^{a_{\max}} \mathbf{x}_\beta^a \otimes \left(\mathbf{x}_\beta^{\Omega_{d-1}^t}\right)^{\otimes a}$$

That is, the vector representations of BBB and δ-basis for a space of a certain dimension are constructed by appropriately combining the vector representations of the same bases but for spaces of lower dimension. From this it follows that

$$\mathbf{x}_\beta^{\Omega_d^t} = \bigoplus_{a=0}^{a_{\max}} \Lambda_a \mathbf{x}_\delta^a \otimes \left(\Lambda_{d-1}^t \mathbf{x}_\delta^{\Omega_{d-1}^t}\right)^{\otimes a}$$

and so the transformation matrix we are looking for is

$$\Lambda_d^t = \bigoplus_{a=0}^{a_{\max}} \Lambda_a \otimes \left(\Lambda_{d-1}^t\right)^{\otimes a}$$

the recursion ending with $\Lambda_0^t = \Lambda_0$. Because of the invertibility of the Λ_a's, and the properties of tensor products, this is invertible, with inverse

$$\left(\Lambda_d^t\right)^{-1} = \bigoplus_{a=0}^{a_{\max}} \Lambda_a^{-1} \otimes \left(\left(\Lambda_{d-1}^t\right)^{-1}\right)^{\otimes a}$$

This tells us that the BBB is, indeed, a basis for $\mathcal{F}_{\Omega_d^t}$ and it shows that the corresponding BBB transformation matrix is

constructed from a handful of small transformation matrices, Λ_a.

Example 5 *The δ- and Building Block bases for binary strings with $\ell = 1$ and $\ell = 3$* For $\ell = 1$, the δ-basis of section IV is $B_\delta(\Omega_1^f) = \{\bar{x}_1, x_1\}$. For binary strings $P_0 = \{0, 1\}$. If we replace the symbol 0 with $*_0$, then the BBB is $B_\beta(\Omega_1^f) = \{e, x_1\}$, where e is a constant function that always returns 1. Arranging the basis elements in columns, $\mathbf{x}_\delta = (\bar{x}_1, x_1)^T$, $\mathbf{x}_\beta = (e, x_1)^T$ we have,

$$\mathbf{x}_\beta = \Lambda_1 \mathbf{x}_\delta \,, \tag{7}$$

where $\Lambda_1 \equiv \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and the subscript 1 refers to the length of the strings.

For $\ell = 3$, if we replace the symbols 0 with $*$'s in $\mathcal{P}_1 = \mathcal{P}_0 = \{0, 1\}$, then the BBB is

$$B_\beta(\Omega_3^f) = \{e, x_3, x_2, x_2x_3, x_1, x_1x_3, x_1x_2, x_1x_2x_3\}.$$

Note that this consists of the characteristic functions of all k-cubes, with $0 \le k \le \ell = 3$, containing the string 111, ranging from the entire 3-cube to the vertex itself. The use of the standard tensor product ordering for the basis elements, mentioned earlier, becomes clear if one substitutes e's for the missing coordinates in each of the above monomials, i.e., writing the basis as $\{e\,e\,e\,, e\,e\,x_3, \, e\,x_2e\,, \ldots\}$. The BBB for $\ell = 3$ can be written as a tensor product

$$\begin{aligned} B_\beta(\Omega_3^f) &= B_\beta(\Omega_1^f) \otimes B_\beta(\Omega_1^f) \otimes B_\beta(\Omega_1^f) \\ &= \{e, \, x_1\} \otimes \{e, \, x_2\} \otimes \{e, \, x_3\} \end{aligned}$$

The matrix Λ_3 that effects the transition between the two bases is the tensor cube of the matrix Λ_1 defined above, i.e., $\Lambda_3 = \Lambda_1^{\otimes 3} \equiv \Lambda_1 \otimes \Lambda_1 \otimes \Lambda_1$. □

In the more general case of Ω_ℓ^f, we can see that Λ_ℓ must satisfy the recursion relation

$$\Lambda_\ell = \Lambda_1 \otimes \Lambda_{\ell-1} = \begin{pmatrix} \Lambda_{\ell-1} & \Lambda_{\ell-1} \\ 0 & \Lambda_{\ell-1} \end{pmatrix} = \Lambda_1^{\otimes \ell}. \tag{8}$$

The matrix elements of the BBB transformation Λ_ℓ, $(\Lambda_\ell)_I{}^J$, are such that $(\Lambda_\ell)_I{}^J = 1$ if the vertex J is contained in the k-cube I, and is zero otherwise, i.e. the string I is a member of the schema J.

