A Neutral Rewrite Mutation Operator for Genetic Programming applied to Boolean Domain Problems

Dmytro Vitel and **Alessio Gaspar**
University of South Florida
4202 E Fowler Ave, Tampa, FL 33620

Paul Wiegand
Winthrop University
701 Oakland Ave, Rock Hill, SC 29730

Abstract

The effect of semantically neutral tree rewrites is analyzed in the context of genetic programming applied to Boolean domain problems. Different setups of the proposed Neutral Rewrite Operator are studied from the perspective of improving performance.

Introduction

The question of the impact of neutral mutations on evolutionary algorithms is a somewhat controversial topic in the evolutionary computation community. These mutations are changes to the encoding of a candidate solution that do not impact its quality ("fitness"). While some consider neutrality useless, (Smith, Husbands, and O'Shea 2001; Collins 2006) others argue that it could facilitate the self-adaptation of populations (Igel and Toussaint 2003). We propose to investigate the benefits of a neutral mutation operator, in terms of convergence speed, when used with Koza-style Genetic Programming (GP) (Koza 1994) and applied to Boolean domain problems. Unlike previous approaches, we leverage sets of syntactic rewriting rules ("transforms") to implement the proposed operator.

We use transforms to define the specific genotype changes that will represent possible neutral mutations. By doing so, we also enable the introduction of domain-specific knowledge into the transforms. In the specific problems considered in this paper, we use Boolean algebra laws to define neutral transforms. Interestingly, this also means that our transforms are not problem-specific but applicable to all problems in the Boolean domain. It is not the goal of this paper to make claims about the optimality of any given set of transforms for the Boolean domain, as a whole, or any specific problem. Instead, we use a minimalist set of transforms, selected while avoiding biases, and focus on establishing the potential of the very idea of using such transforms.

In order to establish the potential benefits of neutral mutations, we consider a set of problems from the Boolean domain that are commonly used as benchmarks in the GP literature (Mcdermott et al. 2012). Our results indicate that introducing neutrality, based on the proposed transforms, is beneficial for most of the selected problems.

Related Works

In biological evolutionary process, Kimura (1968) showed that neutral mutations occur with high frequency. For the average length of one generation (mammalian evolution, 4 years) their rate is roughly two per gamete and four per zygote. The importance of this neutral random genetic drift (contrasted to Darwinian selectivity) has also been considered by the evolutionary computation community.

Igel and Toussaint (2003) approach neutrality from the perspective of genotype and phenotype selection distribution (exploration distribution) functions on each generation. The idea the authors convey is that not only genotypes evolve in the process but also the distributions used in the selection processes. Generalized self-adaptation of a population is its ability to adjust the selection distribution with the help of neutrality. If within the space of genotypes a subset maps to the same phenotype, "movement" inside this subset could result in totally different exploration distribution by the search method, which might speedup convergence. The authors consider several ways to navigate neutral sets (e.g., embedding some strategic parameters into genotypes, natural redundancy...). Instead, We leverage tree rewrites.

Oesch and Maringer (2015) develop a neutral mutation operator for grammatical evolution (GE) (O'Neill and Ryan 2001). In GE, genotypes are fixed-length integer vectors. Each integer value is used to select among alternative rules at each step of the derivation. Since the number of alternatives is smaller than the range of allowed integer values, a modulo operator is applied. The proposed neutral mutation operator modifies integer values so as to preserve the same result when the corresponding modulo is applied. This aproach is validated on symbolic regression of polynomials.

In contrast, Galván-López and Rodríguez-Vázquez (2006) use "introns" to obtain different genotypes encoding the same programs. In Biology, introns are genetic material that does not directly impact the fitness of an individual (Wineberg and Oppacher 1996). The authors introduce a special GP node (p-node) that, when traversing the GP tree to evaluate the individual, redirects the traversal to a random location in the same tree. As such, p-nodes' children are introns: they are not expressed during evaluation and thus do not contribute to the fitness. P-nodes isolate islands of GP-tree which are still used by GP operators. This approach is validated on the 6-bit multiplexer problem (MUX-6).

