Phenotypic Species Definitions for Genetic Improvement of Source Code

Zsolt Németh[1], Penn Faulkner Rainford[2], and Barry Porter[1]

[1]School of Computing and Communications, Lancaster University, UK
[2]Department of Computer Science, University of York, UK
z.nemeth@lancaster.ac.uk

Abstract

Emergent software systems are composed of elementary building blocks, where many of those blocks have variations available which are better or worse in different deployment contexts. Genetic Improvement (GI) for source code has been proposed for creating and curating collections of such blocks, but the combination of new code synthesis with genetic mutation and crossover results in large, complex search spaces. A range of methods to aid such a search have been proposed, with the particular notion of *species* having appeared in the context of Genetic Algorithms (GAs) to identify individuals with similar genotypes for controlling competition, encouraging the exploration of distant local optima, maintaining diversity and avoiding premature convergence. In this paper we examine a species definition for GI for source code, a domain which has specific features: genotype similarity is largely irrelevant; distance between individuals is undefined; and the fitness landscape is extremely rugged. We propose a **phenotypic** species definition that captures an algorithm's **functional** phenotypic characteristics, while excluding its non-functional phenotypic characteristics (and its particular representation in source code). We introduce our proposal in a GI for a hash table scenario, where species are characterised by divergence in probability distributions.

Introduction

Genetic code improvement, inspired by genetic algorithms, is the process of taking an existing piece of program code and finding a better version through genetic search. The definition of 'better' is context specific (such as performance (calls-per-second), or energy consumption), and has been approached in various ways from modifying machine code directly to operating on abstract syntax trees of source code (Petke et al., 2018). We focus on an abstract syntax tree approach, and apply GI to emergent software systems (Filho and Porter, 2017). Emergent software is constructed from a large pool of small building blocks, where many of those blocks have implementation variants available (such as different sorting or scheduling algorithms). These systems continuously learn which composition of building blocks best suits each set of observed deployment conditions, and can integrate new building block variants at any time. Genetic improvement in this domain can be used to automatically search for better building block variants for known deployment conditions, and for new high-performance variants when novel deployment conditions are encountered. Using GI in this context can yield significant improvements (McGowan et al., 2018), but because it is necessary to mix new code synthesis with traditional mutation and crossover, this represents a very large search space problem with large areas of neutral drift (Miikkulainen and Forrest, 2021).

Existing research has attempted to overcome this through methods to improve search space coverage. A popular and effective approach, in systems with large amounts of source material for crossover, has been novelty search (López-López et al., 2018; Villanueva et al., 2020), however novelty search is still limited in its usefulness for very large search spaces (Cuccu and Gomez, 2011). The other method often used is to limit the search space size for GI by operating only in terms of large but known-to-be-functional code blocks (Brownlee et al., 2019; Petke et al., 2014). This reduces the chance of a breaking change and smooths the fitness landscape, but also limits the potential for creation of new functionality – while this may work well for code repair applications, it is likely to be less effective for optimisation. A final approach is guided evolution, similar to that used for agriculture in biological systems. This uses analysis of past GI runs to predict good starting points for optimisation of subsequent runs (Rainford and Porter, 2022).

We present a new approach to this domain with a novel conception of *species* within a GI process for source code. Our species concept is application-domain-specific for a given building block, and provides a way to cheaply model the diversity of a population in real-time. Through 894 experiments we observe clear speciation, and also observe clear correlations between runs that had more species fluctuation over time and those that finished with a significantly better-performing individual: new species cause improvements of 15% in performance, compared to improvements of 3% within existing species. We also analyse the kinds of transformations (mutations and/or crossovers) which are more likely to lead to new species; in future work we may be able to apply these observations to create a higher ratio of runs with higher species turnover and better final outcomes.

Related Work

The seminal paper by (Goldberg and Richardson, 1987) introduced the notion of species formation in GA as *niching by fitness sharing*. The aim was to subdivide the population and thereby control competition between distant points in the search space. This increased diversity in the population and reduced premature convergence as a side effect. The method requires a distance definition (either in the genotype or phenotype space) d, a sharing function $\mathcal{S}$, with fitness values of each individual adjusted by $\mathcal{S}(d)$ so that individuals close to each other share their fitness values (and the proportion of sharing decreases by distance). Apart from the fitness sharing mechanism, species are not defined explicitly.

(Della Cioppa et al., 2007) later advanced an explicit, dynamic identification of species which builds on the above work. This is a formal species definition based on properties of the individuals, rather than the less formal usage of "species" as a synonym for sub-populations in a typically cooperative co-evolutionary scenario where there is no genetic interaction between species (John et al., 2022; Gao et al., 2022; Rui L. Lopes and Almada-Lobo, 2020). Work following this species definition in GAs has focused on the *genotypic* definition whereby the inter-species distance is based on the genomes which are evolved, rather than the phenotypic properties of those individuals in context.

(Goldberg and Wang, 1997) proposed an adaptive niching method following their initial concept described above. Here they employ two sub-populations, customers and businesses. Customers are associated with a business according to some distance definition and businesses act as seeds of a species. (Li et al., 2002) take this further, with evolving parallel sub-populations where individuals are vectors of real numbers and species are defined by Euclidean distance and a species distance constant, σ_s. The Euclidean distance is meant to characterise the similarity of individuals, whereas σ_s is the delimiter to separate species. Species in this model are therefore clusters of individuals around a species seed (the best-fitness individual in the group). This approach supports conservation of species to offer a balance between the otherwise contradicting elitism (conserve the fittest) and diversity (conserve the different). There are many other variants of Euclidean-based species definitions in different GAs (e.g., (Dong et al., 2011; Jelasity and Dombi, 1998; Raghuwanshi and Kakde, 2006; Wong et al., 2009)); as far as we are aware, however, there is no work to date on species definitions for genetic improvement of source code.

