
Proceedings Paper:
Plump, Detlef orcid.org/0000-0002-1148-822X, Atkinson, Timothy and Stepney, Susan orcid.org/0000-0003-3146-5401 (2019) Evolving Graphs with Horizontal Gene Transfer. In: Proceedings Genetic and Evolutionary Computation Conference (GECCO 2019). ACM , pp. 968-976.

https://doi.org/10.1145/3321707.3321788

Evolving Graphs with Horizontal Gene Transfer

Timothy Atkinson, Detlef Plump, Susan Stepney
tja511@york.ac.uk,detlef.plump@york.ac.uk,susan.stepney@york.ac.uk
Department of Computer Science, University of York, York, UK

ABSTRACT

We introduce a form of neutral Horizontal Gene Transfer (HGT) to Evolving Graphs by Graph Programming (EGGP). We introduce the $\mu \times \lambda$ evolutionary algorithm, where μ parents each produce λ children who compete with only their parents. HGT events then copy the entire active component of one surviving parent into the inactive component of another parent, exchanging genetic information without reproduction. Experimental results from 14 symbolic regression benchmark problems show that the introduction of the $\mu \times \lambda$ EA and HGT events improve the performance of EGGP. Comparisons with Genetic Programming and Cartesian Genetic Programming strongly favour our proposed approach.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**;

KEYWORDS

Evolving Graphs, Horizontal Gene Transfer, Neutrality

ACM Reference Format:
Timothy Atkinson, Detlef Plump, Susan Stepney. 2019. Evolving Graphs with Horizontal Gene Transfer. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321788

1 INTRODUCTION

Recombination of genetic material is commonly viewed as a key component of a successful GP (Genetic Programming) system. Koza [12] recommends that most offspring be produced by crossover, rather than by asexual reproduction and mutation. In contrast, CGP (Cartesian Genetic Programming) [15] traditionally uses the elitist $1 + \lambda$ evolutionary strategy, where all offspring are produced by asexual reproduction and mutation; variation and the ability to leave local optima are a byproduct of neutral drift in the neutral parts of the genome [16].

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321788

EGGP (Evolving Graphs by Graph Programming) [1] is a recently introduced graph-based GP approach that operates directly on graph-structured individuals, rather through some 'cartesian' grid encoding as used in CGP and PDGP (Parallel Distributed Genetic Programming) [19]. Each EGGP individual (graph) has an 'active' component that contributes directly to the fitness, and a 'neutral' component that can drift without affecting the fitness. Like CGP, existing work on EGGP has used only asexual reproduction and mutation. Here we extend EGGP to incorporate Horizontal Gene Transfer (HGT) 'events', where the genetic information of one parent is shared with another. Our system operates using the elitist '$\mu \times \lambda$' EA, such that in each generation there are μ parents, which each produce λ children, which compete only with their own parent. This is effectively μ parallel $1 + \lambda$ EAs, with genetic information shared horizontally between elite individuals. To avoid disrupting elitism (by modifying the active components of individuals) or sharing junk (by copying neutral components of individuals), we copy only the active components of one parent onto the neutral component of another; it may later be activated through mutation.

EGGP's individuals, represented as (non-encoded) graphs, are directly modified through the probabilistic graph programming language P-GP 2 [2]. This direct approach eases the conception and implementation of graph-based operations. For example, using edge mutations that consider all possibilities that preserve acyclicty, rather than only those possibilities that preserve the ordering of some Cartesian grid, has been shown to offer faster convergence for standard digital circuit benchmark problems [1]. Additionally, it is possible to incorporate domain specific knowledge, such as Semantic Neutral Drift [3], where logical equivalence laws are applied directly to individuals to create neutral drift in their active components.

Here we replace neutral components with new material directly. This is inspired by Horizontal Gene Transfer (or Lateral Gene Transfer) found in nature. HGT is the movement of genetic material between individuals without mating, and is distinct from normal 'vertical' movement from parents to offspring [11]. HGT plays a key role in the spread of antimicrobial resistance in bacteria [7] and evidence has been found of plant-plant HGT [26] and plant-animal HGT [20]. The mechanism of HGT in transferring a segment of DNA into another individual's DNA may have a clear analogy when considering bit-string based Genetic Algorithms such as the Microbial GA [8], the equivalent analogy is not as obvious when dealing with graphs. Hence we use the term metaphorically: when we refer to HGT, we mean the movement of genetic material between individuals without mating. This is the new mechanism we present in this work.

Our approach is not the first work to recombine and share genetic information in graph-like programs. PDGP uses Subgraph Active-Active Node (SAAN) crossover [19] to share material within a population of Cartesian grid-based programs. A number of crossover operators have been used in CGP, including uniform crossover [14], arithmetic crossover on a vector representation [4], and subgraph crossover [10]. Empirical comparison [9] shows that these crossover operators do not always aid performance, and that CGP with mutation only can sometimes be the best performing approach. Current advice [15, 23] is that the 'standard' CGP approach remains to use mutation only. Our recombination features no modification of active components and does not produce children; nevertheless HGT events followed by edge mutations may perform operations very similar to PDGP SAAN crossover [19] and CGP subgraph crossover [10]. However, our precise mechanism, where active components are pasted into neutral components without any limitations to accessibility, does not obviously translate to PDGP and CGP, which are limited to Cartesian grids.

The rest of this work is organised as follows. In Section 2 we introduce EGGP with a new feature: depth control. In Section 3 we describe our Horizontal Gene Transfer approach, and the $\mu \times \lambda$ EA. In Section 4 we describe experimental settings for comparing our HGT approach to the existing EGGP approach, and to CGP and GP. In Section 5 we present the results of our experiments.

2 EVOLVING GRAPHS BY GRAPH PROGRAMMING (EGGP)

EGGP is a graph-based GP approach where individuals are represented directly as graphs, rather than through some encoding, and are manipulated through graph programming [2]. In this Section, we describe the EGGP approach including details of its representation, initialisation, mutation operators and a new extension, depth control. To distinguish between the original EGGP [1] and EGGP with depth control, we call the former EGGP and the latter EGGP$_{DC}$.

