Data Mining and Knowledge Discovery (2022) 36:1279–1300
https://doi.org/10.1007/s10618-022-00830-7

Simplification of genetic programs: a literature survey

Noman Javed[1] · Fernand Gobet[1] · Peter Lane[2]

Received: 31 March 2021 / Accepted: 17 March 2022 / Published online: 27 April 2022

Abstract

Genetic programming (GP), a widely used evolutionary computing technique, suffers from bloat—the problem of excessive growth in individuals' sizes. As a result, its ability to efficiently explore complex search spaces reduces. The resulting solutions are less robust and generalisable. Moreover, it is difficult to understand and explain models which contain bloat. This phenomenon is well researched, primarily from the angle of controlling bloat: instead, our focus in this paper is to review the literature from an explainability point of view, by looking at how simplification can make GP models more explainable by reducing their sizes. Simplification is a code editing technique whose primary purpose is to make GP models more explainable. However, it can offer bloat control as an additional benefit when implemented and applied with caution. Researchers have proposed several simplification techniques and adopted various strategies to implement them. We organise the literature along multiple axes to identify the relative strengths and weaknesses of simplification techniques and to identify emerging trends and areas for future exploration. We highlight design and integration challenges and propose several avenues for research. One of them is to consider simplification as a standalone operator, rather than an extension of the standard crossover or mutation operators. Its role is then more clearly complementary to other GP operators, and it can be integrated as an optional feature into an existing GP setup. Another proposed avenue is to explore the lack of utilisation of complexity measures in simplification. So far, size is the most discussed measure, with only

Responsible editor: Martin Atzmueller, Johannes Fürnkranz, Tomáš Kliegr, Ute Schmid.

✉ Noman Javed
n.javed3@lse.ac.uk

Fernand Gobet
f.gobet@lse.ac.uk

Peter Lane
p.c.lane@herts.ac.uk

1 CPNSS, London School of Economics and Political Science, London, UK

2 Department of Computer Science, University of Hertfordshire, Hertfordshire, UK

two pieces of prior work pointing out the benefits of using time as a measure when controlling bloat.

Keywords Simplification · Genetic programming · Bloat control · Explainability

1 Introduction

Machine learning has become increasingly popular, and almost ubiquitous, in many diverse areas. Everyone wants high-performing machine learning models producing accurate results, but not everyone is well-placed to make sense of them. It is not simply a question of explaining the results produced by these machine learning models, but *how* and *why* these results were generated is often even more critical. What is the role of input features? What are the benefits of using a particular knowledge representation? What parameter settings work well? These are all pertinent questions. So, the demand for transparent and efficient machine-learning algorithms is increasing. These two objectives can, unfortunately, be in conflict. The increase in predictive accuracy often comes as a result of the increased complexity of the underlying algorithm and model. This complexity makes explainability a challenging task. On the other side, an increased focus on reducing complexity results in a compromise to accuracy. So, there is usually a trade-off between accuracy and complexity, which researchers are trying to balance.

The increasing, and unprecedented, rate at which new data are generated and their volume motivate the need to use automated data-mining workflows to uncover new knowledge. Although human involvement will always be required, automating part of the knowledge-discovery process offers the potential to come up with better feature selection, model selection, and hyper-parameter settings. One of the issues is determining what is meant by a "better" model. Usually, the objective of these automated systems is to find highly efficient models in terms of accuracy, but, as discussed above, sometimes "better" may mean more explainable. An interesting question is to decide if these automated systems play their part in addressing the accuracy-complexity trade-off (Freitas 2019).

We can observe two clear trends in artificial intelligence research. The first one is automation and the second one is explainable artificial intelligence (XAI). Genetic Programming (GP), being an inherently white-box approach, stands a clear chance to deliver on both of these lines. It has been used to automate machine learning pipelines (Olson and Moore 2016). Moreover, it has also been employed to generate explanations for machine learning models (Cavaliere et al. 2020; Hu 2020). One can pose a simple question at this point: how explainable are GP-generated models themselves? The question is a bit vague, as GP is an enabling technology that can generate many kinds of models, ranging from decision trees to neural networks and many others. However, it makes more sense if one thinks of it in terms of understanding of solutions within a specific category. For example, while solving a classification problem using decision trees, how much does GP prefer simpler decision trees over complex ones?

The question of generating explainable models is related to a well-known issue with GP called *bloating*, where continued evolution of a population leads to an excessive increase in size in the generated programs without improving their overall fitness. Bloated individuals require extra computational resources, such as memory. Also, the unwanted code in a bloated program increases the computational time to evaluate its fitness, thus making the whole evolutionary process slow. This unwanted growth renders individuals less explainable. The good news is that this phenomenon has been understood by the GP community from the very beginning and many techniques have been proposed over the years to tackle the problem. All bloat control techniques at least partially address the issue of explainability because of their focus on controlling the size of individual programs. One particular kind of bloat control technique, called *simplification*, involves editing the program. It works by converting an individual program to a smaller-sized variant with, usually, identical behaviour. This technique relies on a set of rules or methods for editing a program and thus can help understand the composition of the individual. It does not differentiate individuals on the basis of size and so individuals both large and small can be modified to make them (even) smaller. In this way, simplification not only controls the bloat of large programs, but also works to reduce the size of all candidate solutions, and in general, smaller solutions are easier to understand.

In this paper we review the literature of simplification of GP models with the following objectives:

1. To differentiate simplification from other bloat control techniques in terms of its potential for explainability;
2. To categorise the literature in terms of the major simplification techniques;
3. To identify the important design and integration constraints and organise the existing approaches in terms of these design considerations.

The next section introduces GP and highlights its usage for data mining and knowledge discovery. Section 3 discusses complexity and simplification. Section 4 deals with the first objective, by highlighting the major differences between simplification and other bloat control techniques. Section 5 reviews the techniques proposed by researchers. Section 6 presents certain considerations to properly design and integrate simplification into the genetic process and organises the literature in terms of these decisions. Section 7 discusses several opportunities and challenges for simplification. In the final section, we conclude and present some future directions for research.

2 Genetic programming and knowledge discovery

Genetic programming (Koza 1992) (GP) is a widely used form of Evolutionary Computation (EC). Like other EC methods, it is inspired by the Darwinian process of natural selection in which individuals with a better fit to the environment have more survival chances. Its originality is to evolve entire programs. Programs are represented as combinations of user-defined operators. Each program is a candidate solution and new solutions are generated by gradually evolving a population of programs. Each program is evaluated against a user-defined fitness criterion and those with higher fitness are used to create the next generation using a crossover operator, which splits and recombines two programs into two new children. Mutation is applied to some of the individuals, usually with a very low probability, in order to generate variation. This process is repeated for a number of generations until a predefined stopping criterion is met. There are a number of varieties of GP, including tree-based, linear (Brameier and Banzhaf 2010) and graph-based GP (Banzhaf et al. 1998). In this paper, we cover only tree-based GP as it is one of the most widely used representations. Moreover, the identification and removal of non-effective code in linear GP are quite straightforward.

