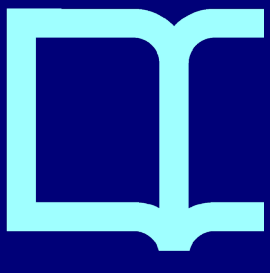


Noname manuscript No.
(will be inserted by the editor)

Promoting Social Diversity for the Automated Learning of Complex MDE Artefacts

Edouard R. Batot · Houari Sahraoui

Received: date / Accepted: date

Abstract Software modelling activities typically involve a tedious and time-consuming effort by specially trained personnel. This lack of automation hampers the adoption of Model Driven Engineering (MDE). Nevertheless, in the recent years, much research work has been dedicated to learn executable MDE artifacts instead of writing them manually. In this context, mono- and multi-objective Genetic Programming (GP) has proven being an efficient and reliable method to derive automation knowledge by using, as training data, a set of examples representing the expected behavior of an artifact. Generally, conformance to the training example set is the main objective to lead the learning process. Yet, single fitness peak, or local optima deadlock, a common challenge in GP, hinders the application of GP to MDE. In this paper, we propose a strategy to promote populations' social diversity during the GP learning process. We evaluate our approach with an empirical study featuring the case of learning well-formedness rules in MDE with a multi-objective genetic programming algorithm. Our evaluation shows that integration of social diversity leads to more efficient search, faster convergence, and more generalizable results. Moreover, when the social diversity is used as crowding distance, this convergence is uniform through a hundred of runs despite the probabilistic nature of GP. It also shows that genotypic diversity strategies cannot achieve comparable results.

Edouard R. Batot
IN3-SOM - Universitat Oberta de Catalunya
E-mail: ebatot@uoc.edu

Houari Sahraoui
Diro-Geodes - Universite de Montreal
E-mail: sahraouh@iro.umontreal.ca

1 Introduction

Model Driven Engineering (MDE) aims at raising the level of abstraction of programming languages. MDE advocates the use of models as first-class artifacts. It combines domain-specific modeling languages to capture specific aspects of the solution, and transformation engines and generators in order to move back and forth between models while ensuring their coherence, or to produce from these models low level artifacts such as source code, documentation, and test suites [73]. Still, designing and developing artifacts able to perform automated tasks in MDE (ensuring the well formedness of models, transforming models, detecting defects and applying refactoring operations, etc.) requires one to have both knowledge in the targeted domain as well as in the design and development tools. If done manually, these activities typically involve a tedious and time-consuming effort by specially trained personnel. Such a lack of automation is considered by many MDE specialists as a threat to MDE adoption [76, 88].

Yet, in recent years, many research contributions have shown that it is feasible to automatically learn how to perform a task through examples, or by analogy to similar, previously solved tasks [85, 52, 36, 64]. Many approaches to learning use Genetic Programming (GP) to ease the burden of hand-programming growing volumes of increasingly complex information. Empirical studies have shown a high potential for automatically learning MDE structured artifacts such as model transformations [52, 69, 5] and model well-formedness rules [36, 8] from examples of task's inputs/outputs. An *example* here must be understood as a couple `<input model; expected output>` defining the constraints that bind artifacts' output to input. The set of training examples represents the expected behavior of the artifact to learn and thus constitutes a convenient objective to lead the search of a solution.

An efficient technique to learn how to automate MDE tasks from examples is Genetic Programming (GP) [56], and more specifically multi-objective GP (MOGP). MOGP consists in evolving a population of programs by the process of natural selection to find the set of programs that better satisfy given functional (conformance to the examples) and non-functional objectives. The selection of the final program with a particular combination of objectives' values is thus postponed until a time when it is known what combinations exist. Studies have shown the value of such techniques and their suitability to real problems. However, from the very beginning, authors pointed out two major drawbacks to the application of GP and MOGP to learning problems: (i) diversity of populations is difficult to maintain during evolution, and populations tend to gather around a *single fitness peak*; and, (ii) individuals tend to grow unnecessarily in size – also called *bloating* effect.