2.1 Representation

In EGGP an individual is a graph. The graph contains indexed input and output nodes, each corresponding to a particular input or output of a given problem. All other nodes are function nodes associated with functions from a chosen function set F. If a node v is associated with function $f \in F$ and the arity of f is $a(f)$, then v has exactly $a(f)$ *outgoing* edges, which indicate the inputs that that function node takes. These outgoing edges are ordered; each edge is labelled with an integer to indicate its position in the order. Ordering removes ambiguity when dealing with non-commutative functions such as division and subtraction. Output nodes have exactly one outgoing edge, indicating that the function computed for that output is given by the behaviour of the node targeted by this single outgoing edge. Output nodes must have no incoming edges, as this would induce some undefined recurrent behaviour.

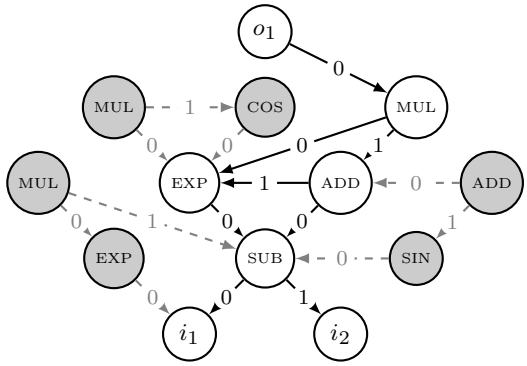


Figure 1: An example EGGP individual. The single output computes the function $o_1 = e^{i_1 - i_2} \times \left(\left(i_1 - i_2\right) + e^{i_1 - i_2}\right)$

Here, the graph is restricted to be acyclic; this ensures that the evolved function is non-recursive. (This constraint could be relaxed to evolve recurrent programs, as in recurrent CGP [22].) An individual is therefore a DAG, with all output nodes as roots, and all input nodes as leaves. Input nodes can be both roots and leaves and other function nodes can also be roots; e.g. if they are targeted by no function nodes or outputs. Wherever there is no directed path from an output node to some node v, v and its outgoing edges are said to be 'neutral', as it does not contribute to the behaviour of the individual. EGGP can undergo 'neutral drift' on its 'neutral' components in a similar manner to CGP [16].

An example EGGP individual is given in Figure 1. There is a single output node, o_1, and two input nodes, i_1 and i_2. neutral nodes and their outgoing edges are coloured grey and dashed respectively; this is a visual aid only. Edge ordering starts at 0; the two outgoing edges of the active SUB node indicate that this node computes the function $i_1 - i_2$, rather than $i_2 - i_1$.

2.2 Initialisation

To generate an individual in EGGP, we begin by creating a graph with i input nodes corresponding to the i inputs associated with a given problem. The parameter n describes the fixed number of function nodes in each individual solution. To generate these n nodes, we repeatedly pick some function f from the function set F and create a new node v_x associated with that function. We then insert edges connecting v_x to any existing node (chosen uniformly at random) until v_x's outdegree matches the function's arity $a(f)$. We repeat this process until there are n function nodes. When using depth control, the inserted edges may only target nodes that would not lead to the individual exceeding the specified maximum depth. Finally, we insert o output nodes corresponding to the o outputs associated with a given problem; each is then connected at random to any other (non output) node in the individual. This approach to initialisation guarantees the generation of an acyclic individual, and in the case of depth control, that generated individuals do not exceed the maximum depth.

2.3 Mutation

2.3.1 Node Mutation. Node mutation is performed by picking uniformly at random a function node to mutate, and changing that node's associated function to some other function chosen at random. Then two fix-up operations are performed.

Firstly, the outdegree (number of outgoing edges) of the mutated node is corrected to match the new function's arity. If the node's outdegree is greater than the new function's arity, edges are chosen uniformly at random and deleted until the outdegree and arity match. If the node's outdegree is less than the new function's arity, edges are inserted targeting valid nodes chosen uniformly at random. A valid node is a node that preserves acyclicity and maximum depth (see Section 2.4). For the original form of EGGP, preserving a maximum depth is not a consideration when choosing a node to target.

Secondly, we reorder the node's outgoing edges. We remove all ordering information from the node's outgoing edges, and assign a new valid random ordering. This process avoids bias in non-commutative functions; for example a node computing $x + y$ can be mutated to compute $x - y$ or $y - x$.

2.3.2 Edge Mutation. Edge mutation is performed by picking uniformly at random an edge to mutate. We then identify all valid targets for that edge (those nodes which preserve acyclicity and maximum depth, excluding the edge's existing target), and redirect the edge to target one of these nodes chosen uniformly at random. In the original form of EGGP, preserving a maximum depth is not a consideration when choosing a node to target.

2.3.3 Mutation Rate. The mutation rate of an individual is m_r. Certain mutations may prevent other mutations. For example, mutating one edge to target some node may then prevent other mutations of that node's outdoing edges with respect to preserving acyclicity. Therefore, iterating through the individual and considering each node or edge in turn for mutation may introduce bias. So our point mutations first choose a random point to mutate, and then mutate it.

We calculate the number of node or edge mutations to apply based on binomial distributions. For an individual with v_f function nodes and e edges, with mutation rate m_r, we sample a number of node mutations $m_v \in \mathcal{B}\big(v_f, m_r\big)$ and edge mutations $m_e \in \mathcal{B}\big(e, m_r\big)$, where $\mathcal{B}\big(n, p\big)$ indicates a binomial distribution with n trials and p probability of success. We then place all $m_v + m_e$ mutations in a list, and shuffle the list, applying mutations in a random order. While this approach is likely to have some biases, it guarantees reproducible probabilistic behaviour. The overall expected number of mutations is $m_r\big(v_f + e\big)$.