Example 6 *Building Block basis for $\Omega^v_{\ell_{\max}}$, with $\ell_{\max} = 2$ and $\mathcal{A} = 3$* We choose arbitrarily the allele 0 as the allele to be replaced by $*$, then the schemata used in the BBB, in the odometer ordering, are

$$\{*, 1, 2, **, *1, *2, 1*, 11, 12, 2*, 21, 22\}\,, \tag{9}$$

where $*$ represents the three strings 0, 1 and 2 and $**$ the nine strings 00, 01, 02, $\cdots$, 22. Similarly, $*1$ represents the three strings 01, 11 and 21. Geometrically the schemata in the BBB are hyperplanes. The BBB is

$$\begin{aligned} B_\beta &= \{\delta_*, \delta_1, \delta_2, \delta_{**}, \, \delta_{*1}, \ldots, \, \delta_{22}\} \\ &= \{\delta^1_*, \delta^1_1, \delta^1_2, \delta^1_*\delta^2_*, \, \delta^1_*\delta^2_1, \ldots, \, \delta^1_2\delta^2_2\} \\ &= \{1, -(x_1^{(1)} - 2), (x_1^{(1)} - 1), \, 1, \\ &\quad -(x_1^{(2)} - 1)(x_1^{(2)} - 2)(x_2^{(2)} - 2)/2, \ldots, \\ &\quad (x_1^{(2)} - 1)(x_2^{(2)} - 1)\}\,. \end{aligned}$$

The coordinate transformation matrix that transforms between the δ- and Building Block bases is

$$\Lambda_2^v = \begin{pmatrix} \Lambda_1^f & \mathbf{0}_r \\ \mathbf{0}_c & \Lambda_2^f \end{pmatrix} \tag{10}$$

where $\mathbf{0}_r$ is a 3×9 matrix with all zero entries, $\mathbf{0}_c$ is a 9×3 matrix with all zero entries and

$$\Lambda_1^f = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

and

$$\begin{aligned} \Lambda_2^f &= \Lambda_1^f \otimes \Lambda_1^f \\ &= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{12}$$

□

Example 7 *Building Block basis for trees with $\mathcal{P} = \{x, y, +, -, \times\}$ and maximum depth $d = 1$ (two levels)* With these choices we have the following program space

$$\Omega_1^t = \Big\{ \mathrm{x}\,,\ \mathrm{y}\,,\ \ \begin{array}{ccc} & + & \\ \mathrm{x} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & + & \\ \mathrm{x} & & \mathrm{y} \end{array}\ ,$$

$$\begin{array}{ccc} & + & \\ \mathrm{y} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & + & \\ \mathrm{y} & & \mathrm{y} \end{array}\ ,\ \ \begin{array}{ccc} & - & \\ \mathrm{x} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & - & \\ \mathrm{x} & & \mathrm{y} \end{array}\ ,$$

$$\begin{array}{ccc} & - & \\ \mathrm{y} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & - & \\ \mathrm{y} & & \mathrm{y} \end{array}\ ,\ \ \begin{array}{ccc} & \times & \\ \mathrm{x} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & \times & \\ \mathrm{x} & & \mathrm{y} \end{array}\ ,$$

$$\begin{array}{ccc} & \times & \\ \mathrm{y} & & \mathrm{x} \end{array}\ ,\ \ \begin{array}{ccc} & \times & \\ \mathrm{y} & & \mathrm{y} \end{array} \Big\}$$

If we replace the primitives x and $+$ with "don't care" symbols $*$ of appropriate arity, we obtain the schemata forming the BBB $B_\beta(\Omega_1^t)$, namely

$$\mathcal{H}_1^t = \left\{ *,\ \mathrm{y},\ \begin{array}{c}*\\ *\ \ *\end{array},\ \begin{array}{c}*\\ *\ \ \mathrm{y}\end{array},\ \begin{array}{c}*\\ \mathrm{y}\ \ *\end{array},\ \begin{array}{c}*\\ \mathrm{y}\ \ \mathrm{y}\end{array},\ \begin{array}{c}-\\ *\ \ *\end{array},\ \begin{array}{c}-\\ *\ \ \mathrm{y}\end{array},\ \begin{array}{c}-\\ \mathrm{y}\ \ *\end{array},\ \begin{array}{c}-\\ \mathrm{y}\ \ \mathrm{y}\end{array},\ \begin{array}{c}\times\\ *\ \ *\end{array},\ \begin{array}{c}\times\\ *\ \ \mathrm{y}\end{array},\ \begin{array}{c}\times\\ \mathrm{y}\ \ *\end{array},\ \begin{array}{c}\times\\ \mathrm{y}\ \ \mathrm{y}\end{array} \right\}.$$