Since its inception, people from different domains responded well to GP and developed various applications. Koza, the founder of GP, tracked the growth of GP and compiled a list of GP applications producing human-competitive results (Koza 2010). He noted that the solutions proposed by GP, although belonging to different domains, share some common characteristics. Moreover, he correlated the increase in usage and efficacy of GP with the growth in computational power. Based on this correlation, he predicted an increased flow of human-competitive results in the future. True to his prediction, we can witness GP venturing in diverse areas and producing impressive results (Krawiec 2015; Acharya et al. 2020; Azzali et al. 2020; Bi et al. 2020). This effectiveness is not just limited to single-objective problems. Many researchers reported supportive evidence for solving multi-objective optimisation problems (Zhang and Rockett 2006; Coelho et al. 2011; Zupančič et al. 2020).

Being a white-box technique, GP can deliver on the promise of XAI (Howard and Edwards 2018). It can not only create interpretable models but can also unlock the behaviour of black-box models. Several researchers have applied GP for this purpose. Genetic programming explainer (GPX) creates local explanation models to fit the data. The data here represents the sample set containing the point of interest plus the newly created noise set in the neighbourhood (Ferreira et al. 2020). Virgolin et al. created a meta-learning system (Virgolin et al. 2020). They first developed an ML model of proxies of human interpretability (PHI) using the data gathered as human feedback. The feedback was about the relationship between mathematical formulas and two forms of interpretability. These two forms were simulatability and decomposability. The model is then plugged into GP with one of the objectives representing human interpretability. The other was the mean square error representing the performance for symbolic regression problems. In another approach (Evans et al. 2019), a group of researchers used multi-objective genetic programming to mimic the working of black-box ML models. They created a decision tree to guess the predictions of the black-box while keeping it as simple as possible. This creates a model-agnostic approach independent of the type of the black-box.

Data mining refers to the extraction of knowledge from data. The two essential requirements of a data mining system are its predictive accuracy and how easy its output is to interpret. These are critical requirements for many data mining algorithms. Satisfying these requirements may increase the utility of the algorithm, especially for the domains where transparency and interpretability are of utmost importance. The requirement of transparency is not only restricted to the early phases of the knowledge discovery process. It is also needed for the post-processing of the discovered models. However, at times it becomes difficult to satisfy both these requirements at the same time. Maximising one may result in the loss of the other. In these situations, data mining lends itself as a multi-objective optimisation problem. GP, being a viable option to solve such optimisation problems, has been used to solve and automate different parts of the data mining system (Pappa and Freitas 2009). Freitas (2003) conducted a detailed survey and noted the use of GP for comprehensible rules discovery and attribute selection. Krawiec (2002) employed GP to construct a new set of features based on the original attributes to improve the classification performance. GP represents these features as Lisp-like expressions, thus exposing the algorithmic transformation of input attributes to new features. In a recent paper (Lensen et al. 2020), authors used multi-objective genetic programming for dimensionality reduction of a dataset. Since there is a possibility of a decline in quality while reducing dimensions, the use of multi-objective GP offers different trade-offs between quality and dimensionality. Moreover, the generated mappings are transparent and interpretable. Thus, a GP-driven data mining system, offering a nice trade-off between accuracy and transparency, can solve classification problems, the discovery of if-else rules, the discovery of association rules, and many other problems (Collet and Wong 2012).

3 Complexity and simplification

Before beginning a review of the simplification literature and related issues, we first provide a definition. However, to define simplification, we must understand complexity. In GP, complexity usually refers to the complexity of an individual. There exist multiple ways in which an individual in GP can be represented. Tree-based GP opts for the tree representation of the individual while linear-GP operates on a sequence of instructions. Graph-based GP uses graphs to represent the individual while grammar-based GP separates the genotype from phenotype and defines a mapping process from genotype to phenotype. In this paper, we are restricting ourselves to tree-based GP because it is one of the most widely used representations and is prone to the phenomenon of bloat, for which simplification is often proposed as a remedy.

There are several ways to calculate the complexity of an individual when it is represented as a tree. Structural complexity refers to the size of the tree, which can mean one of several things: the number of nodes in a tree, the depth of a tree (the number of levels it contains), or the sum of the nodes of all the subtrees of a tree. In most of the simplification schemes reviewed in this paper, the number of nodes in a tree is considered as the measure of the complexity. Only two articles proposed execution time as a measure of complexity. Since one of the objectives of our work is to classify the simplification literature, we consider the tree size as the proxy of complexity. We shall get back to other definitions of complexity later in the discussion section.

Given the above-mentioned definition of complexity, simplification can now be defined as a process of reducing tree size. This reduction may be in the form of deleting some nodes or replacing a subtree with a smaller subtree. So, simplification aims to reduce the complexity of the individual by reducing its size and hence making it more understandable. Since the process involves structural changes, the impact of these modifications may or may not alter the semantics as well. So, for the sake of clarity, we propose the following two types of simplification: ***Exact simplification***: It preserves the semantics of the individual while removing the unwanted parts. ***Approximate simplification***: It is more flexible than the exact simplification, hence compromises the semantics while modifying the structure of the individual.

4 Simplification and bloat control

As mentioned earlier, bloat is one of the biggest challenges to the successful use of GP. The problems caused by bloat manifest themselves in various forms, including a poor exploration of search space due to lack of diversity, lack of robustness, lack of generalisation-ability and lack of understandability. Researchers of GP have been well aware of these problems from the very start and have proposed several bloat-control techniques. We refer to the survey papers (Silva et al. 2012; Alfaro-Cid et al. 2010) for a detailed discussion of bloat and its control techniques. Our objective in this section is twofold: the first is to highlight the fundamental difference between simplification and other bloat-control techniques; and the second is to justify the existence of simplification as more than merely a bloat-control technique.