2.4 Depth Control

Here we introduce the notion of *depth control* to EGGP. This prevents mutations that would cause a child to exceed a given maximum depth D. We annotate individuals with information regarding the depth associated with each node. The 'depth up' u (or 'depth down' d) of a node is the length of the longest path from that node to a root (or leaf) node. We label each node v with the values $\big(u, d\big)$. An exception is made for output nodes, which have $u = -1$ as their outgoing edges are not considered part of the 'depth' of the individual.

Once an individual has been annotated, we can identify pairs of nodes that, if an edge were inserted between them, would cause the individual to exceed the maximum depth D. If we wish to insert an outgoing edge for node v_1, then we eliminate any other node v_2 as a viable candidate on the following criteria: If the depth up value of v_1 is u_1, and the depth down value of v_2 is d_2, then it is impossible to insert an edge and preserve the maximum depth D if $u_1 + d_2 + 1 > d$: we have a path of length u_1 from v_1 to a root node, and a path of length d_2 from v_2 to a leaf node, hence the overall path from a root to a leaf would be $u_1 + d_2 + 1$, which exceeds D. If $u_1 + d_2 + 1 \leq d$, inserting an edge from v_1 to v_2 would preserve D.

We use this strategy in both edge mutation and node mutation. In edge mutation, we use annotations to identify invalid targets for the mutating edge. In node mutation, we use annotations to identify invalid targets for new edges to be inserted for the mutating node. We give an example of depth preserving edge mutation in Figure 2; an edge of an individual is mutated, but all possible targets that would break acyclicity or a maximum depth $D = 4$ are ignored.

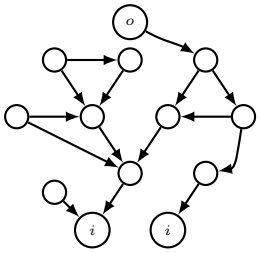
3 HORIZONTAL GENE TRANSFER IN EGGP

In this Section we describe the introduction of Horizontal Gene Transfer events (HGT) to EGGP. HGT events involve the transfer of active material from a donor to the neutral region of a recipient (Section 3.1). To accommodate the need for multiple surviving individuals, we introduce the $\mu \times \lambda$ EA (Section 3.2) as an alternative to the $1 + \lambda$ algorithm previously used in EGGP.

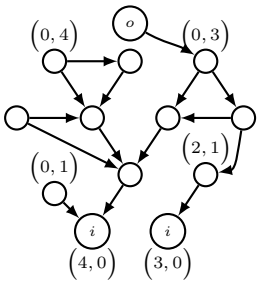
3.1 Active-Neutral Transfer

HGT involves the movement of genetic material between individuals of a population without reproduction. Given a population P, we choose a donor and recipient individual. We copy the entire active component of the donor (excluding output nodes); we remove sufficient neutral material at random from the recipient to fit this active component within the fixed representation size. The copied active component is inserted into the recipient's neutral component, where it remains neutral until it is activated by some mutation. This type of HGT, which we refer to as 'Active-Neutral Transfer', is guaranteed to preserve the fitness of both the donor and recipient, preventing it from disrupting the elitism of the EA. The intention is to promote the production of higher quality offspring by the recipient, by activating its received genetic material through mutation. This process is mutually beneficial; the donor has a mechanism for propagating its genes, while the recipient stands to improve the survivability of its own offspring.

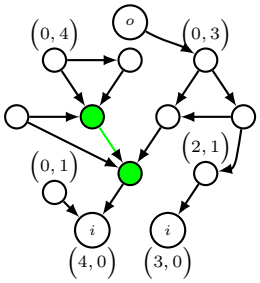
This individual is to undergo an edge mutation preserving acyclicity and a maximum depth $D = 4$.



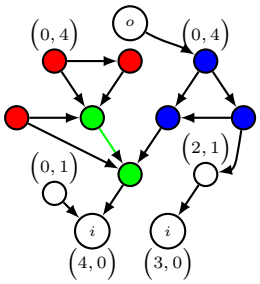
(1) The individual is annotated with depth information. Each node has an associated 'depth up' value u indicating the length of the longest path to a root node (excl. outputs), and a 'depth down' value d indicating the length of the longest path to a leaf node. These are listed as a pair (u, d) for each node.



(2) An edge to mutate is chosen at random and marked (green) alongside its source node s and target node t.



(3) Invalid candidate nodes for redirection are identified. If a node v has a directed path to s it is marked invalid (red), as targeting it would introduce a cycle. If the depth down value of a node v is d_v and the depth up value of s is u_s, when $u_s + d_v + 1 > D$, v is marked invalid (blue), as targeting it would exceed the maximum depth.



(4) The edge e (now shown in red) is mutated to target some randomly chosen unmarked (non-output) node, preserving acyclicity and maximum depth D. Finally, all annotations are removed.

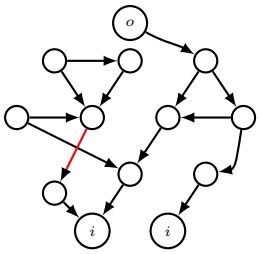


Figure 2: An example of edge mutation preserving acyclicty and depth. Some annotations from step (1) are omitted for visual clarity.

Once material has been transferred, there are a number of possible consequences: the neutral donor material can drift, or become active, through mutation. In this way it is possible for processes such as SAAN crossover in PDGP [19] or block based crossover in CGP [10] to arise out of Active-Neutral transfer followed by mutation.

Our strategy for choosing a donor and recipient is as follows. A recipient is first chosen based on a uniform distribution over the population P, excluding the best performing member. We refer to this 'best performing member' as the 'leader', which we exclude from receiving genetic material so that it can undergo neutral drift without any disruption. Throughout the evolutionary process, it is likely that the leader will change several times, meaning that the entire population is likely to receive genetic material at some point. Once a recipient is chosen, a donor is selected from the population excluding the recipient based on a roulette wheel. The donor may be the leader, allowing the leader to propagate its own genes to other members of the population. The use of a roulette wheel means that any individual can donate material, but the better performing individuals are more likely to do so.

We give an example of Active-Neutral transfer in Figure 3. The entire active component of a gene donor is copied into the neutral material of the recipient while maintaining the overall representation size.