Thanks to the recursive and tensorial structure of the search space we can write

$$B_\beta(\Omega_1) = B_\beta(\mathcal{P}_0) \oplus B_\beta(\mathcal{P}_2) \otimes B_\beta(\Omega_0^t)^{\otimes 2}$$

where $B_\beta(\Omega_0^t) = B_\beta(\mathcal{P}_0)$, and so

$$B_\beta(\Omega_1) = B_\beta(\mathcal{P}_0) \oplus B_\beta(\mathcal{P}_2) \otimes B_\beta(\mathcal{P}_0) \otimes B_\beta(\mathcal{P}_0)$$

where

$$B_\beta(\mathcal{P}_0) = \left\{ \sum_{p\in\{x,y\}} \delta(x=p), \delta(x=\text{“y}'') \right\}$$

and

$$B_\beta(\mathcal{P}_2) = \left\{ \sum_{p\in\{+,-,\times\}} \delta(x=p), \delta(x=-), \delta(x=\times) \right\}$$

Note, that the elements of these two sets are univariate functions, hence the argument x.
If $x_1^{(0)}$ represents the primitive in the (only) node in trees of depth 0, $x_1^{(1)}$ represents the root node of trees of depth 1, $x_2^{(1)}$ represents the first (left-most) child of the root node, and $x_3^{(1)}$ represents the second child, then we can write:

$$\begin{aligned} & B_\beta(\Omega_1) \\ = & \left\{ \sum_{p\in\{x,y\}} \delta(x_1^{(0)}=p), \delta(x_1^{(0)}=\text{“y}'') \right\} \\ \oplus & \left\{ \sum_{p\in\{+,-,\times\}} \delta(x_1^{(1)}=p), \delta(x_1^{(1)}=-), \delta(x_1^{(1)}=\times) \right\} \\ \otimes & \left\{ \sum_{p\in\{x,y\}} \delta(x_2^{(1)}=p), \delta(x_2^{(1)}=\text{“y}'') \right\} \\ \otimes & \left\{ \sum_{p\in\{x,y\}} \delta(x_3^{(1)}=p), \delta(x_3^{(1)}=\text{“y}'') \right\}. \end{aligned}$$

As there are two of them, the transformation matrix to go from the δ-basis to the BBB for the primitives of arity 0 is

$$\Lambda_0 \equiv \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

while for the three primitives of arity 2 it is

$$\Lambda_2 \equiv \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

So, the coordinate transformation matrix for the trees of up to depth 1 considered in this example is

$$\begin{aligned} \Lambda_1^t &= \Lambda_0 \oplus \Lambda_2 \otimes \Lambda_0 \otimes \Lambda_0 \\ &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ &\oplus \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ &\oplus \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\oplus \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&0&0&1&0&0&0&1&0&0&0&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&1&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

$$= \begin{pmatrix} 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&0&0&1&0&1&0&1&0&1&0&1&0&1&1 \\ 0&0&0&0&1&1&0&0&1&1&0&0&1&1&1 \\ 0&0&0&0&0&1&0&0&0&1&0&0&0&1&1 \\ 0&0&0&0&0&0&1&1&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

□

VI. Conclusions

In this paper, inspired by the recombinative GA case for fixed-length strings, we showed that there are natural generalisations of the BBB to the case of variable-length strings, and to the case of trees. Furthermore, we developed a mathematical framework within which these basis transformations may be written. The search or configuration spaces of strings and trees have both a natural recursive and modular structure. The modularity manifests itself mathematically in terms of a tensor sum structure of Ω, composed of naturally indexed subspaces. The modularity is indexed by string length in the case of variable-length strings and by tree depth or tree shape in the case of trees. Recursivity, meanwhile, is manifest in the fact that the search space for a higher dimension subspace can be generated from an associated lower order one. For instance, it was known previously that for fixed-length strings, the basis for a given ℓ could be generated by taking a tensor product of the bases for $\ell = 1$. As Ω, generally, has a modular-recursive structure, manifest in the presence of tensor sums and products, we showed that knowledge of Ω and any basis thereon could be generated from the search spaces and corresponding bases for single bits or primitives of a given arity. Further, coordinate transformations between the different bases can be effected by considering the transformation matrices on the subspaces associated with single bits or primitives of a given arity. It is a great simplification that all the relevant coordinate transformations can be achieved using a few simple underlying matrices which, in some sense, act as "building blocks" themselves for constructing the seemingly complicated bases and transformations that exist on the search spaces.