Yu and Miller (2001) show the usefulness of neutrality in Cartesian GP, using the 3-bit Boolean Parity problem (PAR-3). The authors use additional genes that control the state (active/inactive) of inner circuits (Boolean functions). Their experimental results show that more neutrality is better, regardless of the mutation rates. Later, Yu and Miller (2002) also argue that neutrality can play a critical role in "needle-in-the-haystack" problems. However, Collins (2006) disproved this by showing that the asymmetry in the intron mappings does in fact cause damage to the ability of the algorithm to sample the space effectively.

Proponents of the semantic GP approach Pawlak and Krawiec (2018) adopted the diametrically opposed perspective on neutral mutations. Their "competent" operators for population initialization, selection, mutation, and crossover are designed on the premise to guarantee non-semantically neutral "effective" changes to the programs being evolved. Furthermore, they are also "geometric", in the sense that they guarantee movement along a specific direction in the semantic space. In their work, neutrality is considered wasteful. Instead, population diversity is maintained by diversifying individuals in the semantic space. We should note, however, that such semantic diversity, as well as the need to enforce other constraints, also requires additional computational power. In practice, semantic operators are only approximately competent due to this fact.

Implementing Neutral Mutations

Let us clarify how tree rewriting will serve as the foundation to our proposed neutral mutation operator.

GP evolves trees made of nodes. These represent functions from a set F that are defined for a specific problem domain. Two trees are equivalent if their evaluate into the same fitness value. Let us denote as S_F the set of all possible GP trees, built from F. A **rewrite** (set R) is a function $S_F \to S_F \in R$. We use **transforms** (set T) and **strategies** (set X) in order to design a rewrite.

A **transform** specifies the rewrite to be applied. It is a tuple $(p, r) \in T$ composed of a matching pattern p and a replacement r. It is applied by matching a subset of sub-trees in a given tree s with p, and replacing these matches according to r. Both p and r are built from F with the addition of meta-nodes from a set M: $p, r \in S_{F \cup M}$.

A **strategy** specifies how a transform is applied to a given tree. The simplest strategy is to perform unification of pattern p to some given target site s ($s \in S_F$). Since patterns heavily rely on metavariables (set $M_v \subset M$), the unification process will not only match nodes but also bind metavariables (bindings set b). Therefore, each metavariable is uniquely bound, as defined by the following predicate:

$$Matched := (p = s) \vee$$
$$((\forall m \in M_v \wedge m \leq p \; \exists! v \leq s \wedge (m, v) \in b) \wedge$$
$$p[m \leftarrow v \mid (m, v) \in b] = s)$$

The relation $=$ is the syntactic comparison of two trees (node-by-node equality) while $\leq$ is the subtree relation.

The replacement part r of a transform also contains metanodes. After finding a match, and thus a binding set b, we use the later to substitute the metavariables in r. The resulting tree t replaces the target site s.

$$t = r[m \leftarrow v \mid (m, v) \in b]$$

The above process corresponds to a basic strategy: matching with unification, starting from the root of the tree, and without traversing s. Other strategies may be applied.

A strategy (set X) defines a way to apply one-site unification. It may be combined with other strategies and transforms, and is defined recursively as $2^T \times 2^X \to R \in X$. A strategy is a function that takes a subset of transforms, a subset of other strategies, and returns a rewrite. Applying a strategy to a tree s results in a rewritten tree t. If there are no matches, then $t = s$. In our experiments we considered the following strategies *one-site*, *any-match*, *first-match*, *any*, *all*, *fixpoint*, *n-times-max* and *on*.