3.2 The $\mu \times \lambda$ EA

We cannot use Active-Neutral transfer with the 1+λ algorithm except for sharing genetic material between the offspring; this is likely to be ineffective as direct offspring have much material in common. We therefore introduce the $\mu \times \lambda$ EA, a special case of the $\mu + \lambda$ algorithm. In each generation of the $\mu \times \lambda$ EA, there are μ parents. Each of the μ parents generates λ offspring, and compete for survival only with their own offspring. Without HGT, this effectively creates multiple parallel $1 + \lambda$ algorithms.

In each generation we perform a single Active-Neutral transfer operation with probability p_{HGT}. We then follow the procedure set out in 3.1 by selecting a gene recipient from the μ parents (ignoring the best performing parent, the 'leader') and selecting a donor from the remaining $\mu - 1$ parents by roulette selection.

4 EXPERIMENTS

Here we detail our experimental settings for benchmarking our HGT approach, EGGP$_{HGT}$, on various symbolic regression problems. We compare EGGP$_{HGT}$ to: standard EGGP; the depth control variant EGGP$_{DC}$; the depth control variant using the $\mu \times \lambda$ algorithm (and no HGT), EGGP$_{\mu\times\lambda}$. These experiments allow us to test the following null hypotheses:

- H_1: there are no statistical differences when using the depth control variant EGGP$_{DC}$ in comparison to standard EGGP.
- H_2: there are no statistical differences when using the $\mu \times \lambda$ algorithm for EGGP in comparison to the $1 + \lambda$ algorithm, with both approaches using depth control.

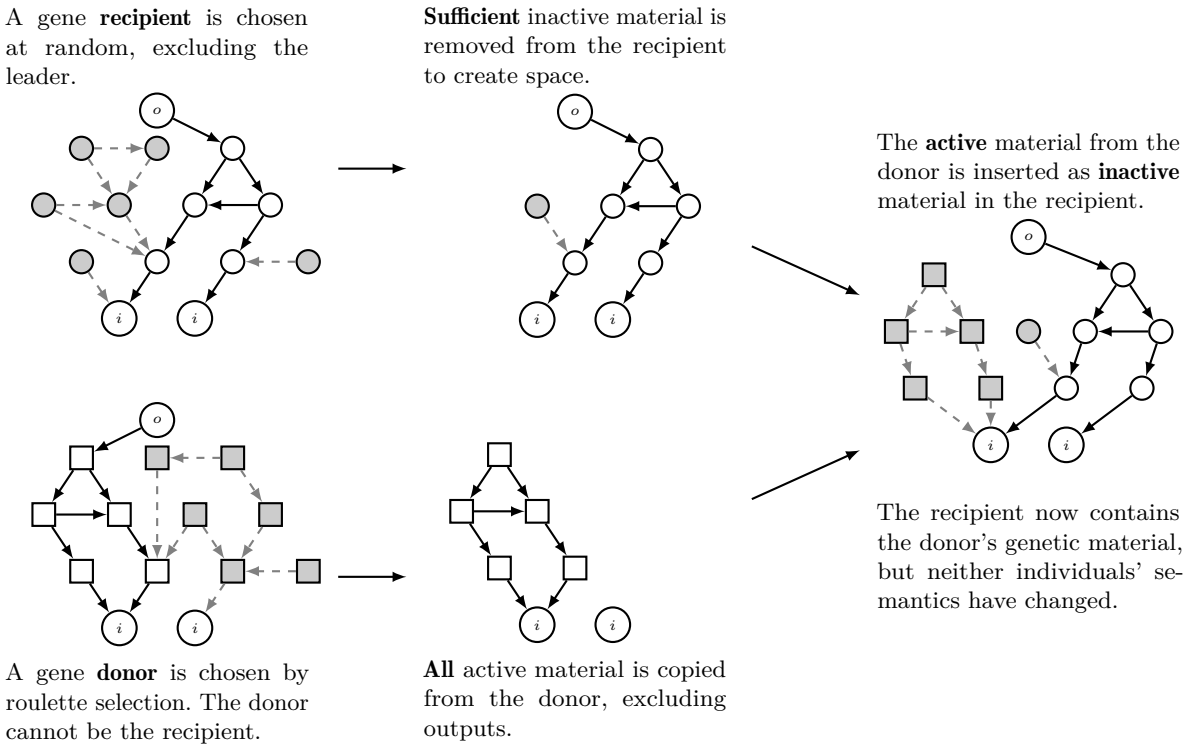


Figure 3: An example of Active-Neutral transfer. The active material of a donor is copied into the neutral material of a recipient. Neither individuals' semantics is changed by this process. Grey nodes and dashed edges indicate the neutral material of individuals; they do not indicate any actual information stored on the individual. The donor's function nodes are shown as squares for clarity.

- H_3: there are no statistical differences when using the HGT approach for EGGP in comparison to using the $\mu \times \lambda$ algorithm without HGT, with both approaches using depth control.
- H_4: there are no statistical differences when using the HGT approach for EGGP in comparison to standard EGGP.

We test these null hypotheses for each benchmark problem. From these tests, we build an image of how the various features contribute to the performance of EGGP_{HGT}, and clarify whether the added HGT feature is truly improving performance by isolating it from the other new features.

We also compare our HGT approach to two other approaches from the literature: Genetic Programming (GP) [12] and CGP [15]. These experiments allow us to test the following null hypotheses:

- H_5: there are no statistical differences when using EGGP_{HGT} in comparison to GP.
- H_6: there are no statistical differences when using EGGP_{HGT} in comparison to CGP.

Again, we test each of these null hypotheses for each benchmark problem. H_5 and H_6 allow us to measure the progress made by introducing HGT to EGGP in comparison to other approaches in literature.