While genotypic species definition has dominated GA research, phenotypic definitions have been explored in artificial organisms (Dolson et al., 2020), including analysis on the link between genotype lineages and phenotypic complexity (Lenski et al., 2003). While artificial organisms are built from more limited instruction sets than general-purpose programming languages, they work in a similar way, suggesting phenotypic species descriptions may apply to GI.

Methodology

GI in emergent software systems

Our genetic improvement method is aimed at emergent software systems, a kind of software which is constructed from a pool of small component building blocks, many of which have implementation variation available (such as alternative sorting, cache replacement, or scheduling algorithms). Emergent software systems continuously observe their deployment environment characteristics and performance at runtime, and continuously learn which composition(s) of building block variants best suit each set of deployment characteristics encountered (Filho and Porter, 2017). The pool of available building blocks can be added to over time, allowing better implementation variants (or better sub-architectures) to be introduced or trialled. We deploy genetic improvement for these kinds of systems to automate the process of deriving new high-utility building block variants – variants which can otherwise be challenging and time-consuming for human engineers to design and build.

We assume the workflow proposed by (Rainford and Porter, 2022), in which a particular building block is identified for potential improvement in a live system, and a function-call monitoring probe is injected at the interface to that block. The probe captures function call parameter values and return values, representing a trace of a sequence of specific calls made on that building block over a period of time. This sequence of calls represents the localised fingerprint of the overall software system's current deployment environment conditions. This call sequence (with the same parameter values) is replayed offline during a GI process, to attempt to find a better candidate individual which processes that sequence of calls more quickly than the current best implementation variant. During the GI process, the original return values captured in the call sequence are used to assert functional correctness of the improved individual.

As observed by (McGowan et al., 2018), this application of genetic improvement requires a mixture of novel source code synthesis combined with traditional mutation and crossover operations. This yields a particularly large and rugged search space, prone to large areas of neutral drift (Miikkulainen and Forrest, 2021).

Phenotypic speciation

There are over 30 alternative concepts of speciation in biology (Zachos, 2016), each with subtle differences. Consequently, species is an often-overloaded term in computational biology, artificial life, or in our case, genetic improvement. Our work targets diversity arising from evolutionary adaptation to specific environmental conditions; our conception of species therefore somewhat resembles the notion of ecospecies – a lineage which occupies an adaptive zone minimally different to that of any other lineage (Wilkins, 2009).

To demonstrate the differences in genotypes and phenotypes specifically in the context of GI for source code, let us

consider the following examples. The code fragments:

```
h = ((h << 5) + h) + key[c]      (1)
```

and:

```
h = ((h << 5) - h) + key[c]      (2)
```

are, by static analysis, very close to each other in their genotypes (program text). Yet, their functional phenotypes are different: one is the core of the Bernstein hash; the other is the Kernighan-Ritchie hash. Let us then consider:

```
h = key[c] + 33 * h              (3)
```

which resembles neither genotypes of the former fragments, and would take a (highly unlikely) long chain of mutations to reach. However, this example is the Bernstein hash function written in a different form. Are then fragments 1 and 3 identical in their phenotypes? They are identical in the *functional* attributes of their phenotypes but they may have different *non-functional* attributes such as speed or memory use; they are variations of the same algorithm.

The genotypic definition of species assumes a strong and smooth association between genotype and phenotype, that individuals with very similar descriptions will have very similar fitness. This assumption breaks easily when working with source code, where even the smallest alterations – that is otherwise a very small distance of individuals in terms of genotype – may result dramatic changes in phenotype such that it no longer compiles. This makes the genotypic description of species for GI a poor choice according to (van Laar, 2021; Novozhilov et al., 2007).

We propose the use of *phenotypic speciation* to grasp the subtle but intrinsic differences in a population, such as those between the Bernstein and Kernighan-Ritchie algorithms above, but also to detect the identity of the two Bernstein function versions. In contrast to the body of work on genotype-oriented speciation, presented in the related work, our phenotypic method is aimed at capturing *how* an implementation behaves rather than the fine details of how that implementation is encoded. To gain our phenotypic species definition we divide the phenotype of an individual into its **functional** (the algorithm it represents) and **non-functional** (how the algorithm is represented) attributes. The challenge is then how to classify individuals into species by **functional** phenotype so that the definition is selective (sensitive to the smallest algorithmic changes), characteristic (a fingerprint-like identification of individuals of a species), and insensitive to implementation details while also being easily computable. The particular definition of a phenotypic species varies and depends on an algorithm's domain; in this paper we present a definition for classifying *mapping functions* into species with *probability divergence metrics*.

By capturing *functional* phenotype in the right way we can give less credence to neutral regions of search space, where fitness variation is insignificant and may be composed of individuals that are different implementations of the same algorithm. Instead we may drive the search towards areas of significant variance, an objective broadly similar to novelty search (Doncieux et al., 2019). Our notion of species will capture the meaningful *diversity* of a population.

Motivating example

We use a hash function as our specific example for experimentation in this paper, which aligns our research with other existing work in GI by (Rainford and Porter, 2022). A hash function takes a text string and derives an integer from that string which lies in the range $\{0 \,..\, H_{max}\}$. The derivation is deterministic, always yielding the same integer when presented with the same string. Hash functions are used in hash table software components, which have `put()` and `gett()` functions to support rapid retrieval of data. The `put()` function is given a text string as a *key* and an arbitrary piece of data as a *value*, and uses a hash function to determine which bucket to place that value into. The `get()` function is provided only with the text string key, computes the correct bucket for that key, then searches all of the keys which were mapped to that bucket to find the matching one, returning the associated data originally given via `put()`.