- **one-site** strategy ($T \to R$) applies given transform unification to a given tree once, at root point.
- **any-match** $T \to R$ searches for all matches in the target site and selects one match in uniform manner. Selected match is then used in replacement.
- **first-match** $T \to R$ searches for first match of given transform in the target site s in top-down fashion (first we try to unify parent nodes and then children).
- **any** $X^+ \to R$ selects one of the given strategies in uniform manner and applies it to target site s. If application did not produce new tree ($t = s$), selection repeats. As result, $t = s$ if all given strategies did not change s.
- **all** $X^+ \to R$ applies all given strategies to target site.
- **fixpoint** $X^+ \to R$ applies given strategies till moment of no change. Note that some combination of fixpoint with other strategies could result in infinite rewrite.
- **n-times-max** $X \to R$ applies given strategy n times where $n \in [1, max]$, max is a parameter (generally, n-times-max is $N^+ \to X \to R$).
- **on** strategy is complex one. It facilitates analysis of given target site before application of given strategies. Its form is $(C \times X)^+ \to R$ where C is set of cases. One case represents a situation about given target site. Patterns p for transforms could play role cases $\{p \mid (p, r) \in T\} \subset C$. But C also includes conditions on more general analysis of target site s. In our experiment we consider number of present in s nodes by type as one of such analysis.

We characterize transforms based on their impact on the size of the tree on which they are applied; reduction, expansion, or size-neutral (relocation of nodes in the tree).

Reduction transforms (set $T_r \subset T$) reduce the size of the GP tree to which they are applied. This could, potentially, be detrimental to the evolutionary search process. Let us consider a scenario in the Boolean domain, as well as a specific transform: $(not\ not\ x, x)$. If applied with the *any-match* strategy, this transform removes the not-not chain only at one location in the target site. However, if applied with the *fixpoint* strategy, it results in the elimination of all occurring not-not. When using Koza-style GP, which does

not feature mutations, this may complicate the reintroduction of a not node in that individual at a later generation.

On the other hand, reductions may also help the evolutionary search process as they may prevent **bloat**, a well-known problem in GP (Whigham and Dick 2010). Bloat may be controlled by prohibiting the application of an evolutionary operator if the resulting tree's depth would exceed a preset limit. We implemented a similar bloat control technique inside the *any-match* and *first-match* strategies. Matches that would grow the trees beyond the preset limit are simply discarded. When the depth limit is reached, reduction is activated in conjunction with the above-mentioned technique to shrink individuals.

A domain-specific instance of reduction, the *full reduce* strategy ($T_r \rightarrow R$), is particularly interesting and will be detailed in the next section. Its general form is:

$$\text{reduce} = \text{fixpoint}(\{\text{first-match}(a) \mid a \in T_r\})$$

In the above, we use *first-match* instead of *any-match* to optimize the rewrite process. This illustrates that different strategy compositions may have significantly different computational costs: e.g., *any-match* collects all matches then selects one, while *fixpoint* does not require to collect them all.

Expansion strategies increase the size of GP trees. They may use transforms that capture a site s in a metavariable, and insert it in the replacement r. Alternatively, metagenerators may insert the synthesised tree into the matched site. These are present only in the replacement r and produce a unique random subtree per their metavariable name. Lastly, special analysis (case) in *on* strategy could be applied.

Metagenerators do not analyze target sites during synthesis. On other hand, cases of *on* strategy analyze target site s before creating new bindings. We implemented one such generative analysis: $missing_{ERC}$. It collects statistics on ephemeral random constants (ERCs) (Koza 1994) in s and asks the ERC implementation to provide new random value outside the range used in target site. The resulting tree is then bound to metaname (parameter of $missing_{ERC}$ analysis) and combined with r to form the rewritten tree t.

In contrast to reduction strategies, expansions are useful at the start of the evolutionary process when individuals are small. Their application, though, is based on the assumption that the optimal solution requires some specific genetic material. Full expansion introduces all nodes from a problem domain function set F into the target site. But, it is always better to incorporate some problem knowledge in expansion strategies in order not to bloat the trees at start.