4.1 Benchmark Problems

We benchmark the approaches on 21 synthetic symbolic regression problems [17]. That work justifies the exclusion of Grammatical Evolution (GE) [18], as it finds that GP generally outperforms GE on these problems. For all 21 problems, see [17]; one example is:

$$F_7\big(x_1, x_2\big) = \frac{\big(x_1 - 3\big)^4 + \big(x_2 - 3\big)^2 + \big(x_2 - 3\big)}{\big(x_2 - 2\big)^4 + 10} \qquad (1)$$

These benchmarks were introduced in response to various criticisms of the GP community for 'arbitrarily' chosen benchmark problems, and the reasoning for their design is set out in detail in [17]. We view these problems as good measures of performance of a GP system. Of the 21 problems, 9 take 2 inputs, 1 takes 3 inputs, 8 take 5 inputs, 1 takes 6 inputs and 3 take 10 inputs. Each function's input variables are randomly sampled from the interval $[-5, 5]$.

We use 1000 training samples, 10,000 validation samples and 40,000 test samples. The training data is used to guide the different approaches, while every solution explored is evaluated on the validation data. The globally best performing individual (with respect to the validation data) is returned at the end of a run, and then evaluated on the test data to produce a test performance measure.

The function set for these problems is that of [17]:

$$\{+, -, \times, \div, e^x, \ln x, \sin x, \tanh x, \sqrt{x}\} \quad (2)$$

Each approach has access to the 18 constants -0.9, -0.8, ..., $-0.1, 0.1, 0.2$, ..., 0.9. In GP these are constants, whereas in the EGGP variants and CGP, they are further input nodes.

4.2 Experimental Settings

We evaluate all individuals using the Mean Square Error (MSE) fitness function.We measure statistics taken over 100 independent runs of each approach on each dataset.

For all EGGP variants, we use a fixed 100 nodes and a mutation rate $m_r = 0.03$. For EGGP and EGGP_{DC} we use the $1 + \lambda$ EA with $\lambda = 4$; for $\text{EGGP}_{\mu\times\lambda}$ and EGGP_{HGT} we use $\mu = 3$ and $\lambda = 1$. This induces a 'minimal' version of the $\mu \times \lambda$ algorithm with $\mu = 3$ being the minimal value we could choose for μ such that HGT occurs not only from the 'leading' thread, but also between threads, and $\lambda = 1$ being the minimal value for λ. For EGGP_{DC}, $\text{EGGP}_{\mu\times\lambda}$ and EGGP_{HGT} we set a maximum depth of $D = 10$, and limit the maximum size to 50 active nodes. The maximum active size is ensured by removing and replacing any generated individual that exceeds the maximum size; it is necessary to prevent errors in the HGT approach where, for example, the size of the donor's active component exceeds that of the recipient's neutral component (causing the overall number of nodes to grow when copying the entire active component over). In practice, this condition is used in very few instances, as depth control constrains the size. The rate p_{HGT} is 0.5.

For CGP, we use the experimental parameters in [25], [15, Ch.3], at which values CGP outperforms GP on symbolic regression problems. We use 100 fixed nodes, and a mutation rate of 0.03. We use the $1 + \lambda$ EA with $\lambda = 4$. We do not use any of the published CGP crossover operators, as their usefulness, particularly on symbolic regression problems, remains disputed [9], and [15, 23] recommend the $1 + \lambda$ approach. We also use no form of depth control with CGP, as the approach is known to have inherent anti-bloat biases [21].

For GP, we use the experimental parameters in [17] with a minor adjustment. The population size is 500, with 1 elite individual surviving in each generation. Subtree crossover is used with a probability of 0.9, and when it is not used, the 'depth steady' subtree replacement mutation operator is used, which, when replacing a subtree of depth d generates a new subtree of depth between 0 and d [17]. Tournament selection is used to select reproducing individuals, with a tournament size of 4, and the maximum depth allowed of any individual is 10. Unusually for GP, we add each new individual to the population one-by-one, discarding one of the children produced by each crossover operator. This allows us to immediately replace invalid individuals with respect to the maximum depth, guaranteeing that every individual in a new population is valid and should be evaluated. To initialize the population, we use the ramped half-and-half technique [12], with a minimum depth of 1 and a maximum depth of 5.

For all experiments, the maximum number of evaluations allowed is 24 950, a value taken from [17] (50 generations with a population size of 500 and 1 elite individual that does not require re-evaluating). In GP this is achieved by allowing the search to run for 50 generations. In EGGP and CGP, we use the optimisation from [15, Ch.2], where individuals are evaluated only when their active components are mutated; there is no fixed number of mutations, and the search continues until the total number of evaluations is performed. There is no analogous optimisation for GP, as GP individuals contain no neutral material. This optimisation makes a large difference to the depth of search; for example, in CGP running on F_1, the median number of generations is 12 385, but if all individuals are evaluated (rather than only those with active region mutations), the number of generations would be capped at 6237 (assuming elite individuals are never re-evaluated).

4.3 Implementation

Our implementation of the EGGP variants described here is based upon the publicly available core EGGP implementation[1]. EGGP mutation operators and depth annotation are prototyped as P-GP 2 programs [2], then re-implemented in more efficient C code for the actual experiments. There are currently no known implementations of topological sort (used to calculate depth) in P-GP 2 that are better than quadratic time with respect to the size of the input graph, while such a sort is possible in linear time in C. HGT events are implemented as P-GP 2 programs.

Our CGP experiments are based on the publicly available CGP library [23] with modifications made to accommodate the 'active evaluations only' optimisation and the use of validation and training sets. Our GP experiments are based on the DEAP evolutionary computation framework [5] with modifications made to accommodate our crossover strategy, mutation operator, and use of validation and training sets.

5 RESULTS

Our experimental results are given in Table 1. Results for benchmarks F13–17, 19, 20 (omitted here) show very little variety in performance; the results of [17] suggests these are poor benchmark problems in that the functions are almost invariant on their inputs. While F1–3 also exhibit relatively invariant responses, approaches here and in [17] produce a variety of performances that compel their inclusion. Similarly, while F4 and F21 do not show a variety of performances, the functions themselves produce a variety of responses on different inputs, again compelling their inclusion.