More formally a hash function can be defined as $h : \mathcal{S} \mapsto \mathcal{X}$, where $\mathcal{X} \subset \mathbb{N}$ and $\mathcal{S}$ is a set of strings, albeit calculated deterministically, its output can be considered as a random variable. The probability $P(h(s) = x)$, $s \in \mathcal{S}$, is approximated by the relative frequencies of strings put into bucket x as $P(x) = \frac{\text{strings in bucket } x}{\text{all strings stored in the hash table}}$.

The divergence defined by Kullback and Leibler (1951) is a statistical measure of how one probability distribution differs from another. For discrete distributions it is:

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{4}$$

where P and Q are discrete distributions over $\mathcal{X}$. D_{KL} is non-negative, non-symmetrical $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$ and $D_{KL}(P \parallel Q) = 0$ if and only if distributions P and Q are identical.

Given a hash table of n buckets and a hash function $h : \mathcal{S} \mapsto \{1 \ldots n\}$, Q is a uniform distribution over $\{1 \ldots n\}$, $Q(x) = \frac{1}{n}$, then the Kullback-Leibler divergence of $P(h(s) = x)$ from $Q(x)$ is

$$D_{KL}(P \parallel Q) = \log n + \sum_{x \in \{1 \ldots n\}} P(x) \log P(x) \tag{5}$$

Note, that if Q is the uniform distribution then $D_{KL}(P \parallel Q)$ is the same for all permutations of $x \in \mathcal{X}$ hence, if P and R are discrete distributions over $\mathcal{X}$ and $D_{KL}(P \parallel Q) = D_{KL}(R \parallel Q)$ then it does not imply that P and R are identical. The behaviour of a hash table, however, is insensitive to such permutations: all permutations of the same distribution would exhibit the same performance.

We *define* species by one particular functional attribute of the phenotype: the calculated Kullback-Leibler divergence. Two individuals are of the same species if their D_{KL} values are identical, irrespective of their genotype and other attributes of the phenotype. Individuals of the same species

can have very different genotypes and other attributes of their phenotypes. They are all better or worse realisations of the same algorithm, that while determines the adaptive zone the given lineage may occupy, does not overly restrict the potential set of its attributes.

Speciation in practice

Our GI program starts off with an initial population composed entirely of individuals carrying the exact same code (this is our candidate software building block to attempt to improve). The evolution repeats the (i) crossover, (ii) mutation, (iii) fitness evaluation, and (iv) selection cycle, then the new generation of selected individuals enter the next iteration. Fitness is associated with typically non-functional attributes of the phenotype, hence the evaluation of the particular functional attribute and the classifying of species also take place in the (iii) fitness evaluation step.

We measure fitness purely as execution speed of an individual; this is tested by inserting and retrieving 1,000 elements of a certain training dataset into an empty hash table of 100 buckets. Following the speed test, the final data distribution in the hash table is established. We gain this information from the automated insertion of a probe function to the hash table with returns a measure of its data layout. We note there is no probe effect associated with this function call, it is a purely post-mortem examination and does not interfere with the functionality or speed of the hash function.

The Kullback-Leibler divergence, which we use as our species metric, is calculated from the received data, with the resulting D_{KL} recorded as an attribute of the phenotype. Both the data collection and the calculation of D_{KL} are of linear complexity and cause negligible delay in testing an individual. Records of each individuals, including all attributes of their phenotype and the species fingerprint, are logged for offline analysis. At runtime the selection phase ranks individuals by certain phenotype attributes (*excluding* those attributes used to determine species).

To help demonstrate why our phenotypic species metric for this application is useful, Figure 1 depicts a hypothetical hash table of 10 buckets where 40 elements are placed by some hash function. Figures 1a and 1b are permutations of the same distributions, while 1c is significantly different and Fig. 1d merely differs from Fig. 1b by a single element. If $P_i(h(s) = i)$ is estimated by the relative frequencies then P_i values for, e.g., Fig. 1a are $\{\frac{5}{40}, \frac{6}{40}, \frac{2}{40}, \dots \frac{4}{40}\}$, $n = 10$, the Kullback-Leibler divergence values D_{KL}, as defined in Eq. 5, are: 0.128, 0.128, 0.018, 0.157 for data layouts in Fig. 1a-1d, respectively. As it can be seen, permutations of the same distribution have the same divergence values otherwise, the displacement of even a single element is detected by D_{KL}. If each bucket had the same number of elements, 4, then $D_{KL} = \log 10 + 10 * \frac{4}{40} * \log \frac{4}{40} = \log 10 - \log 10 = 0$, the uniform distribution is correctly identified.