Movement strategies move nodes of concrete type up or down the tree. This movement is not free, it requires the application of axioms of the problem domain. For the Boolean domain, movements of *not* nodes usually result in transformations of other nodes on the way. As with expand strategies, these tend to increase breadth and depth of the tree.

Neutral rewrites preserve the semantic. This limits the transforms that can be used to a set $T_n \subset T$, based on problem domain's axioms. Importantly, including/excluding any $t \in T_n$ could bias moves through neutral sets. While it is safer to keep all movements equiprobable inside the neutral sets, this is not practically possible.

Neutral Mutation in the Boolean domain

From the perspective of the GP algorithm configuration, we opted to use the same Boolean functions set for all problems: $F = \{and, or, not\}$. Adding other functions to F, such as the ternary if, would require also expanding the set of transforms.

We use next Boolean domain axioms as transforms in our Neutral Rewrite Operator (NRO): *distributivity* (ruleset A_d), *commutativity* (set A_c), *associativity* (set A_s), *absorption* (set A_{ab}), *complementation* (set A_{cmp}), *identity* (set A_i), *DeMorgan laws* (set A_{DM}). We also consider set A_{basis} that has rules for expressing and through $\{not, or\}$ basis and vise-versa.

Similarly to semantic equivalence of argument permutations, the fact that axiom applicability is two-way states $(p, r) \in T_a \leftrightarrow (r, p) \in T_a$. We accommodate this by introducing additional transforms in strategies which could be trivially obtained from the above-mentioned axioms.

In our experiments, we used the following strategies;

$$T_{rb} = A_{ab} \cup A_{cmp} \cup A_i \cup \{(not\ not\ x, x)\}$$
$$G = \{A_d^*, A_c, A_a^*, A_{ab}, A_{cmp}, A_i, A_{DM}^*\}$$
$$\text{group} = \text{any}(\{\text{any-match}(a) \mid a \in g, g \in G\})$$
$$\text{reduce}_b = \text{fixpoint}(\{\text{first-match}(a) \mid a \in T_{rb}\})$$
$$\text{axioms}_b = \text{n-times-max}(5, \text{any}(\{\text{group}(g) \mid g \in G\}))$$

A^* denotes the fact that the transform set A also contains reverse transforms (replacement r plays role of the pattern). Strategy $axioms_b$ performs from 1 to 5 applications of an inner strategy that rewrites only one match of target site s. Inner strategies are built in such way to uniformly select axiom group and then transform from it.

This framework of strategies and transforms facilitates the construction of potentially complex rewrites. Let us illustrate this with the following manually crafted strategy str. It uses *on* strategy to control what rewrite to apply, depending on the analysis done on the target site s. The argument is a set of tuples: the first element is the case predicate, the second is a child strategy to be applied when the predicate succeeds. The *on* strategy can be seen as a form of advanced switch-statement. In str we have 3 cases.

$$E = \{(x, (n\ or\ not\ n)\ and\ x), (x, (n\ and\ not\ n)\ or\ x)\}$$
$$D = A_{basis}^* \cup A_{DM}^* \cup A_a^* \cup A_d$$
$$\text{depth}(n) = (d(s) >^* n,\ \text{reduce}_\text{b})$$
$$\text{missing}_\text{ERC} = (\text{missingBusBit}(n), \text{expand}_\text{b}))$$
$$\text{expand}_\text{b} = \text{any}(\{\text{any-match}(e) \mid e \in E\})$$
$$\text{default} = (\_, \text{any}(\{\text{any-match}(t) \mid t \in D\}))$$
$$str(n) = \text{on}(\{\text{depth}(n), \text{missing}_\text{ERC}, \text{default}\})$$

Case $depth(n)$ applies $reduce_b$ when depth of target site s is greater than n (we assume that, for this depth, all necessary genetic material is already present and will survive compression). We marked $>^*$ with star to signify that, after reduction, we go to next case so at one step other neutral rewrite could be performed on top of $reduce_b$.