The easiest to implement and most widely used bloat-control technique puts a limit on the maximum depth of the individual (Koza 1992). The problems with this technique are how to identify the boundary threshold and what to do if good solutions lie on the other side. To address this, researchers have developed alternative ways of controlling the size. One way assigns poor fitness to relatively big individuals, thus limiting their chances of survival in later generations. An alternative way to achieve the same goal uses a waiting room, where newly created individuals have to wait before becoming part of the population. The waiting time is proportional to the individuals' size. Yet another way is to penalise big individuals by reducing their chance of appearing in the next generation. Rather than forcing direct or indirect limits, another way to control bloat is by designing size-aware genetic operators of crossover, mutation, selection and evaluation. These intelligently crafted operators are biased towards smaller sized individuals, thus limiting the number of big individuals and hence limiting their chances of survival.

Dignum and Poli (2008) proposed a population-based technique, operator equalisation, to control bloat, and Silva et al. (2012) added some of its variants. The focus is shifted from dealing with a single individual to dealing with the distribution of individuals. Individuals are grouped on the basis of sizes in different buckets. The capacities of these buckets create the distribution of the population and every newly generated population is supposed to maintain this distribution. Several variants of this technique have been proposed, from fixed-bucket capacities to adaptable ones. The idea is to keep checks on the count of individuals of certain sizes. Hence, instead of defining a threshold limiting size, this technique creates various thresholds and maintains the counts within those boundaries.

The common theme in all of the techniques listed so far is the idea of restraining the growth of individuals while respecting their structural integrity. Code editing or simplification differs fundamentally from the other bloat control techniques by violating this principle, thus making changes to the syntactic structures of the individuals by removing or replacing certain parts. In this regard, it resembles the optimisation phase of a compiler. Compilers make multiple passes over the intermediate code to optimise its behaviour by converting the existing form to a reduced and more resource-efficient form (Aho et al. 1986). In this way, simplification can be viewed as an optimisation technique, transforming an individual to its optimised variant. However, this transformation is usually destructive and the resultant individual may be significantly different from the original one, particularly at a syntactic level. Unlike in a compiler, the semantics usually undergoes some change, and the simplified code will only be an approximation to the original.

By following the principle of Occam's Razor, simplification tends to generate smaller-sized variants of complex individuals. This inherent property enables it to tackle the issue of interpretability and understandability in machine learning. Simplified models, being smaller in size, are (usually) more explainable as compared to their larger-sized counterparts. This issue of explainability is now receiving considerable attention from AI researchers. One of the main reasons is that some fields of study, such as psychology and neuroscience, are reluctant to trust black-box solutions. AI has to offer white-box and transparent solutions to win their trust.

An added advantage offered by simplification is increasing the possibility of generating robust and generalisable individuals (Helmuth et al. 2017). It does so by pruning the unnecessary parts of the individuals. How these excess genes appear in the first place can be explained by 'the fitness causes bloat' theory (Langdon 1998). According to this theory, the size of the individuals gradually increases to fit the corner cases of the training data. This fine-tuning leads to overfitting the training data, which results in the loss of generalisability and robustness. Hence, simplification by pruning the unwanted parts of the code keeps overfitting under check, thus boosting the chances of robustness and generalisability. On a side note, pruning is not specific to GP. It is frequently used in other tree-based approaches, such as decision trees, random forests, and boosted trees.

Controlling bloat is one of the advantages of simplification when it runs throughout the evolutionary process. Simplification justifies its existence even without being used as a bloat-control technique. It can work in offline mode on one or several GP-generated solutions. When applied in this way, it exhibits itself purely to achieve the objective of explainability. So, in light of these fundamental differences between bloat control and the potential to generate robust and interpretable solutions, we propose to consider simplification separately from bloat-control mechanisms.

5 Review of simplification techniques

In the following paragraphs, we review various simplification techniques and group them under different categories. Some of these techniques fall under the category of exact simplification as they do not change the semantics, whereas others do interfere with the semantics and hence lie under the umbrella of approximate simplification.

5.1 Removal of Dormant nodes

The simplest form of simplification is the identification and removal of dormant code. By dormant code, we mean the code that is unreachable while executing an individual. Another form of unreachability is where code is executed but has no contribution to the generation of the final output. One example of unreachable code is any statement written after the *return* statement in *C*++, *Java*, and many other programming languages. Song et al. (2010) calculated the contribution of nodes at the time of evaluating fitness. They identify nodes as non-contributing nodes if they make no impact on the state of execution. They remove these nodes before selecting the individual as a parent. Jackson (2010) identified dormant nodes by creating an identical tree and marking all nodes as non-visited. While evaluating the fitness cases, the status of these nodes is changed from non-visited to visited when they are executed. Those having non-visited status at the end of evaluation are considered as dormant nodes. A dormant node implies that the subtree rooted at that node is also dormant. So, all these dormant nodes can be removed without any performance loss. All of these techniques are semantic preserving, and hence belong to the category of exact simplification.

5.2 Rule-based simplification

This category refers to approaches inspired by the algebraic simplification of expressions. For instance, an expression like $A * 0$ can be simplified by replacing it with 0. Similarly, consider the following *if* statement:

```
(if (> a 0) b b)
```

In the lisp code above, no matter what the value of a is the result will be b. So, the whole if expression can be replaced by b. It is highly unlikely that a human would write such code, but in genetic programming, where the operators are selected randomly for growing individuals, or as a result of crossover or mutation, such a situation may arise. Rule-based simplification works by defining a rule set, where a rule can be replaced by a smaller variant with the same semantics. Wong and Zhang (2006) present a bottom-up greedy approach to replace a subexpression with a smaller expression that is algebraically equivalent. The authors used hashing to find whether two different-looking expressions are similar or not. Their idea is to extend the application of simplification to dissimilar-looking individuals. They applied simplification to every individual. However, it is recommended to use simplification every two generations, otherwise it will be too slow. The authors performed a building-block analysis using numerical nodes as the building blocks. While simplification disrupts the building blocks generated in non-simplified GP, it has the potential to generate new building blocks (Wong and Zhang 2007). Murano et al. (2018) applied algebraic simplification to multimodal genetic programming and reports an improvement in search-ability without compromising performance. Burlacu et al. (2019) used hashing to identify tree isomorphisms and hence create new opportunities to apply algebraic simplification. Borcheninov and Okulovsky (2012) suggested internalising the application of algebraic rules by making it part of the evolutionary operators. Rule-based simplification works by matching the rules, and since these rules are pre-defined by domain experts, it belongs to the category of exact simplification.