Table 1 lists the median fitness (MF) and inter-quartile range in fitness (IQR) of each approach on each dataset over 100 runs. Overall, the lowest MF score is achieved by EGGP_{HGT} in 10 cases, EGGP_{DC} in 2 cases and GP in 2 cases. There are no cases where EGGP, $\text{EGGP}_{\mu\times\lambda}$ or CGP achieve the lowest MF score.

To test for statistical significance we use the two-tailed Mann-Whitney U test [13], which (essentially) tests the null hypothesis that two distributions have the same medians (this non-parametric analogue of the t-test does not assume

[1] https://github.com/UoYCS-plasma/EGGP

	EGGP		**EGGP**$_{DC}$		**EGGP**$_{\mu\times\lambda}$		**EGGP**$_{HGT}$		**GP**		**CGP**	
F	MF	IQR	MF	IQR	MF	IQR	MF	IQR	MF	IQR	MF	IQR
F1	4.45E-3	7.35E-3	6.26E-3	6.45E-3	3.59E-3	1.39E-3	**2.47E-3**	1.79E-3	5.77E-3	3.40E-3	6.74E-3	4.30E-3
F2	8.17E6	6.05E6	1.41E7	9.95E6	8.06E6	5.02E6	**5.94E6**	3.06E6	1.28E7	7.86E6	1.73E7	2.54E6
F3	1.18E-2	7.34E-3	1.48E-2	4.27E-3	9.92E-3	3.82E-3	**7.22E-3**	4.00E-3	1.04E-2	3.56E-3	1.48E-2	4.39E-3
F4	2.58E13	1.05E9	**2.58E13**	3.57E8	2.58E13	7.51E10	2.58E13	1.96E9	3.55E13	8.35E13	2.58E13	2.35E9
F5	3.96E0	3.56E0	4.48E0	4.30E0	2.30E0	2.61E0	**6.90E-1**	2.08E0	5.13E0	3.81E0	7.17E0	1.47E0
F6	1.69E1	2.24E1	2.11E1	3.99E1	7.23E0	1.18E1	4.46E0	6.24E0	**2.61E0**	6.86E0	9.28E0	2.03E1
F7	3.06E2	7.40E2	4.16E2	6.76E2	2.20E2	1.53E2	**1.51E2**	9.62E1	4.20E2	3.50E2	5.76E2	4.39E2
F8	3.91E-2	7.43E-2	1.03E-1	1.13E-1	2.85E-2	2.00E-2	**2.19E-2**	1.21E-2	1.09E-1	4.99E-2	4.49E-2	9.59E-2
F9	7.09E2	5.40E3	2.59E3	1.36E4	1.81E2	3.68E2	1.57E2	3.53E2	**1.46E2**	3.04E1	1.71E2	1.11E3
F10	1.52E-1	2.05E-1	2.36E-1	2.22E-1	1.07E-1	8.30E-2	**7.69E-2**	5.75E-2	3.22E-1	5.62E-2	1.66E-1	1.42E-1
F11	3.93E1	7.26E1	4.53E1	6.33E1	2.43E1	1.37E1	**1.59E1**	1.20E1	3.88E1	3.37E1	4.96E1	4.73E1
F12	1.21E3	5.25E2	1.22E3	5.20E2	6.95E2	1.19E2	**6.83E2**	1.44E2	1.25E3	5.02E1	7.08E2	5.19E2
F18	4.07E4	9.27E3	4.08E4	3.91E4	4.40E3	3.86E4	**3.69E-1**	2.07E4	4.13E4	3.54E2	1.20E2	4.10E4
F21	1.07E0	6.16E-4	**1.07E0**	1.38E-5	1.07E0	7.74E-4	1.07E0	6.88E-4	1.07E0	4.90E-4	1.07E0	1.53E-5

Table 1: Results from Symbolic Regression benchmarks as described in Section 4. MF indicates the Median Fitness over observed runs; the lowest (best) MF result across all algorithms is highlighted in bold. IQR indicates the Inter-quartile range in fitness.

	H_1		H_2		H_3		H_4		H_5		H_6	
F	p	A	p	A	p	A	p	A	p	A	p	A
F1	0.08	-	$< \alpha$	**0.76**	$< \alpha$	**0.71**	$< \alpha$	**0.76**	$< \alpha$	**0.92**	$< \alpha$	**0.91**
F2	$< \alpha$	0.70	$< \alpha$	**0.76**	$< \alpha$	0.68	$< \alpha$	**0.71**	$< \alpha$	**0.87**	$< \alpha$	**0.95**
F3	$< \alpha$	0.68	$< \alpha$	**0.82**	$< \alpha$	0.70	$< \alpha$	**0.72**	$< \alpha$	**0.75**	$< \alpha$	**0.91**
F4	0.98	-	0.33	-	0.08	-	0.52	-	$< \alpha$	0.68	0.89	-
F5	0.06	-	$< \alpha$	**0.76**	$< \alpha$	0.70	$< \alpha$	**0.84**	$< \alpha$	**0.86**	$< \alpha$	**0.99**
F6	0.26	-	$< \alpha$	**0.78**	$< \alpha$	0.63	$< \alpha$	**0.84**	0.37	-	$< \alpha$	0.63
F7	0.12	-	$< \alpha$	**0.74**	$< \alpha$	**0.71**	$< \alpha$	**0.76**	$< \alpha$	**0.93**	$< \alpha$	**0.94**
F8	$\geq \alpha$	-	$< \alpha$	**0.75**	$< \alpha$	0.62	$< \alpha$	**0.77**	$< \alpha$	**0.95**	$< \alpha$	**0.79**
F9	0.02	-	$< \alpha$	**0.78**	0.77	-	$< \alpha$	0.69	0.23	-	0.17	-
F10	0.01	-	$< \alpha$	**0.74**	$< \alpha$	0.65	$< \alpha$	**0.76**	$< \alpha$	**0.99**	$< \alpha$	**0.81**
F11	0.57	-	$< \alpha$	**0.76**	$< \alpha$	**0.73**	$< \alpha$	**0.85**	$< \alpha$	**0.90**	$< \alpha$	**0.89**
F12	0.85	-	$< \alpha$	**0.76**	0.12	-	$< \alpha$	**0.81**	$< \alpha$	**0.89**	0.15	-
F18	0.84	-	$< \alpha$	**0.71**	$< \alpha$	0.68	$< \alpha$	**0.85**	$< \alpha$	**0.91**	$< \alpha$	0.62
F21	$\geq \alpha$	-	$< \alpha$	0.66	0.11	-	0.57	-	0.32	-	$< \alpha$	0.62