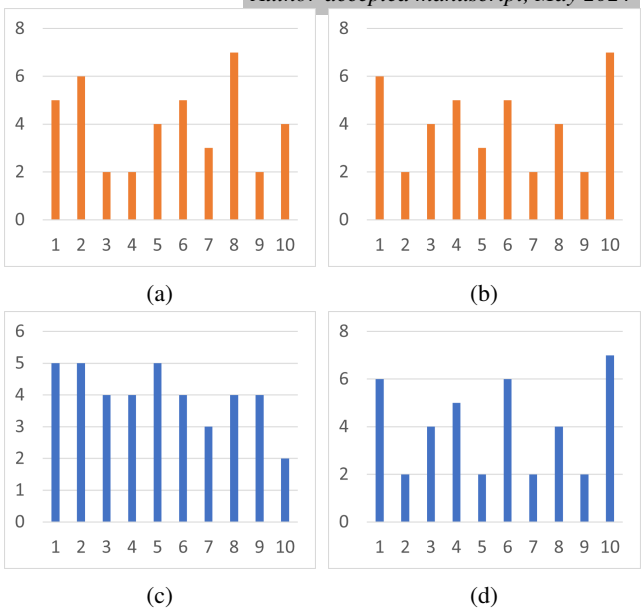

Figure 1: Examples of data distributions in a hash table

Generality

Finally, we note the species definition used in this paper is clearly application-specific; nonetheless it can be adapted to a larger class of optimisation problems. Hash functions make a *mapping* between a set of strings (or other objects) $\mathcal{S}$ and a set of natural numbers $\mathcal{X} = \{1, 2, \dots n\} \in \mathbb{N}$ where $|\mathcal{S}| \gg |\mathcal{X}|$ (such that that multiple elements of $\mathcal{S}$ may map onto an element of $\mathcal{X}$). The actual elements of $\mathcal{S}$ mapped to $i \in \mathcal{X}$ depend on the mapping function and can be treated as a probability variable; the distance of its probability distribution from a reference distribution Q is characteristic to the mapping function. Our species classification method can be directly applied to a broader scope of mapping-related problems, such as schedulers and cache eviction policies, and in future we aim to extend phenotype-based speciation to other classes of algorithms beyond mapping functions.

Evaluation

In our evaluation we aim to show the effect of phenotypic species classification using the hash table example. We classify species at each generation and are then able to track the appearance, lifetime, and extinction of species during the GI run. We can also examine correlations between fitness changes and species events, and the mutation and crossover effects which tend to lead to species creation and extinction.

In this section we firstly examine species effects at a micro level, in terms of individual GI runs, and secondly examine species effects at a macro level to reveal the effects of speciation volume and frequency. Our GI framework is written in the Dana adaptive systems language (**?**), and is made available for replication of our results [1].

[1] https://doi.org/10.5281/zenodo.11477826

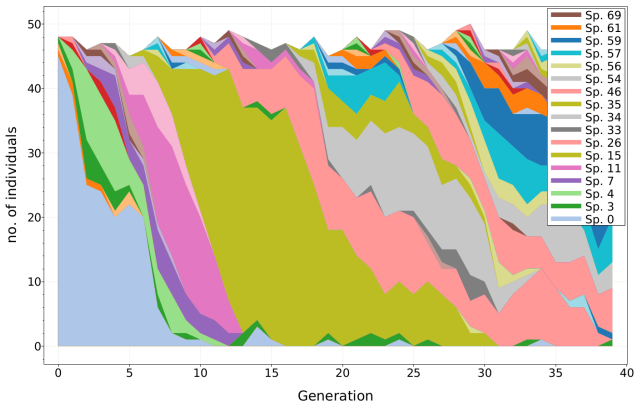

Figure 2: Abundance of species in the first 40 generations.

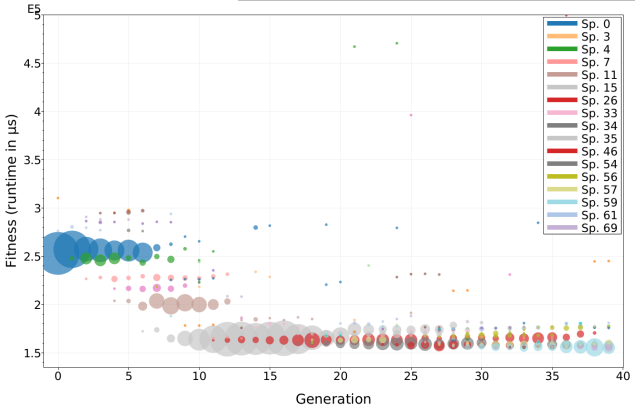

Figure 3: Best fitness and abundance of species.

The analysis for both evaluation elements is based on data from 894 independent experiments in which we recorded the creation, lifespan, and decline of species and their effect on the evolutionary process. The experiments evolve a hash function against a training set of 1,000 words from an Economist article, i.e., storing and retrieving these words in a hash table of 100 buckets as fast as possible. Our population size is 50 individuals and we continue the GI process for 200 generations. The mutation probability is 0.8, crossover probability is 0.2; mutation subtypes *Insert before*, *Insert after*, *Modify* and *Delete* are weighted as 0.15, 0.15, 0.3, 0.3, respectively; the weight function at selection is reciprocal and the elite set is 2 individuals. The starting generation consists of a homogeneous set of individuals for a multiplicative hash function, and our fitness function is simply execution time in μs, where *smaller* values represent better fitness.

Species in a single evolutionary process

While each individual experiment naturally has a somewhat different outcome, it is useful to study single example experiments at a micro level to show speciation effects in detail. Figure 2 shows the progression of species for one such single GI run, focusing on the first 40 generations. Each colour represents a different species, with the y-axis showing the number of individuals identified as a member of that species within a given generation (generation number shown on the x-axis). Novel species are always plotted at the top of the volume, above all existing species. This graph shows how our phenotypic species definition captures the introduction, development, and extinction of species over time. Novel species begin as a small fraction of the population, then if successful can quickly dominate that population, before further new species arrive. Some species are highly successful, filling the volume, while others appear only briefly at the top of the volume before disappearing. While not all runs have such diversity of species (we cover macro-level statistics in the following section), this behaviour suggests that we have a useful way to analyse the characteristics of a GI run that is both computationally simple and problem-specific.