5.3 Numerical simplification

Rule-based simplification suffers from various limitations. One of them is the definition of a rule set, which requires domain experts with an excellent knowledge of the application area. Another limitation is the application order of the rules. In case more than one rule matches the precondition and can be fired, which one will be preferred? This choice may lead to two completely different outputs. Yet another limitation is the reliance on a hash function to generate the same hashes for different-looking expressions. Chances are high that many simplification opportunities can be missed. To overcome these limitations, researchers propose calculating the contribution of a node while generating the output of its parent node. If the contribution is negligible, the tree rooted at that child node can be completely removed, thus shrinking the size of the individual.

```
c = a + b where a = 3 * 10^8 and b = 0.000001
```

In the example above, the contribution of b in the value of c is extremely small and can be ignored, at least in the majority of cases. To further complicate the situation, imagine b is not just a single node but a complex calculation rooted in a big subtree. Now, with numerical simplification, you can safely remove this subtree without impacting the final result. Algebraic simplification can never find this kind of opportunity. Kinzett et al. (2008) use an algorithm which calculates the numerical contribution of a node on its parents based on the evaluation of training examples. If the impact is less than a threshold, the subtree rooted at the child node can be deleted. A node can also be replaced by a constant if the difference between its maximum and minimum values is below a threshold. The authors conducted several other studies to analyse the building blocks and propose several minor variants of the approach (Kinzett et al. 2009b, a, 2010). The same set of authors, in another approach, proposed to split the simplification process into two phases: proposers and evaluators. Proposers propose a simplified tree by extending the option of applying algebraic rules. Algebraic rules are approximated by calculating the possibility of predicting a parent node through its child node using linear regression. Evaluators calculate the mean squared error between the simplified and the original tree and accept the proposal if the value is below a certain threshold (Johnston et al. 2009, 2010). Rockett (2020) proposed a pruning technique based on statistical permutation tests. A pruning is proposed based on the contribution of a subtree. The proposal is evaluated using a permutation test to study the effect of pruning at the parental level. A validation set is used to corroborate the proposal.

5.4 Fitness-based simplification

This category refers to approaches where a portion of an individual's tree is removed or replaced with a constant. The fitnesses of the original tree and the modified tree are then compared and the best one is retained and undergoes further simplification if required. These fitness-based approaches are very similar to the application of a destructive mutation operator. However, they have to make sure that the individual remains syntactically valid after the removal of one or more branches. This destructive simplification often comes at the cost of changes in semantics, so we categorise it under approximate simplification. Alfaro-Cid et al. (2010) proposed a random pruning approach in which some branches of a tree are pruned and replaced by a terminal. The pruned branches are then planted as separate trees. They reported a significant reduction in mean tree sizes while maintaining the quality of fitness. In a similar study, Spector and Helmuth (2014) define pruning as an iterative process which removes one or more instructions from the Push program and then compares the fitness of the pruned program with the fitness of the original program. Nguyen et al. (2016) applied simplification to surrogate-assisted GP in a top-down way. If the subtree has more than one node, replace it with one; otherwise, if it is a constant, replace it with zero. If the modified tree has better fitness, retain it; otherwise, carry on the simplification process to the next levels of the tree. Helmuth et al. (2017) adopted a similar approach and defined various simplification mechanisms in PushGP, but at the genotype level. One of their simplification operators silences a gene, which hence will not be expressed in the phenotype. Another option is to "NOOP" a gene, that is to replace it with a NOOP instruction which, upon execution, will do nothing. The authors of the paper proposed pruning as an operator (Javed and Gobet 2021). The idea was to split a parent into its children and replace it with the child having the highest fitness.

5.5 Semantics-based simplification

Semantics-based approaches extend the fitness-based approaches by trying to keep the semantics of the original tree and the modified tree the same. They achieve this by approximating the semantics of the pruned subtree with the replacement subtree. This process is computationally costly, and hence these approaches are usually slow. Naoki et al. (2009) proposed a semantics-based approach which consists of replacing subtrees with an equivalent but shorter tree picked from a simple set. The semantics equivalence is calculated based on the similarity in the output vector on certain regression points. Chu and Nguyen (2017) propose to replace a randomly selected subtree with a terminal of approximate semantics. They did that by growing a tree, using the selection terminal, of approximate semantics. The idea is that the newly grown tree will be shorter in size than the original one. They extended this technique by using an approximate subtree rather than a terminal. That approximate subtree is picked from a library of trees and then a population is grown to match the approximate semantics (Chu et al. 2018). The same authors, Nguyen and Chu (2020), propose another technique in which a subtree is replaced by a tree of desired semantics rather than the approximate semantics of the original tree.

5.6 Population-level simplification

Population-wide simplification is inherently different from all the other simplification approaches. Rather than working on single individuals, it tries to simplify the whole population at once. The authors of this paper proposed generation-wide simplification (Gws) (Javed and Gobet 2021). In Gws, after every k^{th} generation, the new population is created by rupturing the individuals rather than by using the routine genetic operators of crossover and mutation. Rupturing of an individual produces many offspring, where every child is a member of the power set of the parent individual. Every child is stored only once to maintain uniqueness in the population and keep the redundancy under check. Although different individuals may produce children with the same genetic makeup, the uniqueness constraint ensures that no two individuals are genetically identical in the new generation. The resulting population may exceed the population size, in which case, only the fittest individuals are retained. If the number of offspring produced is less than the population size, new members are added following the routine genetic operations of crossover and mutation. Population-wide simplification replaces a parent with one of the fittest children formed by rupturing the parent generation. The semantics of these children may differ from that of the parents: as it does not guarantee to preserve the semantics, it belongs to the category of approximate simplification.

6 Considerations for the design and integration of GP simplification

The previous section presented a review of various simplification algorithms proposed by GP researchers. The reader will have noticed the subtle variations in these algorithms. These variations raise several questions that researchers need to consider while implementing and integrating simplification into the evolutionary process. These questions are critical design decisions that a researcher needs to make. For example, using simplification as a part of the evolutionary process or using it in an offline mode is a critical consideration. It has significant consequences that we shall discuss later in the relevant subsection. Similarly, the choice of syntax versus semantics-based simplification may alter the course of the evolutionary process if used in an online mode. There are several other decisions that we shall cover in the subsequent subsections.

The choice of simplification technique is also crucial for a user of genetic programming. It has consequences beyond the cost in terms of time and resources. Different simplification schemes may lead to a different set of evolved solutions. These differences originate because the simplification algorithm may change the search process and leads to exploration in distinct areas of the search space. So, this section serves an additional purpose of facilitating the end-user to make better-informed decisions.

The following subsections present an organisation of the literature in terms of the design choices and their associated consequences.