Both bloating and single fitness peak symptoms have been well investigated for more than two decades, and valuable research directions were explored [15, 77, 60]. Nevertheless, while adapting MOGP as an automatic process to learn MDE artefacts (*e.g.,* well-formedness rules, model transformations) from examples, we encountered these same scenarios in a great amount of runs. Solutions agree on finding the correct outputs for a large number of examples, but fail all on a few same examples – a *single fitness peak* is reached. Evolutionary computation of MDE artifacts seems to favor solutions with a high fitness, *i.e.*, a high percentage of correct output found, at the expense of the diversity of the solutions.

On promoting diversity, Vanneschi *et al.* [84] show in their work the superior importance of research harnessing the imbalance between diversity and convergence [58, 16] with indirect semantic methods that *"act on the syntax of the individuals and rely on survival criteria to indirectly promote a semantic behavior"*. Inasmuch as semantics are considered in GP as a vector of examples, MDE artefact learning from examples methodology offers an auspicious support for such investigations with a built-in evaluation of *semantic fitness* [1].

In a previous paper [9], we introduced a new Social Diversity measure of individuals, inspired from information retrieval research field. This measure consists in giving more chances to programs that solve examples not properly solved by their congeners despite their limited global performance on the example set. We implemented this diversity strategy to improve the execution of a genetic programming version of the well-established multi-objective genetic algorithm NSGA-II [32]. We also validated it by comparing the resulting accuracy and convergence with one phenotypic diversity strategy using a crowding distance as defined in NSGA-II. Although, our evaluation produced compelling evidence about the performance of the social diversity strategy, we did not properly evaluate it with genotypic diversity strategies that are frequently used in GP and MOGP. In this paper, we extend our previous work by defining two genotypic alternative strategies for evaluation purpose. To conduct a fair evaluation, we considered both problem-independent and dependent genotypic strategies. We compare the impact of the Social Diversity strategy to the genotypic alternatives through an empirical study featuring the problem of automatic learning of well-formedness rules from examples and counter examples.

The contributions of this paper with respect to our previous publication [9] are as follows:

- A better illustration of the social diversity strategy with an example of arithmetic expressions.
- A definition of two genotypic diversity strategies, one problem-independent and one problem-dependent. These strategies are used for evaluation purpose but can be used as alternatives to or in combination with our social diversity strategy.
- An empirical study to compare the four diversity strategies: social, phenotypic, domain-independent genotypic, and domain-dependent genotypic.

The rest of this paper is organized as follows. The following section gives the background and discusses the difficulty of learning complex artifacts from example with genetic programming, through the problem of well-formedness rule learning. Section 3 details how employing Social Diversity fosters efficiency and accuracy of a GP run. We first depict an illustrative example and then introduce our strategy. In Section 4, we explain how MOGP can be used for MDE artifact learning through the case of WFR, and how the social diversity can be integrated in the learning process. The definition and implementation of the two genotypic diversity strategies are given in Section 5. In Section 6, we assess our contribution through an empirical evaluation and discuss results obtained with the different diversity strategies. Section 7 discusses the related work with respect to our contribution. Section 8 concludes on the usability and relevance of a Social Diversity applied to a methodology for evolutionary learning from examples.

2 Background and Problem: Learning complex artifacts with Genetic Programming

In this section, we start by introducing a case that illustrates well the complexity of learning complex artifacts in MDE. Then we briefly present the principle of Genetic Programming (GP) and we discuss its main limitations.

[1] In the remainder of this paper, *semantic fitness* refers to the level of conformance to training examples an individual satisfies.

2.1 Learning Complex Artifacts – an Illustrative Case

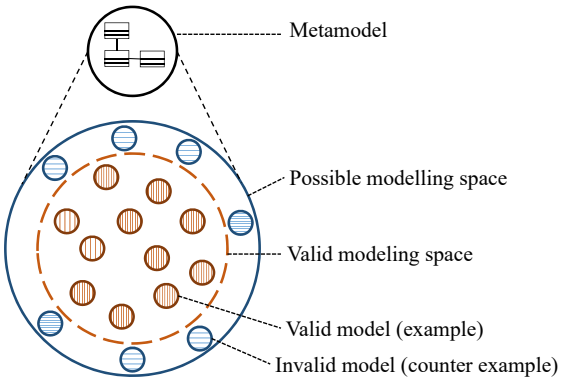


Fig. 1: Metamodel, modelling space and application domain.