Sometimes the mathematics of EAs, particularly trees, can seem overwhelmingly complicated. This is chiefly due to the difficulties of seeing the underlying mathematical structure. Here, we have shown that the language of tensor sums and tensor products is a very elegant way in which the structure inherent in GAs and GP can emerge. It is known that the BBB leads to great simplification of the dynamics of recombinative GAs with homologous recombination, leading to a formulation where the creation of a string or schema via a particular crossover mask is associated uniquely with a *single* pair of conjugate schemata - Building Blocks. This is distinct to the case of strings, where, for a given mask, there are potentially many string combinations that may give rise to a particular target string. Interestingly, the dynamical equations written in the Building Block basis are identical to those previously found using coarse-graining techniques. Having found and analysed analogs of the BBB for variable-length strings and trees, one may ask if these generalisations of the BBB also lead to simplifications of the dynamics of homologous recombination for variable-length strings and trees. The answer is yes and bears the key characteristic that for a given analog of a crossover mask there is one and only one conjugate pair of schemata that give rise to an offspring string or tree. This result will be examined in more detail in another publication.

Although we have here concentrated on the δ- and Building Block bases, it is clear that the formalism we have developed is applicable to *any* coordinate transformation. It will be interesting to see what other interesting bases exist that simplify the dynamics of EAs or facilitate the analysis of fitness functions. Two natural candidates, for example, are the generalisations of the Fourier and Taylor bases from the case of fixed-length strings.

Acknowledgements

CRS and RP thank the EPSRC for financial support (grant number GR/T24616/01). CRS also thanks DGAPA of the UNAM for a Sabbatical Fellowship and Conacyt project 30422-E. The research has also been partially supported by the Leverhulme Trust through a visiting professorship to CRS.

References

[1] C. Chryssomalakos and C. R. Stephens. Covariant genetic dynamics. Submitted.

[2] C. Chryssomalakos and C. R. Stephens. What basis for genetic dynamics? In Kalyanmoy Deb, editor, *Proceedings of GECCO 2004*, pages 1018–1029, Berlin, Germany, 2004. Springer Verlag.

[3] D. E. Goldberg. Genetic algorithms and walsh fuctions: II. Deception and its analysis. *Complex Systems*, 3(2):153–171, April 1989.

[4] David E. Goldberg. Genetic algorithms and Walsh functions: Part I, A gentle introduction. *Complex Systems*, 3(2):129–152, 1989.

[5] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[6] Nicholas Freitag McPhee and Ellery Fussell Crane. A theoretical analysis of the hiff problem. In *GECCO*, pages 1153–1160, 2005.

[7] Riccardo Poli. Exact schema theorem and effective fitness for GP with one-point crossover. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 469–476, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.

[8] Riccardo Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163, June 2001.

[9] Riccardo Poli. General schema theory for genetic programming with subtree-swapping crossover. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 143–159, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[10] Riccardo Poli and Nicholas Freitag McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1):53–66, 2003.

[11] Riccardo Poli and Nicholas Freitag McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2), 2003.

[12] Riccardo Poli and Christopher R. Stephens. Theoretical analysis of generalised recombination. In *CEC-2005*, 2005. Accepted.

[13] C. R. Stephens. Some exact results from a coarse grained formulation of genetic dynamics. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 631–638, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[14] C. R. Stephens and H. Waelbroeck. Effective degrees of freedom in genetic algorithms and the block hypothesis. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 34–40, East Lansing, 1997. Morgan Kaufmann.

[15] C. R. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124, 1999.

[16] Christopher R. Stephens. The renormalization group and the dynamics of genetic systems. *Acta Phys. Slov.*, 52:515–524, 2003.

[17] Christopher R. Stephens and Riccardo Poli. EC theory – "in theory": Towards a unification of evolutionary computation theory. In Anil Menon, editor, *Frontiers of Evolutionary Computation*, pages 129–156. Kluwer, Boston, MA, 2004.