We next examine the correlation between new species and fitness in more detail. Figure 3 shows the fitness of the *best* individual of a species and the abundance of species; these data are taken from the same specific GI run as above. On this graph, the y-coordinate of the centre of each bubble is the fitness of the *best* individual of that species; the radius of the bubble is proportional to the abundance of the species in the population. Note the radius of the bubbles *does not* imply anything about the fitness distribution of other members of that species – indeed other individuals of the same species may have far worse fitness. In generation 0 (henceforth gen. 0) the initial, multiplicative hash function (species 0, henceforth sp. 0) has the best fitness value, albeit three other species appeared during this generation's mutation step. Sp. 4 has the best performance in gen. 1, then sp. 7 in gen. 2. Another significant improvement is visible in gen. 4 by species 11 and then in gen. 6 by sp. 15. The improvement in gen. 8 by sp. 15 is via a different implementation of intrinsically the same algorithm. This data demonstrates that major improvements in fitness are the result of new species (i.e., new distributions of elements in the hash table), and not by variations of the same algorithm where the hash table layout is the same. From gen. 20 onwards there are many species close to the best fitness; once such a stage is reached it may then be that minor implementation details determine further fitness improvements.

Figure 3 also gives an insight into the dynamicity of species from creation to extinction; apart from the best fitness, it also shows the abundance of each species. The multiplicative hash is not particularly good for the training dataset, hence its initial high abundance quickly declines and reaches extinction around gen. 10. Sp. 4 did not have the chance to grow, on one hand sp. 0 is still present in high abundance whereas sp. 7 appeared as a fitter alternative. Sp. 15 was the next that could grow to a dominant position around gen. 12. Interesting to notice a few other species in the "shadow" of sp. 15 – they have very good fitness but still

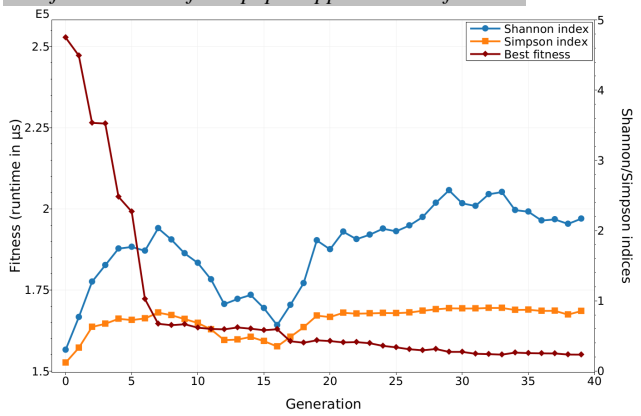

Figure 4: Best overall fitness and the diversity quantified by the Shannon and Simpson indices.

not enough to compete with sp. 15 once it gained its mass. From gen. 21 to gen. 39 there is a competition of species with nearly the same fitness and abundance. Similarly to best fitness, minor code tweaks may be decisive whether a species can grow to dominance, e.g., in gen. 39.

The Shannon-index (Shannon, 1948) and Simpson-index (Simpson, 1949) are common statistical methods to describe biodiversity, which we can also apply to a GI process. The Shannon-index describes the richness of the species landscape overall, while the Simpson-index gives more weight to larger, dominant species. Figure 4 plots both indices against the best individual's fitness over the course of the GI run. Here we confirm the above view that improvements in fitness correlate with the introduction of new species, but we also see that the increase in overall diversity of the species landscape does not necessarily lead to fitness improvement. Broadly speaking the former is evident in the first half of the generations shown in the graph, while the latter is evident in the second half. Between these two phases, around generation 16, a significant reduction is visible in both indices, in alignment with the 4 species in Figure 3. After this generation, the latter phase of the graph shows a high number of species, but most are single-individual groups with just a few species having considerable mass (again see Figure 3).

Species in general aspects

In this section we present macro-level analyses across many experiments, rather than focusing on any single one. For this analysis we use the results of 894 experiments in total to give a more comprehensive insight into the evolutionary process above the level and details of individual experiments.

We first examine whether fitness improvements are generally caused by novel species appearing, or happen via modifications to existing species. Figure 5 summarises the relative improvements (individual vs parent fitness) in performance in the moment when a best fitness was recorded during an evolution, and categorises whether the best performance was an effect of a new species or not. The distributions for the novel species are shown in the left side of the graph, and those for existing species on the right; the vertical axis represents the relative change in fitness, e.g., 1 means the current fitness is the same as that of the parent; lower values are better improvements. Since the chart shows the cases when best fitness was recorded, all relative improvement values are smaller than 1. This graph shows two different effects: firstly that there are *more* cases when a performance improvement was achieved with no new species, at 5057 cases vs. 2813 cases. Of these no-new-species cases, however, the improvement degree is far smaller: 75% of the fitness changes are above 0.98 ($\leq$2% improvement), with a median of 0.99 (1% improvement), with a few cases in the 0.9-0.8 ranges, and very sporadically cases below this. Secondly, it shows that the *relative improvement* is far more significant (Mann-Whitney U test, p-value 0) when at least one more fit individual of a new species arrived: 75% of these new-species cases are below 0.93 ($\geq$7% improvement), 25% are below 0.82 ($\geq$18% improvement), the median of relative change is 0.84 (16% improvement), and there are a large number of cases in the 0.7-0.8 range and around the 0.5 point. The distribution of these statistics is shown in Table 1. This observation generalises the finding from our analysis in Figure 3: considerable changes in performance are almost always brought by the arrival of new species.