MDE artifacts are complex structures both in terms of syntax and semantic. Writing them requires expertise in software engineering as well as in the application domain the elements represent. It is then an error-prone and time consuming process. Alternatively, automatically learning them from examples brings an important added value from the perspectives of effort and quality. But, this is far from being trivial.

As an example, consider the case of well-formedness rules (WFRs). Due to their high level of abstraction, metamodels usually define too-large modelling spaces. They must be enriched with constraints, or rules, limiting the scope of their possible instantiations, *i.e.,* well-formed models in contrast to ill-formed models. Fig. 1, borrowed from [20], schematizes the idea of possible vs valid modeling spaces: a metamodel defines a modelling space (within blue line) ; of which only a sub-space is valid (within red dashed line). A set of WFRs allows to automatically differentiate between valid (well-formed) and invalid (ill-formed) models – it formally describes the limit of a specific domain modeling space.

Cadavid et al. analyzed more than 400 metamodels and their WFRs [21]. They found that the majority of WFRs can be expressed as a combination of instances of a limited set of OCL constraint patterns. We view then a WFR as a complex structure represented as a tree whose nodes are logical operators (*AND*, *OR*, *IMPLIES*, and *NOT*) and first-order quantifiers (*forAll* and *exists*), and whose leaves are building blocks in the form of *OCL patterns* instances. Consequently, a set of WFRs is a tree with as root a vector whose elements are pointers to the individual WFR trees. Fig. 2 shows an example of a an arbitrary set of 3 WFRs for the state-machine metamodel. The first and second rules constrain a *final* state to have respectively one incoming transition and no outgoing transition. The third rule requires that a pseudostate *choice* must have at least one incoming or outgoing transition. As for their execution, the *defacto* language is the Object Constraint Language (OCL[2]).

Writing OCL constraints is a burden [22]. Learning them from examples and counter examples of models is also difficult. OCL expressiveness allows for an infinite number of syntactical variations. To find the set of WFRs that is able to distinguish between examples of well-formed and ill-formed models by an exhaustive exploration is out of consideration. This complexity can be addressed by reducing the solution space and by using a learning strategy that efficiently explores this space. The space reduction can be achieved by considering only OCL constraint patterns and their combinations rather than all possible constraints that can be written in OCL for the targeted metamodel. Gray box in Fig. 2 shows the application of one OCL pattern. The context and the parameters are variation points that can be modified during the learning process as proposed by Cadavid *et al.* [20]. The learning strategy must be guided by the conformance to the examples (semantic guidance). Moreover, since solutions must be legible by the final user (*i.e.,* within human reach), the size of constraints to learn must be kept as small as possible. Learning WFRs from examples is then defined as a multi-objective optimization problem.

This massive multi-dimensional problem space - and the arborescent nature of WFR representations - makes GP a valuable methodology to explore the potential individual solutions. The goal here is to find the ”near” optimal solutions that exhibit the best trade-off between the example conformity and legibility. Beyond the WRFs case, the same reasoning holds for the other MDE artifacts (e.g., model transformations). Learning other artefacts from examples are similar problems in terms of dependency towards the examples and diversity issue. Model transformation [55, 5], design defect [52], and refactoring rules [64], are the best examples.

We believe that the idea to consider the social dimension of individuals' characteristics shall apply to the evolutionary computation of these other complex artefacts. Since fitness lies on examples' comparison as well, *inverse example resolution frequency* can be used in the exact same manner. We prospect, as future work, to replicate this study with other artefacts.

2.2 Genetic Programming

Genetic programming is an optimization evolutionary technique that consists in automatically creating a program from a set of examples of inputs/outputs given as a specification.