Case $missing_{ERC}$ checks what inputs are absent in the target site s. For selected problems it makes sense that the solution depends on of them. So, in case of the ab-

sence, we apply the expansion strategy. Applied analysis $missingBusBit(n)$ creates a metavariable with name n bound to a missing input and transforms from set E use it.

Eventually, we have the $default$ case with a catch-all predicate. The default strategy contains transforms that perform movement of nodes and two transforms from A_{basis}.

To conclude, str compose rewrites to compress individual, to add missing genetic material, and to transform existing material neutrally. We use it in our proposed NRO.

Goals of experiments

The goal of our experiments[1] is to check if the proposed NRO improves performance. We evaluate this using the average best fitness per generation and the mean of best-of-run fitness. The fitness is defined as the number of misses in output value (1-bit) for a whole truth table (training set) of a concrete Boolean domain problem. Less misses is better, and the correct circuit of a given problem has a fitness of zero.

Our **null-hypothesis** is: NRO does not improve evolution compared to traditional Koza-style GP. The **motivation** of our experiments is to find the setup and problem pairs where an NRO is useful. The alternative hypothesis is that NRO improves the mean of best-of-run fitness.

Methods and parameters

In order to determine the potential of our proposed approach, we apply it to four Boolean domain problems: Multiplexer (MUL), Parity (PAR), Majority (MAJ), Comparator (CMP). These were selected as they are commonly used in the GP literature to compare variants and operators. Each problem considers n-bit inputs and one 1-bit output. We use instances of the above problems with specific values for n: MUL-11, PAR-11, MAJ-11, CMP-12, MUL-20. Since there are 2^{2^n} Boolean problems with 1-bit output and n bits inputs, our problems selection is not representative of the Boolean domain as a whole. However, as we will discuss later, the four selected problems are diversified enough to gain insights about where NRO could be useful.

We apply a Friedman's non-parametric test, which checks for difference in distributions of any pairs of given measurements, regardless of distribution. Additionally, for post-hoc analysis, we use Nemenyi's test, which gives p-values for each pair of measurements for which Friedman's test confirms differences. Our criteria is as follows: we conclude that setup A has better performance ($>$) than B if the mean of best-of-run fitness of A is statistically smaller than B. Here, Friedman's test and Nemenyi's tests show statistical significance when p-value < 0.05, and 0.1 for $>^*$. The relations $>, >^*$ can be seen as domination relations (A dominates B).

Table 1 shows the settings used for all experiments. We used the following evolutionary algorithms:

1. RTsTx – ramped half-and-half population initialization, tournament selection (7), crossover-reproduction 90%-10%. This traditional Koza setup serves as control group.

[1]Code for experiments is hosted at (Gaspar 2021)

Evolution setup	RTsTx, RTsNaTx RTsNTx, RTsTmx
Generations	100
Runs per experiment	30
Population size	1024
Crossover max depth	17
Function set	$\{and,\ or,\ not\}$
Problems	MUL-11/20, PAR-11, MAJ-11, CMP-12
Max match rewrite depth	12
RTsTmx mutation prob	10%
RTsNaTx strategy	$axioms_b$
RTsNTx strategy	$str(10)$