[18] Christopher R. Stephens and Riccardo Poli. Coarse graining in an evolutionary algorithm with recombination, duplication and inversion. In *CEC-2005*, 2005. Accepted.

[19] Christopher R. Stephens, Riccardo Poli, Alden H. Wright, and Jonathan E. Rowe. Exact results from A coarse grained formulation of the dynamics of variable-length genetic algorithms. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 578–585, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[20] Christopher R. Stephens and H. Waelbroeck. Effective degrees of freedom in genetic algorithms. *Phys. Rev.*, 57:3251–3264, 1998.

[21] Marc Toussaint. Notes on information geometry and evolutionary processes. *CoRR*, nlin.AO/0408040, 2004.

[22] Michael D. Vose and Alden H. Wright. The simple genetic algorithm and the Walsh transform: Part I, theory. *Evolutionary Computation*, 6(3):253–273, 1998.

[23] Michael D. Vose and Alden H. Wright. The simple genetic algorithm and the Walsh transform: Part II, the inverse. *Evolutionary Computation*, 6(3):275–289, 1998.

[24] E. D. Weinberger. Fourier and taylor series on fitness landscapes. *Biol. Cybern.*, 65:321–330, 1991.

Riccardo Poli is a professor in the Department of Computer Science at Essex. His main research interests include Genetic Programming (GP) and the theory of evolutionary algorithms. He has published about 170 refereed papers on evolutionary algorithms, particle swarms, neural networks and image/signal processing. He has co-authored the book Foundations of Genetic Programming (Langdon and Poli, 2002). He has been co-chair of EuroGP, the European Conference on Genetic Programming for 1998, 1999, 2000 and 2003, deme chair for the Genetic and Evolutionary Computation Conference (GECCO) 2002, co-chair of the 2002 Foundations of Genetic Algorithms (FOGA) Workshop, general chair of GECCO in 2004. Prof Poli is an associate editor of Evolutionary Computation, of Genetic Programming and Evolvable Machines and of the International Journal of Computational Intelligence Research. He has given invited tutorials and talks at numerous international conferences, workshops and Summer schools. He is a member of the Engineering and Physical Sciences Research Council (EPSRC) Peer Review College, an expert reviewer for the EU, for Science Foundation Ireland, the Swiss government and the Italian Ministry for Scientific Research. Prof Poli has attracted, as principal investigator or co-investigator, funding for over £1.8M from EPSRC, the Defence and Evaluation Research Agency, Leverhulme Trust, Royal Society, and others. In July 2003 Prof. Poli was elected to be a Fellow of International Society for Genetic and Evolutionary Computation (ISGEC), which as now become the ACM special interest group SIGEVO.

Chris Stephens is Professor at the Institute for Nuclear Sciences of the Universidad Nacional Autonoma de Mexico. After receiving his undergraduate degree at The Queen's College, Oxford he completed his graduate work at the University of Maryland in the area of theoretical physics. He then had several postdoctoral positions, including the University of Utrecht, where he worked with Gerard 't Hooft, the 1999 Nobel Laureate in Physics, and a Marie Curie Fellowship at the Dublin Institute for Advanced Studies. He has had visiting positions at various leading academic institutions, including the Weizmann Institute, the Joint Institute for Nuclear Research, Dubna, the Universities of Birmingham and Essex and others. He is a founding partner of Adaptive Technologies Inc. a research company dedicated to the production of agent-based technologies for dynamical optimization in finance and industry. He is author or co-author of over 100 publications and his work has been cited over 1000 times. He has given over 120 invited lectures in more than 20 countries. Among the academic honours he has received are the Jorge Lomnitz Prize of the Mexican Academy of Sciences and a Leverhulme Professorship from the Leverhulme Trust. He is also a member of the editorial board of Genetic Programming and Evolvable Hardware.
His research interests are very broad, having published in a wide array of international journals - ranging from Classical and Quantum Gravity to the Journal of Molecular Evolution. An overiding theme, however, has been the Renormalization Group - a general methodology for solving complex, non-linear problems with many degrees of freedom via coarse graining - and, more recently, applying it to the area of genetic dynamics. His principal contribution in Evolutionary Computation has been to show how exact coarse-grained formulations lead to a unification and reconciliation of many previously antagonistic theoretical elements, such as Holland's Schema theorem and the Vose model.