	Novel species	Existing species
Mean	0.853	0.977
Median	0.849	0.991
Q1	0.825	0.984
Q3	0.936	0.996

Table 1: Relative fitness improvement by novel and existing species when best fitness was recorded

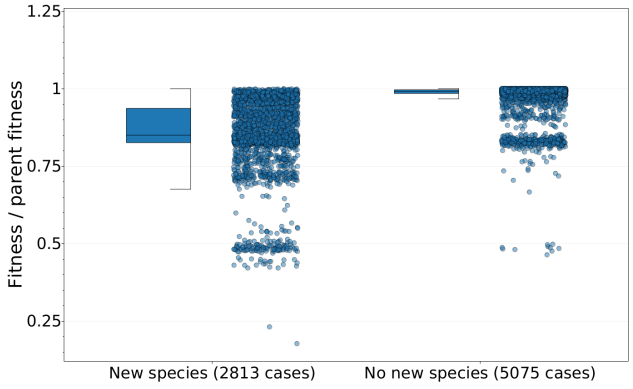

Figure 5: Distribution of delta fitness (current fitness/parent's fitness, lower is better) when best fitness was recorded: in case of new species vs no new species.

We next examine what causes novel species to appear.

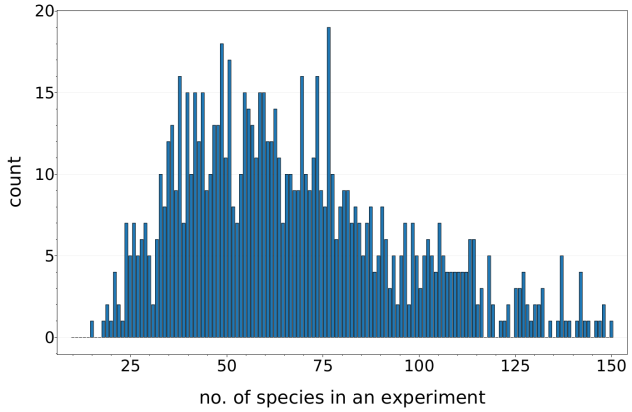

Figure 6: Number of species in an experiment

This analysis helps us to understand cause and effect in general, and may also be useful in future work to help develop steering mechanisms which are likely to provoke the introduction of new species with higher probabilities.

Figure 8 summarises the set of transformations, their frequencies and the proportion of cases where new species were created. Each column of this graph is a stacked bar with two colours: the blue column shows the total count of each transformation, while the orange area shows how many of these transformations yielded new species. In all cases transformations can be composed of both a crossover and a mutation; because classical GI approaches tend to perform both actions within a single generation their effects are not precisely separable. The three main types of mutations are Deletion (D), Insertion (I) and Modification (M); their further sub-types are encoded by the suffixes but in this presentation these further details are not essential and their specifies are omitted. The presence of crossover is indicated by the x prefix. Our first observation is that crossover is generally more successful at creating new species. There are also some types of mutations that appear unable (or at least less frequently able, in the timeframe of these experiments) to spawn new species. On the other hand a certain subcategory of Inserts, in the centre area of the graph, is adept in originating new species either with or without crossover.

The overall number of species in each GI experiment are summarised in Figure 6 as a histogram. The lowest number of species start from 15 (in a singular case) whereas on the other end of the range the highest numbers are well above 100. While the distribution has a generally normative shape it is also highly fluctuating, with the average number of species being 71, the median 64 and in 50% of the experiments the number of species is in the 47-87 range. There is no observed correlation, however, between the number of species by volume and the overall performance improvement in the code for a given run; our analysis rather suggests that having the right species turnover rate within a run has a higher importance to improvement potential.

Finally, we analysed the temporal distribution of the species, Figure 7 shows the number of species over generations across all macro-analysed runs, and for the entire 200 generations of each run (in contrast to the first 40 generations examined in our micro-analysis). The dark-shaded blue box represents the 2nd and 3rd quartiles whereas the grey bracket represent the non-outliers. The overall tendency here is clear: there is a sudden rise in species richness that decays around gen. 50; from gen. 100 the changes then are minimal and from gen. 160 the diversity is negligible.

Discussion

In this section we discuss the potential uses of species, including visual analysis of an evolutionary process; aid in curating a library of high-utility genetic material; and active steering of an evolutionary process to yield novel species.

First, species help with quick visual analysis of the evolutionary process. We can clearly see how species come and go over time, and how these events affect both best-fitness and mean population fitness (the latter being a factor of the relative volume of each species in a given population). Likewise we can clearly see how plateaus in best-fitness correlate to a prolonged lack of novel species. We envision the GI process in an ecosystem of individuals which are dynamically grouped according to certain criteria, such as spatial separation. Speciation represents an orthogonal approach where individuals are grouped by functional attributes. All these dynamic groups represent a higher abstraction above individuals and the evolutionary process could be investigated and analysed as interacting sets of individuals, where speciation and species is a dimension of the ecosystem.

Second, one of the general aims of GI for emergent software is to create a repository of building block variants for deployment upon detecting suitable conditions which correlate with high performance for those variants. This notion of species can help with establishing a meaningful genetic library of stored individuals which are known to be algorithmically different (rather than tweaks to the fine details). Using species metrics to identify and discriminate between such individuals will help to offer a broader spectrum of non-functional attributes when a candidate building block is selected in an adaptation scenario. It may also help to develop a library of genetic material for mixed high-utility starting points of new GI runs: when considering a new improvement of an individual for a novel deployment environment, we can begin a GI process from a set of known-diverse existing species. This may help to maximise the gene pool diversity, encouraging both selection of the best species for a novel environment and the formation of novel species as combinations of diverse starting material.