Table 2: Statistical tests for hypotheses H_1 - H_6. The p value is from the two-tailed Mann-Whitney U test. The corrected threshold for statistical significance is $\alpha = \frac{0.05}{14}$. Where $p < \alpha$, the effect size from the Vargha-Delaney A test is shown; large effect sizes ($A > 0.71$) are shown in bold. Where $\alpha \leq p < 0.005$, p is listed as $\geq \alpha$.

normally distributed data). We use a significance threshold of 0.05 and perform a bonferroni procedure for each hypothesis giving a corrected significance threshold of $\alpha = \frac{0.05}{14}$. Where we get a statistically significant result ($p < \alpha$), we also calculate the effect size, using the non-parametric Vargha-Delaney A Test [24]. $A \geq 0.71$ corresponds to a large effect size. These results of these statistical tests for all hypotheses are given in Table 2.

5.1 Building EGGP$_{HGT}$: H_1, H_2, H_3, H_4

The introduction of depth control (H_1) appears to have relatively little effect and is sometimes detrimental. In 12 of our benchmark problems, we observe no significant difference when introducing the feature. On 2 problems, standard EGGP achieves a statistically significant lower (better) median fitness than EGGP$_{DC}$, but never with large effect. These results indicate that depth control is not necessarily a helpful feature for EGGP, but never causes EGGP to outperform EGGP$_{DC}$ with large effect, and in many cases makes no

significant difference to performance. This implies that the performance of EGGP$_{HGT}$ (discussed later) cannot be explained by its new depth control feature alone. We suggest that these results may be due to neutral material contributing to active nodes' 'depth up' values, preventing the active component from undergoing certain mutations even if these mutations would produce an active component of a valid depth. There may be circumstances where this restriction of the landscape hinders the performance of EGGP$_{DC}$.

Comparing EGGP$_{\mu\times\lambda}$ and EGGP$_{DC}$ (H_5) we find that the introduction of the $\mu \times \lambda$ algorithm yields a statistically significant lower median fitness and a large effect size on 12 of the 14 problems. On 1 problem (F4) there is no significant difference, and on 1 problem (F21) EGGP$_{DC}$ achieves a statistically significant lower median fitness, but without large effect. Overall, our study of H_2 provides substantial evidence that the $\mu \times \lambda$ algorithm aids the performance of EGGP, and should potentially be adopted generally.

The differences between EGGP$_{HGT}$ and EGGP$_{\mu\times\lambda}$ (H_3) are more subtle than the comparison of H_2, but there is a prevalent trend. The introduction of HGT yields a statistically significant lower median fitness in 10 problems, 3 of which occur with large effect, and no significant differences on the other 4. These results suggest that HGT is, generally, a beneficial feature capable of yielding major differences in performance. We observe no instances where HGT leads to a significant decrease in performance.

Overall, the results from studying our hypotheses H_1, H_2 and H_3 allow us to explain the success of EGGP$_{HGT}$ in comparison to GP and CGP (discussed in Section 5.2) as a composition of the core EGGP approach, the use of the $\mu\times\lambda$ EA and the introduction of Active-Neutral HGT events. Each of our 3 new features has been added to our overall approach in isolation, allowing us to isolate the beneficial properties of $\mu \times \lambda$ and HGT events. The role of depth control remains unclear from our investigations; it appears to be an unhelpful feature alone but may interact with the HGT process with respect to maintaining smaller individuals. An extended investigation into the role of depth control in our designed approach is desirable in the future.

H_4 compares our final proposed approach EGGP$_{HGT}$ to our original EGGP approach. The proposed approach achieves a statistically significant lower median fitness in 12 of the 14 problems; 11 of which occur with large effect. On the 2 remaining problems, we observe no significant differences. Therefore the combination of our 3 features – depth control, $\mu\times\lambda$ and horizontal gene transfer – lead to a marked improvement over standard EGGP for the studied problems.

5.2 EGGP$_{HGT}$ vs. GP & CGP: H_4, H_6

EGGP$_{HGT}$ achieves a statistically significant lower median fitness in comparison to GP (H_5) on 11 problems, 10 of which show a large effect. On the other 3 problems, we observe no statistical differences. On a clear majority of the studied problems, EGGP$_{HGT}$ significantly outperforms a standard GP system, and is never outperformed by that GP system.

EGGP$_{HGT}$ achieves a statistically significant lower median fitness in comparison to CGP (H_6) on 11 problems, 9 of which show a large effect. On 3 of the other 4 problems, there is no significant difference, and on only 1 problem (F21) is there a statistical difference favouring CGP, but without large effect. Hence we have EGGP$_{HGT}$ significantly outperforming CGP under similar conditions on a majority of benchmark problems, and only outperformed on 1 problem.

Collectively, these results place EGGP$_{HGT}$ favourably in comparison to the literature. Although our experiments are not exhaustive – they are not the product of full parameter sweeps, but rather are testing approaches under standard conditions – they demonstrate that EGGP with Horizontal Gene Transfer is a viable and competitive approach for symbolic regression problems.

6 CONCLUSIONS & FUTURE WORK

In this work we have introduced a new and effective form of neutral Horizontal Gene Transfer in the EGGP approach. Our approach utilises Active-Neutral transfer to copy the active components of one elite parent into the neutral material of another. Experimental results show that both HGT and the introduction of the $\mu \times \lambda$ EA lead to improvements in performance on benchmark symbolic regression problems. Comparing the final approach, EGGP$_{HGT}$, to GP and CGP yields highly favourable results on a majority of problems.