Table 1 Offline versus online

Paper	Offline	Online
Naoki et al. (2009)	✓	
Spector and Helmuth (2014)	✓	
Nguyen et al. (2016)	✓	
Helmuth et al. (2017)	✓	
Howard and Edwards (2018)	✓	
Wong and Zhang (2006)		✓
Wong and Zhang (2007)		✓
Kinzett et al. (2008)		✓
Johnston et al. (2009)		✓
Kinzett et al. (2009a)		✓
Johnston et al. (2010)		✓
Kinzett et al. (2010)		✓
Song et al. (2010)		✓
Jackson (2010)		✓
Alfaro-Cid et al. (2010)		✓
Silva et al. (2012)		✓
Chu and Nguyen (2017)		✓
Murano et al. (2018)		✓
Chu et al. (2018)		✓
Burlacu et al. (2019)		✓
Nguyen and Chu (2020)		✓

6.1 Offline versus online simplification

Koza used to edit individuals after a successful run. Later on, researchers started developing simplification algorithms to simplify automatically at the end of an evolutionary run. This type of simplification is called *offline simplification*. It refers to the process of selecting a few best individuals at the end of a run and simplifies them to generate equally performing but shorter individuals. This type of simplification was common in the early days of GP. But gradually, researchers started using simplification as part of the evolutionary run to tackle the bloat phenomenon. So, the idea is to keep in check the growth of the individuals as the evolution progresses. It was primarily because of this bias towards bloat control that the focus of simplification has shifted from its original purpose, which was making individuals more understandable. Thus, most of the studies present the idea of *online simplification*, except the earlier works of Koza and the ones listed in Table 1.

The choice between offline and online becomes the first decision for an end-user or researcher. Now, this decision depends on the objective for which one wants to employ simplification. If the aim is to make final solutions more understandable without worrying about bloat, then one should go for the offline mode. However, if bloat is a concern, and one wants to use simplification to keep it in check, online simplifi-

Table 2 Individual selection

Simplification	All generations	After k generations
Probabilistic	Alfaro-Cid et al. (2010)	
	Borcheninov and Okulovsky (2012)	
Whole population		Wong and Zhang (2006)
		Wong and Zhang (2007)
	Silva et al. (2012)	Kinzett et al. (2008)
		Johnston et al. (2009)
		Kinzett et al. (2009a)
	Burlacu et al. (2019)	Johnston et al. (2010)
		Kinzett et al. (2010)
		Howard and Edwards (2018)
Top k%	Chu and Nguyen (2017)	
	Chu et al. (2018)	
	Nguyen and Chu (2020)	

cation can be of rescue. But one has to pay some extra price in terms of the increase in execution time and computational resources.

6.2 Criteria of individual selection for simplification

Simplification is a costly process and needs time and extra computational resources such as memory and processing cycles. However, it is not much of a concern when used offline as it has to deal with few final solutions only. However, the online mode requires careful considerations to make the process more feasible. For example, it makes sense to simplify only a part of the population rather than all individuals of every generation.

One needs to ask two questions: First, whether to use simplification in every generation; second, how many individuals to simplify in a generation, and how to select those individuals. To answer the latter question, we arrange the simplification literature into three categories:

- Probabilistic: A certain number of individuals, selected randomly, will undergo simplification.
- All individuals: The whole population will experience simplification
- Top $k\%$ individuals: Only the $k\%$ individuals with the highest fitness values will be selected for simplification

Table 2 presents this taxonomy. To save cost, very few researchers use simplification in every generation. The issue with probabilistic selection is that it may choose individuals that do not need to be simplified. The same goes for the top $k\%$ selection, as being fittest does not necessarily mean that simplification is needed. Another problem with top $k\%$ is the choice of appropriate value of k. A more balanced approach is to use simplification after every few generations, and select individuals not just based on fitness but also based on their sizes.

6.3 Syntactic, numerical and semantic simplification

Another decision to make is the selection of the type of simplification. By type, we mean simplification based on syntax or semantics. Syntax-based simplification, as the name suggests, selects part of an individual for replacement or removal by matching it against a pre-specified set of rules as explained in Sect. 5.2. Since domain experts design these rules, they guarantee to preserve the semantics of the expression. So, it belongs to the category of exact simplification. The problem with syntax-based simplification is that it fails to capture the cases where individuals with two different genetic makeup yield the same output. It is also not possible to prune the parts of an individual that are not contributing to the final result.

To overcome the problems of algebraic simplification, researchers came up with the idea of numerical simplification. This identifies the contribution of children in the parent and prunes the non-contributing parts. Because of its nature, there is no need to involve domain experts in creating algebraic rules. However, defining the nodes' contribution thresholds is not straightforward and depends on the nature of the application. But once it is set, the removal of the subtrees does not impact the semantics of the individual. So, numerical simplification also belongs to the Exact simplification category.

Both syntactic and numerical simplifications fail to consider the possibility that two syntactically different individuals may yield the same output. Semantic simplification, as explained in Sect. 3, creates further opportunities for simplifying individuals. It does so at the cost of compromising on the semantics and thus belongs to the category of approximate simplification. The limitation of semantic approaches is that the semantics are also problem-dependent. Moreover, finding the semantic equivalent subtree of a smaller size is a time-consuming task.

It is evident from the above discussion that choosing between one of these options is not an easy choice. It often depends on the application and requires the involvement of the domain expert. Table 3 categorises different papers as syntactic, numerical and semantic-based simplification.

6.4 Generalisation-ability of simplification

We define generalisation-ability as the possibility of applying the same simplification techniques to a variety of problems belonging to different domains. Although many simplification schemes exist in the literature, most of them are implemented and tested on only a specific category of applications. Some of them are designed to work on regression problems, while others can simplify the individuals solving classification problems. Without tweaking, it is not possible to scale them and use them on another category of problems.

By contrast, there exist some simplification techniques that can work on a variety of problems. These approaches are domain and problem-independent. So, is

Table 3 Syntactic, numerical and semantic simplification

Paper	Syntactic	Numerical	Semantic
Wong and Zhang (2006)	✓		
Wong and Zhang (2007)	✓		
Borcheninov and Okulovsky (2012)	✓		
Murano et al. (2018)	✓		
Burlacu et al. (2019)	✓		
Kinzett et al. (2008)		✓	
Johnston et al. (2009)		✓	
Kinzett et al. (2009a)		✓	
Johnston et al. (2010)		✓	
Kinzett et al. (2010)		✓	
Naoki et al. (2009)			✓
Helmuth et al. (2017)			✓
Chu and Nguyen (2017)			✓
Howard and Edwards (2018)			✓
Chu et al. (2018)			✓
Nguyen and Chu (2020)			✓
Rockett (2020)			✓

generalisation-ability a desirable trait for a simplification algorithm? The performance of problem-specific approaches is usually better than the generic ones. They can create more opportunities for simplification. However, if the goal is to provide a technique that can work on multiple categories of problems without any need for intervention from the domain experts, researchers can prefer more generic approaches.