Finally, perhaps the most intriguing challenge is to consider how the evolutionary process can be steered by speciation. We will further investigate which kinds of transformation, either by type or by location, in the source code

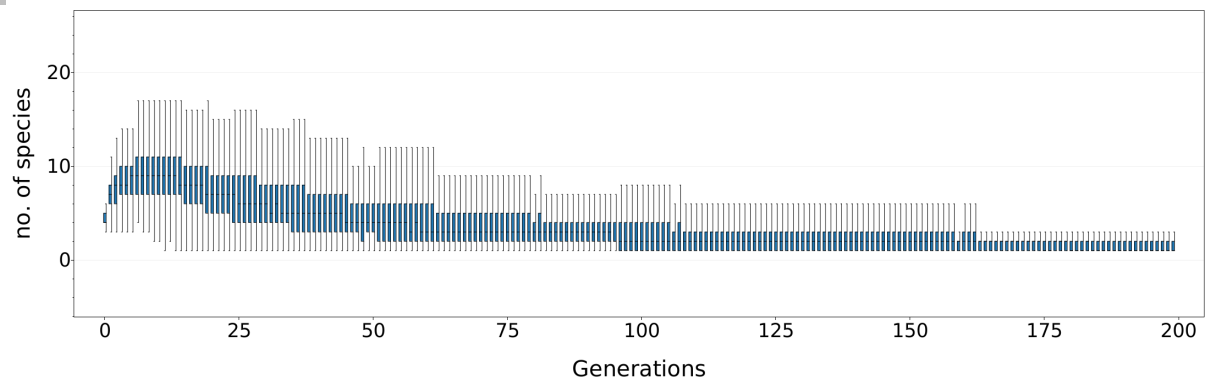

Figure 7: Distribution of the number of species in each generation

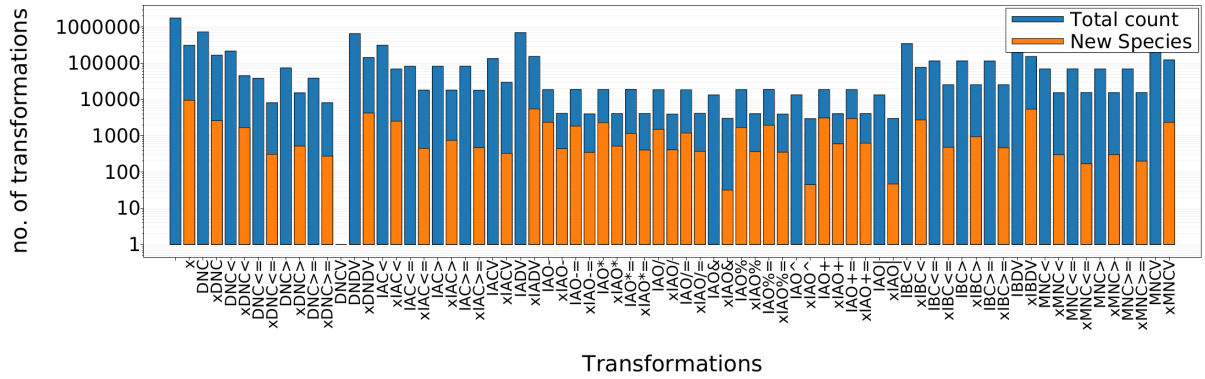

Figure 8: Transform types (mutation and crossover), their overall number as blue bars, and the proportion of cases where new species were created (orange bar) via these transforms.

of individuals can spawn new species and whether the non-functional characteristics of the species can be implied. By understanding how transformations lead to novel species and to extinctions, we may then be able to 'artificially' induce the formation of novel species by deploying these kinds of transformations at the right times with higher probability. Because we know that higher species novelty frequencies tend to lead to better individuals, this may in turn enable a level of control over the convergence of an evolutionary algorithm towards desirable outcomes. We will also investigate in more detail the co-existence of multiple species, and the effects of richness, abundance and dominance. We suspect that it is not the number of species and level of diversity which is pivotal to the outcome of a GI process. Rather, we suspect that the turnover of the species, the rate of change of these metrics which will have the most effect.

Conclusion

We have introduced and investigated the notion of species in GI for source code in emergent software scenarios, with the aim of addressing some of the challenges in exploring the search space. While the concept of species has existed in GA and artificial life scenarios for some time, we are not aware of it having been introduced to GI. Due to the specific nature of GI, genotype-oriented definitions (as commonly used in GA) of species are not applicable. We have therefore proposed a definition of species with the following key features: (i) it is based on a selected functional aspect of phenotype; (ii) it discriminates upon the algorithmic essence of source code; (iii) it provides a fingerprint-like identification of similar individuals; and (iv) it is based on the divergence of probabilistic distribution functions for a given representative problem in a general class of mapping functions.

We have shown that species can give information useful to analysing an evolutionary process, including markers of cause and effect. In future we intend to further investigate species for GI, especially how it can be applied beyond analysis. One pathway is in curating a library of useful variations with measurable diversity; another is in the use of species to directly steer an evolutionary process via targeted mutation interventions to drive towards novel loci of optima.

Acknowledgements

This work was supported by the Leverhulme Trust Research Grant 'Genetic Improvement for Emergent Software', RPG-2022-109.