These results have implications for broader research in evolutionary algorithms and genetic programming. The reuse and recombination of genetic material is generally assumed to be a useful feature of an evolutionary system (e.g. GP crossover [12]), but our Active-Neutral HGT events achieve reuse without altering the active components of individuals. Hence our approach contributes evidence to the notion that neutral drift aids evolutionary search [6]. Active-Neutral HGT events move beyond neutrality through mutation; we are effectively biasing the neutral components of individuals towards areas of the landscape we know to be 'good' with respect to the fitness function. While this is empirically beneficial here, it remains unknown whether this neutral biasing is helpful outside of the EGGP approach. Our favourable comparisons with GP and CGP support this direction of thought; GP offers recombination without neutral drift, whereas (vanilla) CGP offers neutral drift without recombination.

Our work here opens up a number of avenues for further research. It is desirable to investigate the influence of population parameters μ, λ and the HGT rate p_{HGT} on the performance of the described approach. Here, we have chosen small values of μ and λ and a relatively high p_{HGT}; it is therefore interesting to consider whether larger values of μ and λ help or hinder the HGT process, and whether it is necessary to introduce multiple HGT events in a single generation when using larger populations. Additionally, an investigation isolating depth control from HGT would help clarify whether HGT is more useful when individuals are smaller or larger.

REFERENCES

[1] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Evolving graphs by graph programming. In *Proc. European Conference on Genetic Programming, EuroGP 2018 (LNCS)*, Vol. 10781. Springer, 35–51.

[2] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Probabilistic Graph Programs for Randomised and Evolutionary Algorithms. In *Proc. International Conference on Graph Transformation, ICGT 2018 (LNCS)*, Vol. 10887. Springer, 63–78.

[3] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Semantic Neutral Drift. *arXiv preprint arXiv:1810.10453* (2018).

[4] Janet Clegg, James Alfred Walker, and Julian Frances Miller. 2007. A new crossover technique for cartesian genetic programming. In *Proc. 9th annual conference on Genetic and evolutionary computation, GECCO 2007*. ACM, 1580–1587.

[5] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13, Jul (2012), 2171–2175.

[6] Edgar Galván-López, Riccardo Poli, Ahmed Kattan, Michael O'Neill, and Anthony Brabazon. 2011. Neutrality in evolutionary algorithms... What do we know? *Evolving Systems* 2, 3 (2011), 145âĂŞ163.

[7] Carlton Gyles and Patrick Boerlin. 2014. Horizontally transferred genetic elements and their role in pathogenesis of bacterial disease. *Veterinary pathology* 51, 2 (2014), 328–340.

[8] Inman Harvey. 2009. The microbial genetic algorithm. In *European Conference on Artificial Life, ECAL 2009 (LNCS)*, Vol. 5778. Springer, 126–133.

[9] Jakub Husa and Roman Kalkreuth. 2018. A Comparative Study on Crossover in Cartesian Genetic Programming. In *Proc. European Conference on Genetic Programming, EuroGP 2018 (LNCS)*, Vol. 10781. Springer, 203–219.

[10] Roman Kalkreuth, Günter Rudolph, and Andre Droschinsky. 2017. A new subgraph crossover for cartesian genetic programming. In *Proc. European Conference on Genetic Programming, EuroGP 2017 (LNCS)*, Vol. 10196. Springer, 294–310.

[11] Patrick J Keeling and Jeffrey D Palmer. 2008. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics* 9, 8 (2008), 605.

[12] John R Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Vol. 1. MIT Press.

[13] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Statist.* 18, 1 (1947), 50–60.

[14] Julian F Miller. 1999. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proc. 1st Annual Conference on Genetic and Evolutionary Computation, GECCO '99*, Vol. 2. Morgan Kaufmann Publishers Inc., 1135–1142.

[15] Julian F Miller. 2011. Cartesian genetic programming. In *Cartesian Genetic Programming*. Springer, 17–34.

[16] Julian F Miller and Stephen L Smith. 2006. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation* 10, 2 (2006), 167–174.

[17] M. Nicolau, A. Agapitos, M. O'Neill, and A. Brabazon. 2015. Guidelines for defining benchmark problems in Genetic Programming. In *2015 IEEE Congress on Evolutionary Computation, CEC*. 1152–1159.

[18] Michael O'Neill and Conor Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (2001), 349–358.

[19] Riccardo Poli. 1997. Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming. In *Proc. International Conference on Genetic Algorithms, ICGA*. Morgan Kaufmann, 346–353.

[20] Julie A Schwartz, Nicholas E Curtis, and Sidney K Pierce. 2014. FISH labeling reveals a horizontally transferred algal (Vaucheria litorea) nuclear gene on a sea slug (Elysia chlorotica) chromosome. *The Biological Bulletin* 227, 3 (2014), 300–312.

[21] Andrew Turner and Julian Miller. 2014. Cartesian Genetic Programming: Why No Bloat?. In *Proc. European Conference on Genetic Programming, EuroGP 2014 (LNCS)*, Vol. 8599. Springer, 222–233.

[22] Andrew James Turner and Julian Francis Miller. 2014. Recurrent cartesian genetic programming. In *Proc. International Conference on Parallel Problem Solving from Nature, PPSN 2014 (LNCS)*, Vol. 8672. Springer, 476–486.

[23] Andrew James Turner and Julian Francis Miller. 2015. Introducing a cross platform open source Cartesian Genetic Programming library. *Genetic Programming and Evolvable Machines* 16, 1 (2015), 83–91.

[24] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[25] James Alfred Walker and Julian Francis Miller. 2008. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation* 12, 4 (2008), 397–417.

[26] Satoko Yoshida, Shinichiro Maruyama, Hisayoshi Nozaki, and Ken Shirasu. 2010. Horizontal Gene Transfer by the Parasitic Plant Striga hermonthica. *Science* 328, 5982 (2010), 1128–1128.