Table 4 organises the papers in terms of domain specific and domain independent approaches.

7 Discussion

Genetic Programming equipped with simplification offers several opportunities. GP-generated solutions are inherently understandable as they are generated using a combination of user-defined operators. Since these operators are implemented by programmers and are known to domain experts, it is easier to make sense of their combinations. The generated solutions are like programs and the problem of understanding them is similar to understanding what a program is doing. However, without simplification mechanisms in place, this understanding can still be challenging because of the presence of unwanted code. After reviewing the literature and organising it in terms of the design and integration constraints, we observed the potential of simplification and the advantages it can bring to GP. We also observed the adverse effects of simplification on the evolutionary search. Several of the approaches listed in previous sections may lead towards a local optimum and loss in diversity. For this reason, several researchers have opted to apply simplification after a few generations or only

Table 4 Generalisation-ability

Paper	Domain specific	Domain independent
Wong and Zhang (2006)	✓	
Wong and Zhang (2007)	✓	
Kinzett et al. (2008)	✓	
Kinzett et al. (2009b)	✓	
Johnston et al. (2009)	✓	
Kinzett et al. (2009a)	✓	
Naoki et al. (2009)	✓	
Johnston et al. (2010)	✓	
Kinzett et al. (2010)	✓	
Song et al. (2010)	✓	
Borcheninov and Okulovsky (2012)	✓	
Chu and Nguyen (2017)	✓	
Chu et al. (2018)	✓	
Howard and Edwards (2018)	✓	
Murano et al. (2018)	✓	
Burlacu et al. (2019)	✓	
Nguyen and Chu (2020)	✓	
Rockett (2020)	✓	
Jackson (2010)		✓
Alfaro-Cid et al. (2010)		✓
Silva et al. (2012)		✓
Spector and Helmuth (2014)		✓
Nguyen et al. (2016)		✓
Helmuth et al. (2017)		✓

on a portion of the population. The following subsections present the potential of GP simplification and our observations on what further research should focus on.

7.1 Mutation/control operators versus simplification operators

The evolutionary algorithm contains and supports a variety of operators. The core of the algorithm uses mutation and crossover operators to construct new individuals. In this section, we consider how simplification can be incorporated both into these existing operators and as a separate operator: our contention is that simplification can usefully be considered as a *separate*, standalone operator.

In the early days, when researchers started using simplification as a bloat control technique, evolutionary operators were defined and implemented to incorporate aspects of simplification. Ekart (1999) proposed a mutation operator to identify non-functional bits of code—the introns—and remove them. She achieved this by defining algebraic simplification rules and applying them to expressions that can be replaced by a simpler one. This operator guarantees to preserve the semantics and fitness of the original expression. Araujo (2004) proposed a cut operator that randomly cuts a subtree, using a higher probability for individuals larger in length. Although she did not label it as a mutation operator, the cut operator can be thought of as such because of its random application—the percentage of occurrence can be controlled by the user. Belle and Ackley (2002) introduced a Uniform Subtree Mutation with the idea that a tree will undergo mutations proportional to its size, following a binomial distribution. Size-fair mutation works like a normal mutation but ensures that the distribution of sizes before and after the mutation is maintained. Thus, in general, the average sizes of the individuals will remain the same after the mutation.

Mutation was not the only way researchers have tried to implement simplification. Kinnear (1993) proposed hoist—a modified version of single crossover—in which a copy of a randomly selected subtree is obtained from an individual. Following the philosophy of size-fair mutation, size-fair crossover randomly selects a crossover point in the first parent and then calculates the size of the subtree rooted at that point (Langdon 2000). The crossover point in the second parent is then chosen so that the size of the subtree rooted at this point is the same as the size of the first parent's subtree. In this way, it ensures that the size of children will not increase after applying the crossover operator.

Simplification can be implemented using other genetic operators, as some of the cases listed above illustrate. However, there are several drawbacks associated with this approach:

- Mutation and crossover are inherently random in nature and must preserve this randomness as it is a fundamental property of GP stochastic search. By contrast, simplification is akin to tuning rather than random modification.
- The primary purpose of mutation and crossover is to produce new individuals. By contrast, the purpose of simplification is to optimise an individual: these two operator types can be considered as complementary, where the primary role of mutation and crossover is the random exploration and exploitation of the search space, and the role of simplification is more similar to a local search.
- These operators have no say after the end of an evolutionary run. By contrast, this is the most important time to analyse the individuals for simplification to a shorter, but equally performant, variant.

Although crossover and mutation operators can be tweaked to act like simplification, this approach does not produce a neat and modular design or implementation. Hence, because of these fundamental differences between simplification and other genetic operators, we argue that simplification should be implemented as a standalone operator and not interfere with the natural flow of GP. In this regard, it will act as a plug-and-play feature of GP working in tandem with other operators.

7.2 Potential for creative AI

Creative Artificial Intelligence (CAI) attempts to develop human-like solutions for areas in which good solutions are lacking. CAI poses two main challenges: first, the complexity of the search space and, second, the requirement for explainable solutions. GP, unlike many machine learning approaches, generates interpretable solutions. These solutions can often be trimmed to simpler versions; these simplification procedures usually make the solutions easier to understand. So, GP equipped with simplification can soften the challenge of explainability. The first challenge of complexity stems from the fact that CAI's search space is enormous and high dimensional, with many local optima (Miikkulainen 2021). GP can handle this because of its population-based nature, in which multiple searches are launched at the same time rather than focusing on just one. GP starts these searches from different random starting points and allows them to progress in parallel, which offers better opportunities to avoid local minima. These challenges posed by CAI are also prevalent in the process of creating automated data mining and knowledge discovery systems.

7.3 Consideration of other measures of complexity

To be a feasible approach, simplification-equipped GP needs to address several challenges. One such challenge is the appropriate definition of complexity. The issue of complexity has already been addressed in several studies, but often in the context of model selection and the generalisation of performance on unseen data. These studies defined several types of complexities ranging from genotypic complexity to phenotypic complexity and statistical-based measures. A number of complexity measures could be used when simplifying GP programs, such as Minimum Description Length (MDL), Kolmogorov Complexity, Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC), to mention just a few. Le et al. (2016) provide a useful review of complexity measures in the framework of GP.