Table 1: Settings

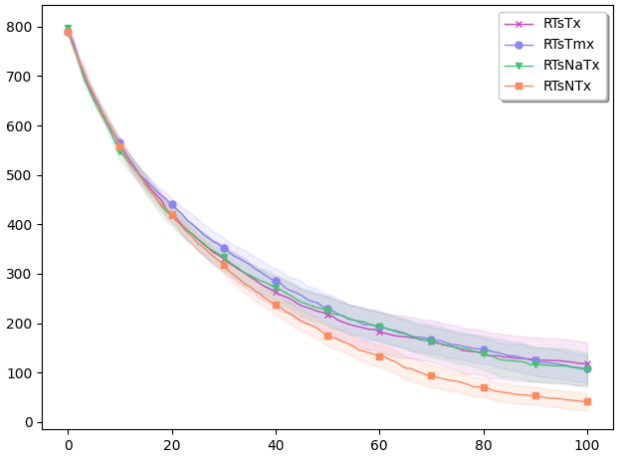

Figure 1: Best fitness and 95% confidence interval, MUL-11

2. RTsNaTx – similar to RTsTx, but with NRO added before crossover. NRO rewrites with $axioms_b$ strategy. Reproduction does not apply NRO before it.
3. RTsNTx – similar to RTsNaTx, but NRO rewrites with $str(10)$. The depth of 10 was chosen arbitrarily without attempts at finding an optimal value for this parameter.
4. RTsTmx – similar to RTsNaTx, but NRO replaced by mutation via random tree replacement at 10% probability.

Results and discussion

Figure 1 demonstrates mean of best of generation fitness value and 95% confidence interval for MUL-11 problem. Table 2 summarizes mean and standard deviation of best-of-run fitness of configurations. Friedman's test shows that at least one setup differs in distribution of best-of-run fitness mean. Nemenyi's test shows that RTsNTx setup has the best performance on MUL-11 compared to classic RTsTx, RTsNTmx (p-value < 0.05) and potentially better than RTsNaTx (p-value < 0.1). For MUL-20, we cannot state that RTsNTx and RTsTmx are statistically significantly different. RTsNTx dominates classic RTsTx and NRO with axioms.

Same setups for PAR-11 problem (figure 2 and table 3) show that RTsNaTx is better than classic RTxTs, RTxTms (p-value < 0.05) while comparison RTsNaTx and RTsNTx did not give statistically significant conclusion. This means that optimal setup of NRO could be problem specific.

Parameter	MUL-11	MUL-20
RTsTx mean ± std	117.7 ± 116.7	255856 ± 24755
RTsTmx mean ± std	108.3 ± 74.2	232104 ± 28457
RTsNaTx mean ± std	106.8 ± 90.4	243680 ± 29362
RTsNTx mean ± std	41.3 ± 45.3	217039 ± 29487
Friedman's p-value	**0.0029**	**5e-7**
Nemenyi's p-values		
RTsTmx vs RTsTx	0.9	**0.001** (>)
RTsNaTx vs RTsTx	0.9	0.351
RTsNTx vs RTsTx	**0.026** (>)	**0.001** (>)
RTsNaTx vs RTsTmx	0.9	0.170
RTsNTx vs RTsTmx	**0.010** (>)	0.409
RTsNTx vs RTsNaTx	0.059* (>)	**0.001** (>)
Conclusion on setup domination by performance		
MUL-11	RTsNTx > RTsTx, RTsTmx	
	RTsNTx >* RTsNaTx	
MUL-20	RTsNTx > RTsTx, RTsNaTx	
	RTsTmx > RTsTx	
> - setup domination for p-value threshold of 0.05		
>* - setup domination for p-value threshold of 0.10		

Table 2: Statistical significance of difference between setups on MUL-11/20

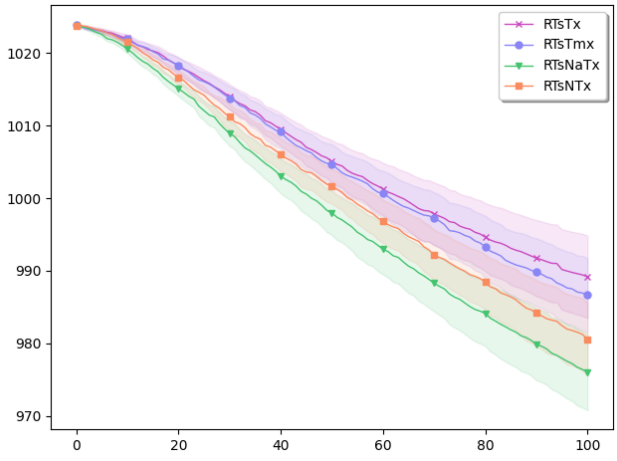

Figure 2: Best fitness and 95% confidence interval, PAR-11

MAJ-11 problem results (figure 3 and table 3) demonstrate a situation when strategy $str(10)$ degrades the performance. With p-value < 0.1 of Nemenyi's test, we say that RTsTx and RTsNaTx have potentially better performance than RTsNTx, though performances of RTsTx and RTsNaTx are not comparable (via domination relation). On this problem NRO (with strategies we tried) shows itself useless.