[2] http://www.omg.org/spec/OCL/

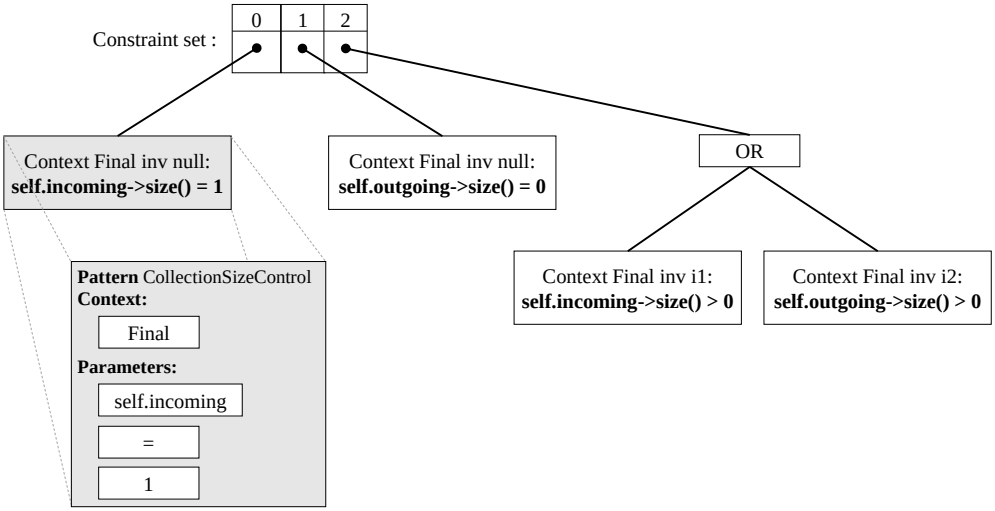


Fig. 2: An example of solution containing 3 WFRs.

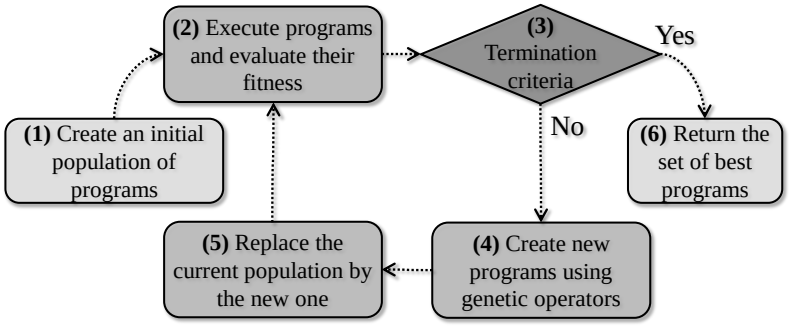


Fig. 3: A typical genetic programming cycle

The process starts by creating a set of programs, usually created randomly, and then evolves those programs guided by the provided example. Such a process is depicted in Fig. 3. At the beginning, an initial population of programs is created (1). Then, every program of the population is executed on the example inputs, and its fitness is evaluated. To this end, in general two or many objectives compete. As a first objective, the semantic fitness is measured by comparing the produced outputs with the expected ones, i.e., the example outputs. The other objectives are usually related to non-functional aspects of the sought program. For instance, an objective can be the minimization of the size of the individuals to avoid unnecessary big programs (2). If a termination criterion is reached (3), the set of best or near-optimal solution programs is returned (6). Otherwise, a new population of programs is created using genetic operations (crossover and mutation) applied on selected potential reproducers (4). The new population replaces the previous one and all individuals are replaced, no matter their fitness (5), and a new iteration starts (2). The loop is repeated until a termination criterion is reached (commonly, a perfect fitness, or an arbitrary number of iterations). We will illustrate in great details in Section 4 how much a tangible support GP, and more precisely multi-objective GP, is used to learn MDE artefacts such as WFRs automatically from examples and counter examples.

Although GP allows to find good solutions for many problems, it is known to suffer from two main issues, namely *bloating* and *single fitness peak*. In the remainder of this section, we briefly discuss the *bloating* issue, and then focus in details on the *single fitness peak* issue. The latter illustrates the consequences of imbalance between the diversity of individuals among a population and the convergence of the algorithm toward the *best* solutions, which is the main aspiration of this paper.

2.2.1 Bloating

Luke *et al.* [60] suggest that, from a high level perspective, *bloating* (or *code growth*) happens because adding genetic material to individuals is *more* positively correlated to the fitness than removing material. They define bloat as the ”uncontrolled growth of the average size of an individual in the population”. Numerous countermeasures have been pro-

posed to tackle the effects of bloating, offering to present readers a few options to choose from [77]. More precisely, in a multi-objective context, Pareto-based Multi-objective Parsimony Pressure (*i.e.,* using an objective devoted to constraining size of individuals) has been found very effective – with limited side effects [34]. We use this technique in our experiments.