References

Brownlee, A. E. I., Petke, J., Alexander, B., Barr, E. T., Wagner, M., and White, D. R. (2019). Gin: genetic improvement research made easy. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 985–993, New York, NY, USA. Association for Computing Machinery.

Cuccu, G. and Gomez, F. (2011). When novelty is not enough. In *Applications of Evolutionary Computation*, pages 234–243. Springer Berlin Heidelberg.

Della Cioppa, A., De Stefano, C., and Marcelli, A. (2007). Where are the niches? dynamic fitness sharing. *IEEE Transactions on Evolutionary Computation*, 11(4):453–465.

Dolson, E., Lalejini, A., Jorgensen, S., and Ofria, C. (2020). Interpreting the tape of life: Ancestry-based analyses provide insights and intuition about evolutionary dynamics. *Artif. Life*, 26(1):58–79.

Doncieux, S., Laflaquière, A., and Coninx, A. (2019). Novelty search: a theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 99–106, New York, NY, USA. Association for Computing Machinery.

Dong, N., Wu, C.-H., Ip, W.-H., Chen, Z.-Q., Chan, C.-Y., and Yung, K.-L. (2011). An improved species based genetic algorithm and its application in multiple template matching for embroidered pattern inspection. *Expert Systems with Applications*, 38(12):15172–15182.

Filho, R. R. and Porter, B. (2017). Defining emergent software using continuous Self-Assembly, perception, and learning. *ACM Trans. Auton. Adapt. Syst.*, 12(3):1–25.

Gao, S., Sun, C., Xiang, C., Qin, K., and Lee, T. H. (2022). Learning asynchronous boolean networks from single-cell data using multiobjective cooperative genetic programming. *IEEE Transactions on Cybernetics*, 52(5):2916–2930.

Goldberg, D. E. and Richardson, J. T. (1987). Genetic algorithms with sharing for multimodalfunction optimization. In Grefenstette, J. J., editor, *Proceedings of the 2nd International Conference on Genetic Algorithms, Cambridge, MA, USA, July 1987*, pages 41–49. Lawrence Erlbaum Associates.

Goldberg, D. E. and Wang, L. (1997). Adaptive niching via coevolutionary sharing. pages 21–38.

Jelasity, M. and Dombi, J. (1998). Gas, a concept on modeling species in genetic algorithms. *Artificial Intelligence*, 99(1):1–19.

John, T. C., Abbasi, M. S., Al-Sahaf, H., and Welch, I. (2022). Automatically evolving malice scoring models through utilisation of genetic programming: A cooperative coevolution approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 562–565, New York, NY, USA. Association for Computing Machinery.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Lenski, R. E., Ofria, C., Pennock, R. T., and Adami, C. (2003). The evolutionary origin of complex features. *Nature*, 423(6936):139–144.

Li, J.-P., Balazs, M. E., Parks, G. T., and Clarkson, P. J. (2002). A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.*, 10(3):207–234.

López-López, V. R., Trujillo, L., and Legrand, P. (2018). Novelty search for software improvement of a SLAM system. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 1598–1605, New York, NY, USA. Association for Computing Machinery.

McGowan, C., Wild, A., and Porter, B. (2018). Experiments in genetic divergence for emergent systems. In *Proceedings of the 4th International Workshop on Genetic Improvement Workshop*, GI '18, pages 9–16, New York, NY, USA. Association for Computing Machinery.

Miikkulainen, R. and Forrest, S. (2021). A biological perspective on evolutionary computation. *Nature Machine Intelligence*, 3(1):9–15.

Novozhilov, A. S., Wolf, Y. I., and Koonin, E. V. (2007). Evolution of the genetic code: partial optimization of a random code for robustness to translation error in a rugged fitness landscape. *Biol. Direct*, 2:24.

Petke, J., Haraldsson, S. O., Harman, M., Langdon, W. B., White, D. R., and Woodward, J. R. (2018). Genetic improvement of software: A comprehensive survey. *IEEE Trans. Evol. Comput.*, 22(3):415–432.

Petke, J., Harman, M., Langdon, W. B., and Weimer, W. (2014). Using genetic improvement and code transplants to specialise a c++ program to a problem class. In *Genetic Programming*, pages 137–149. Springer Berlin Heidelberg.

Raghuwanshi, M. and Kakde, O. (2006). Genetic algorithm with species and sexual selection. In *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–8.

Rainford, P. F. and Porter, B. (2022). Using phylogenetic analysis to enhance genetic improvement. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, pages 849–857, New York, NY, USA. Association for Computing Machinery.

Rui L. Lopes, Gonçalo Figueira, P. A. and Almada-Lobo, B. (2020). Cooperative coevolution of expressions for (r,q) inventory management policies using genetic programming. *International Journal of Production Research*, 58(2):509–525.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Simpson, E. H. (1949). Measurement of diversity. *Nature*, 163:688.

van Laar, D. (2021). *Fitness Landscape Analysis applied to functional Genetic Improvement*. PhD thesis.

Villanueva, O. M., Trujillo, L., and Hernandez, D. E. (2020). Novelty search for automatic bug repair. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 1021–1028, New York, NY, USA. Association for Computing Machinery.

Wilkins, J. S. (2009). *Defining species: a sourcebook from antiquity to today*, volume 203. Peter Lang.

Wong, K.-C., Leung, K.-S., and Wong, M.-H. (2009). An evolutionary algorithm with species-specific explosion for multimodal optimization. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 923–930, New York, NY, USA. Association for Computing Machinery.

Zachos, F. E. (2016). *An Annotated List of Species Concepts*, pages 77–96. Springer International Publishing, Cham.