CMP-11 results can be seen in figure 4 and table 3. On this problem RTsNTx dominates RTsTx and RTsNaTx though it is not statistically comparable to the RTsTmx setup.

Conclusion

The proposed NRO (with $axioms_b$ and str strategies) improved performance on 4 out of 5 Boolean domain problems (tables 2, 3). We still anticipate, as MAJ-11 revealed, that some problems may not benefit from using NRO. This is consistent with the no free lunch theorem: no algorithm could have the best performance on all optimization problems (Wolpert and Macready 1997). More research is

Parameter	PAR-11	MAJ-11	CMP-11
RTsTx mean ± std	989 ± 15	164 ± 14	223 ± 65
RTsTmx mean ± std	987 ± 13	165 ± 13	163 ± 52
RTsNaTx mean ± std	976 ± 14	162 ± 15	204 ± 60
RTsNTx mean ± std	981 ± 13	174 ± 14	140 ± 51
Friedman's p-value	**0.0024**	0.051*	**2e-5**
Nemenyi's p-values			
RTsTmx vs RTsTx	0.9	0.89	0.087* (>)
RTsNaTx vs RTsTx	**0.008** (>)	0.9	0.864
RTsNTx vs RTsTx	0.170	0.098* (<)	**0.001** (>)
RTsNaTx vs RTsTmx	**0.012** (>)	0.779	0.379
RTsNTx vs RTsTmx	0.228	0.379	0.137
RTsNTx vs RTsNaTx	0.639	0.059* (<)	**0.001** (>)
Conclusion on setup domination by performance			
PAR-11	RTsNaTx > RTsTx, RTsTmx		
MAJ-11	RTsTx, RTsNaTx >* RTsNTx		
CMP-11	RTsNTx > RTsNaTx, RTsTx		
	RTsTmx >* RTsTx		
> - setup domination for p-value threshold of 0.05			
>* - setup domination for p-value threshold of 0.10			

Table 3: Statistical significance of difference between setups on PAR-11, MAJ-11, CMP-11