Beyond the complexity of GP programs in a formal sense, one must also consider complexity as experienced by humans when trying to understand GP programs. A substantial amount of literature in psychology and cognitive science indicates that at least three factors affect complexity: (a) cognitive factors (e.g., short-term memory capacity, learning rate, speed of processing information) (Simon 1989; Gobet et al. 2011; Hunt 2011); (b) the level of expertise and domain knowledge (Gobet 2016; Shadbolt and O'Hara 1997), which in our case also includes knowledge of the GP operators used and the type of programs generated; and (c) extrinsic factors (e.g., length of programs, presence of recursion, and type of representations used) (Kotovsky and Simon 1990; Weinberg 1998). Ai et al. (2021) explore such ideas with respect to simple two-person games. Their empirical results suggest that machine-learning explanations are helpful only when their level of complexity is appropriate: explanations should not provide too much information and in particular not more information than the explicit description of the solution to a problem. A suitable level of abstraction is needed to satisfy these desiderata. Whether their conclusions hold with GP programs is an open and important question.

8 Conclusion and future work

This paper has reviewed the literature on the simplification of genetic programs from several perspectives and has identified various trends. One such trend is the shift in the primary objective of simplification. The work on simplification initially started with the focus on understanding the GP-generated models. Gradually, this focus drifted towards controlling bloat. There is now a need for reverting this drift because of the emerging requirement for explainable solutions. Another prominent trend is the shift from syntax-based approaches to semantic ones. Semantic-based techniques offer the advantage of identifying syntactically different individuals with similar semantics. Hence, they can outperform syntactic ones by identifying more simplification possibilities. However, the challenge is on the implementation side of these semantic-based approaches because they can be computationally demanding.

Integrating simplification into an existing GP setup can be challenging. It can adversely affect evolutionary progress. We found that several of these challenges revolve around making simplification a computationally viable option. Some of the points that need careful consideration are the criteria for the selection of individuals for simplification, the scale of application of the simplification algorithm, and the parallelization of this algorithm. The first one deals with how to choose individuals to undergo simplification. Not all individuals are complex enough to be simplified, and including too many individuals increases the cost of simplification. The second consideration is the scale of applicability, which includes questions such as whether simplification should be a part of the evolutionary run, or whether it should be applied only after a run. Another question with online simplification is finding the threshold beyond which simplification hurts the process of evolution. The last consideration is the development of parallel versions of simplification algorithms to make them more efficient.

While organising the literature on simplification, we identified several potential avenues of research that deserve further attention. Most of the simplification literature assumes size as a measure of an individual's complexity. Only two of the studies used execution time as a measure and developed time-based simplification. Their experiments revealed the potential of time-based simplification and its potential to deal with bloat. Several other complexity measures exist and are used in genetic programming but have not been used to date in simplification algorithms. Thus, exploring other measures of complexity and measuring the impact on bloat and the explainability of solutions is an area to explore further. Another less explored area is the development of domain-independent simplification techniques. In most cases either the proposed solutions are problem-specific, or no evidence is furnished of their applicability to other domains. We suggest exploring further the generalisation-ability of simplification algorithms. Furthermore, there is a rich relationship between simplification and compiler optimisation which we have highlighted here and plan to investigate in future.

Acknowledgements This article is part of the project "Genetically Evolving Models in Science (GEMS)" that has received funding from the European Research Council (ERC) under the grant agreement no. ERC-2018-ADG-835002.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Acharya D, Goel S, Asthana R, Bhardwaj A (2020) A novel fitness function in genetic programming to handle unbalanced emotion recognition data. Pattern Recognit Lett 133:272–279
Aho AV, Sethi R, Ullman JD (1986) Compilers, principles, techniques, and tools. Addison-Wesley
Ai L, Muggleton SH, Hocquette C, Gromowski M, Schmid U (2021) Beneficial and harmful explanatory machine learning. Mach Learn 6:66
Alfaro-Cid E, Merelo J, de Vega FF, Esparcia-Alcázar AI, Sharman K (2010) Bloat control operators and diversity in genetic programming: a comparative study. Evol Comput 18(2):305–332
Araujo L (2004) Genetic programming for natural language parsing. In: European conference on genetic programming (EuroGP), pp 230–239
Azzali I, Vanneschi L, Mosca A, Bertolotti L, Giacobini M (2020) Towards the use of genetic programming in the ecological modelling of mosquito population dynamics. Genet Program Evol Mach 21(4):629–642
Banzhaf W, Francone FD, Keller RE, Nordin P (1998) Genetic programming: an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann, San Francisco
Belle TV, Ackley DH (2002) Uniform subtree mutation. In: European conference on genetic programming (EuroGP), pp 152–161
Bi Y, Xue B, Zhang M (2020) Genetic programming-based feature learning for facial expression classification. In: 2020 IEEE congress on evolutionary computation (CEC), pp 1–8
Borcheninov Y, Okulovsky Y (2012) Internal and online simplification in genetic programming: an experimental comparison. In: Proceedings of the Spring/Summer young researchers' colloquium on software engineering
Brameier MF, Banzhaf W (2010) Linear genetic programming, 1st edn. Springer, Berlin
Burlacu B, Kammerer L, Affenzeller M, Kronberger G (2019) Hash-based tree similarity and simplification in genetic programming for symbolic regression. In: International conference on computer aided systems theory, pp 361–369
Cavaliere F, Della Cioppa A, Marcelli A, Parziale A, Senatore R (2020) Parkinson's disease diagnosis: towards grammar-based explainable artificial intelligence. In: 2020 IEEE symposium on computers and communications (ISCC), pp 1–6
Chu TH, Nguyen QU (2017) Reducing code bloat in genetic programming based on subtree substituting technique. In: 2017 21st Asia Pacific symposium on intelligent and evolutionary systems (IES), pp 25–30
Chu TH, Nguyen QU, Cao VL (2018) Semantics based substituting technique for reducing code bloat in genetic programming. In: Proceedings of the ninth international symposium on information and communication technology, pp 77–83
Coelho ALV, Fernandes E, Faceli K (2011) Multi-objective design of hierarchical consensus functions for clustering ensembles via genetic programming. Decisi Support Syst 514:794–809
Collet P, Wong ML (2012) Evolutionary algorithms for data mining. Genet Program Evol Mach 13(1):69–70
Dignum S, Poli R (2008) Operator equalisation and bloat free GP. In: European conference on genetic programming (EuroGP), pp 110–121

Ekart A (1999) Shorter fitness preserving genetic programs. In: European conference on artificial evolution, pp 73–83

Evans BP, Xue B, Zhang M (2019) What's inside the black-box? A genetic programming method for interpreting complex machine learning models. In: Proceedings of the genetic and evolutionary computation conference (GECCO), pp 1012–1020

Ferreira LA, Guimarães FG, Silva R (2020) Applying genetic programming to improve interpretability in machine learning models. In: 2020 IEEE congress on evolutionary computation (CEC), pp 1–8