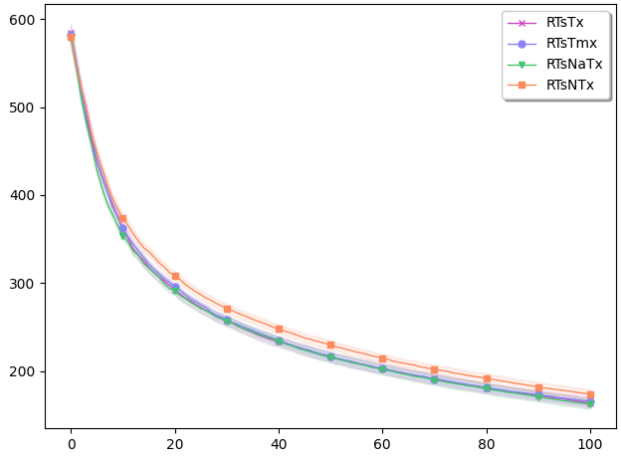

Figure 3: Best fitness and 95% confidence interval, MAJ-11

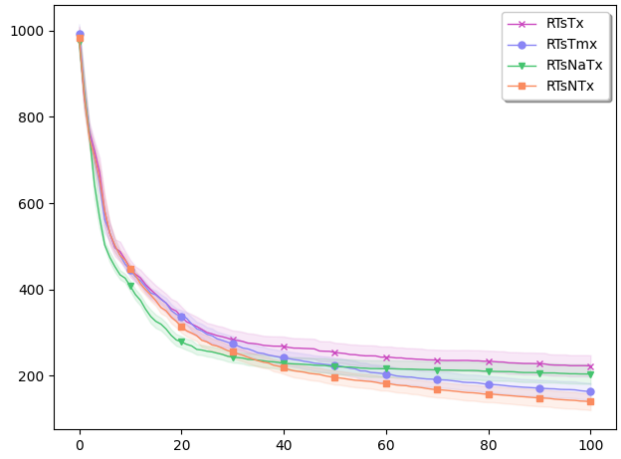

Figure 4: Best fitness and 95% confidence interval, CMP-11

needed to characterize problems that benefit from a NRO.

In this work, we designed NRO with combinations of transforms and strategies. We restricted ourselves to neutral rewrites based on Boolean domain axioms. Future work will also include the study of NRO in the context of other domains, as well as with respect to semantic GP.

References

Collins, M. 2006. Finding needles in haystacks is harder with neutrality. *Genetic Programming and Evolvable Machines* 7(2):131–144.

Galván-López, E., and Rodríguez-Vázquez, K. 2006. The importance of neutral mutations in gp. In Runarsson, T. P.; Beyer, H.-G.; Burke, E.; Merelo-Guervós, J. J.; Whitley, L. D.; and Yao, X., eds., *Parallel Problem Solving from Nature - PPSN IX*, 870–879. Berlin, Heidelberg: Springer Berlin Heidelberg.

Gaspar, A. 2021. Neutral rewrite operator implementation project. https://github.com/cereal-lab/Papers/tree/master/2021-FLAIRS-vitel-NeutralMutationOperatorForBooleanDomain.

Igel, C., and Toussaint, M. 2003. Neutrality and self-adaptation. *Natural Computing* 2(2):117–132.

Kimura, M. 1968. Evolutionary rate at the molecular level. *Nature* 217(5129):624–626.

Koza, J. R. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4(2):87–112.

Mcdermott, J.; White, D.; Luke, S.; Manzoni, L.; Castelli, M.; Vanneschi, L.; Jaśkowski, W.; Krawiec, K.; Harper, R.; De Jong, K.; and O'Reilly, U.-M. 2012. Genetic programming needs better benchmarks. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, 791–798.

Oesch, C., and Maringer, D. 2015. A neutral mutation operator in grammatical evolution. In Angelov, P.; Atanassov, K.; Doukovska, L.; Hadjiski, M.; Jotsov, V.; Kacprzyk, J.; Kasabov, N.; Sotirov, S.; Szmidt, E.; and Zadrożny, S., eds., *Intelligent Systems'2014*, 439–449. Cham: Springer International Publishing.

O'Neill, M., and Ryan, C. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5(4):349–358.

Pawlak, T. P., and Krawiec, K. 2018. Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. *Evolutionary Computation* 26(2):177–212.

Smith, T.; Husbands, P.; and O'Shea, M. 2001. Neutral networks and evolvability with complex genotype-phenotype mapping. In *European Conference on Artificial Life: ECAL2001*, 272–281.

Whigham, P., and Dick, G. 2010. Implicitly controlling bloat in genetic programming. *Evolutionary Computation, IEEE Transactions on* 14:173 – 190.

Wineberg, M., and Oppacher, F. 1996. The benefits of computing with introns. In *Proceedings of the 1st Annual Conference on Genetic Programming*, 410–415. Cambridge, MA, USA: MIT Press.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82.

Yu, T., and Miller, J. 2001. Neutrality and the evolvability of boolean function landscape. In *Fourth European Conference on GP*, volume 2038, 204–217.

Yu, T., and Miller, J. 2002. Finding needles in haystacks is not hard with neutrality. In Foster, J. A.; Lutton, E.; Miller, J.; Ryan, C.; and Tettamanzi, A., eds., *Genetic Programming*, 13–25. Berlin, Heidelberg: Springer Berlin Heidelberg.