Freitas AA (2003) A survey of evolutionary algorithms for data mining and knowledge discovery. In: Advances in evolutionary computing. Springer, pp 819–845

Freitas AA (2019) Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In: International cross-domain conference for machine learning and knowledge extraction, pp 48–66

Gobet F (2016) Understanding expertise: a multi-disciplinary approach. Palgrave, London

Gobet F, Chassy P, Bilalić M (2011) Foundations of cognitive psychology. McGraw Hill, London

Helmuth T, McPhee NF, Pantridge E, Spector L (2017) Improving generalization of evolved programs through automatic simplification. In: Proceedings of the genetic and evolutionary computation conference (GECCO), pp 937–944

Howard D, Edwards MA (2018) Explainable ai: the promise of genetic programming multi-run subtree encapsulation. In: 2018 international conference on machine learning and data engineering (iCMLDE), pp 158–159

Hu T (2020) Can genetic programming perform explainable machine learning for bioinformatics? In: Genetic programming theory and practice XVII. Springer, pp 63–77

Hunt E (2011) Human intelligence. Cambridge University Press, Cambridge

Jackson D (2010) The identification and exploitation of dormancy in genetic programming. Genet Program Evol Mach 11(1):89–121

Javed N, Gobet F (2021) On-the-fly simplification of genetic programming models. In: Proceedings of the 36th annual ACM symposium on applied computing, pp 464–471

Johnston M, Liddle T, Zhang M (2009) A linear regression approach to numerical simplification in tree-based genetic programming (Tech. Rep.). New Zealand: Research report 09-7, School of Mathematics Statistics and Operations Research, Victoria University of Wellington

Johnston M, Liddle T, Zhang M (2010) A relaxed approach to simplification in genetic programming. In: European conference on genetic programming (EuroGP), pp 110–121

Kinnear KE (1993) Evolving a sort: Lessons in genetic programming. In: IEEE international conference on neural networks, pp 881–888

Kinzett D, Zhang M, Johnston M (2008) Using numerical simplification to control bloat in genetic programming. In: Asia-Pacific conference on simulated evolution and learning, pp 493–502

Kinzett D, Johnston M, Zhang M (2009a) How online simplification affects building blocks in genetic programming. In: Proceedings of the 11th annual conference on genetic and evolutionary computation (GECCO), pp 979–986

Kinzett D, Johnston M, Zhang M (2009) Numerical simplification for bloat control and analysis of building blocks in genetic programming. Evol Intell 2(4):151–168

Kinzett D, Zhang M, Johnston M (2010) Analysis of building blocks with numerical simplification in genetic programming. In: European conference on genetic programming (EuroGP), pp 289–300

Kotovsky K, Simon HA (1990) What makes some problems really hard: explorations in the problem space of difficulty. Cognit Psychol 22:143–183

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Koza JR (2010) Human-competitive results produced by genetic programming. Genet Program Evol Mach 11(3–4):251–284

Krawiec K (2002) Genetic programming-based construction of features for machine learning and knowledge discovery tasks. Genet Program Evol Mach 34:329–343

Krawiec K (2015) Behavioral program synthesis with genetic programming, 1st edn. Springer, Berlin

Langdon WB (1998) Evolution of size in variable length representations. In: Proceedings of the IEEE conference on evolutionary computation (ICEC), pp 633–638

Langdon WB (2000) Size fair and homologous tree crossovers for tree genetic programming. Genet Program Evol Mach 1(1):95–119

Le N, Xuan HN, Brabazon A, Thi TP (2016) Complexity measures in genetic programming learning: a brief review. In: 2016 IEEE congress on evolutionary computation (CEC), pp 2409–2416

Lensen A, Zhang M, Xue B (2020) Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. Genet Program Evol Mach 21(3):399–431

Miikkulainen R (2021) Creative AI through evolutionary computation: principles and examples. SN Comput Sci 2(3):1–7

Murano K, Yoshida S, Harada T, Thawonmas R (2018) A study on multimodal genetic programming introducing program simplification. In: 2018 Joint 10th international conference on soft computing and intelligent systems (SCIS) and 19th international symposium on advanced intelligent systems (ISIS), pp 109–114

Naoki M, McKay B, Xuan N, Daryl E, Takeuchi S (2009) A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. In: International work-conference on artificial neural networks, pp 171–178

Nguyen QU, Chu TH (2020) Semantic approximation for reducing code bloat in genetic programming. Swarm Evol Comput 58:100729

Nguyen S, Zhang M, Tan KC (2016) Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans Cybernet 47(9):2951–2965

Olson RS, Moore JH (2016) TPOT: a tree-based pipeline optimization tool for automating machine learning. In: Workshop on automatic machine learning, pp 66–74

Pappa GL, Freitas AA (2009) Automating the design of data mining algorithms: an evolutionary computation approach, 1st edn. Springer, Berlin

Rockett P (2020) Pruning of genetic programming trees using permutation tests. Evol Intell 13(4):649–661

Shadbolt N, O'Hara K (1997) Model-based expert systems and the explanation of expertise. In: Expertise in context: human and machine, pp 315–337

Silva S, Dignum S, Vanneschi L (2012) Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genet Program Evol Mach 13(2):197–238

Simon HA (1989) Models of thought, vol II. Yale University Press, New Haven

Song A, Chen D, Zhang M (2010) Contribution based bloat control in genetic programming. In: 2010 IEEE congress on evolutionary computation (CEC), pp 1–8

Spector L, Helmuth T (2014) Effective simplification of evolved push programs using a simple, stochastic hill-climber. In: Proceedings of the companion publication of the 2014 annual conference on genetic and evolutionary computation (GECCO), pp 147–148

Virgolin M, De Lorenzo A, Medvet E, Randone F (2020) Learning a formula of interpretability to learn interpretable formulas. In: International conference on parallel problem solving from nature (PPSN), pp 79–93

Weinberg G (1998) The psychology of computer programming. Dorset House Pub

Wong P, Zhang M (2006) Algebraic simplification of GP programs during evolution. In: Proceedings of the 8th annual conference on genetic and evolutionary computation (GECCO), pp 927–934

Wong P, Zhang M (2007) Effects of program simplification on simple building blocks in genetic programming. In: 2007 IEEE congress on evolutionary computation (CEC), pp 1570–1577

Zhang Y, Rockett PI (2006) Feature extraction using multi-objective genetic programming. In: Multi-objective machine learning. Springer, pp 75–99

Zupančič J, Filipič B, Gams M (2020) Genetic-programming-based multi-objective optimization of strategies for home energy-management systems. Energy203C