2.2.2 Single fitness peak

The second problem with GP is the risk of a *single fitness peak* [29], consisting in a premature convergence together with a loss of diversity. Candidate solutions get stuck in a local optima and often no further improvement in fitness is noticed [91]. To tackle this issue, the level of population diversity must be given due consideration during a GP run [32, 17]. More precisely, two phases of such a run are appropriate to tackle diversity loss: at the initial population creation, to ensure a broad genetic material base[3] ; and/or during the evolution itself, to ensure that diversity does not fall from one generation to the next.

In both cases, diversity exists in two kinds: a *genotypic diversity* measures structural variance between individuals, whereas a *phenotypic diversity* focuses on the behavior of individuals. To control diversity within the population during the search, authors show that, if phenotypic diversity produces better results, genotypic diversity also improves the results when compared to standard executions (*i.e.,* with no diversity control) [19, 84].

Genotypic Diversity Genotypic diversity relates to the variability of individuals' characteristics among a population with regards to their structure. It is a measure of the distance between individuals at the syntax level [62, 63]. However, there is no consensual definition of genotypic or syntactic diversity for MDE since the syntax of its artifacts is very complex [11, 17, 18]. To ensure sufficient diversity within the initial population, teams have used different metrics based on coverage estimations and showed interesting results [51, 12]. Coverage refers here to the rate of instanciation of language elements. The more distinct elements an instance or a group of instance use, the more it/they *covers* the language definition (*i.e.,* the metamodel). Works vary in nature and offer automatic generation of *diverse* models [8, 90], or provide a user with visual assistance to eliciting learning inputs data [38, 71, 59]. In any case, both techniques can be employed to provide with diverse initial population of solutions as well as with qualified input data. Recent work on the nature of software artifacts mention a relative *naturalness of the source code* that allows one to apply NLP techniques to software engineering problems [49].

[3] For an in-depth discussion on the challenging long run investigation on generating a set of diverse models, see Varro *et al.* [86]

Phenotypic Diversity In contrast to genotypic diversity, phenotypic diversity is measured on the behavior of a program – independently to its syntax. In evolutionary learning, a phenotypic (or semantic [84]) measure, refers generally to the proportion of examples correctly processed by a program (*i.e.,* producing the expected output when executed on a specific input). It is a tangible fact that phenotypic diversity is more efficient than genotypic diversity to avoid the single fitness peak problem and thus to accentuate convergence [84]. Nonetheless, if some early studies went as far as to expand the Darwinian metaphor and considered preference between individuals during a GP run [68], to the best of our knowledge, there exists no study explicitly measuring benefits of phenotypic diversity on the specific case of evolutionary learning of MDE artifacts.

Indirect Semantic Diversity Methods Roughly speaking, Indirect Semantic Diversity methods combine both genotypic and phenotypic diversities' benefits. The rationale behind these methods lies in their ability to distinguish between the aim of the method: to produce individuals with strong semantic fitness, and the mean of its application: genetic modifications performed on their syntax [26]. Vanneschi *et al.* demonstrate the power of indirect diversity methods in genetic programming and call for more research in this field [84]. Note that learning artifacts from examples in MDE is based on the conformity of candidate solutions with the provided examples, which carry the semantics of the sought artifacts.

3 Social Diversity for Multi-Objective Genetic Programming

Notwithstanding a significant amount of work on diversity issues in the Evolutionary Computation community, as we will develop in Section 7, single fitness peaks occur during evolutionary learning of MDE-artefacts. Processing most examples correctly, *alpha* individuals [15] spread among populations though they struggle to solve all examples exhaustively. These individuals, are called *alphas* since they tend to spread their genetic material to the entire population (see illustration in Fig. 5). This leads, on the one hand, to a disproportionate number of solutions with good fitness, at the expense of the overall semantic fitness a population may achieve. On the other hand, unfortunately, solutions able to solve the remaining corner cases reach a (much) lower fitness. Withal, since reproducers are chosen with regard to their fitness, the genetic material these latter *partial solutions* convey is lost and *corner cases* are never solved.

In this section, we first introduce a simplified example that illustrates the idea behind our social diversity without the cognitive cost of a full blown showcase. This reduces the accidental complexity and shall help the reader better grasps the idea of the approach and its concrete application to MDE

	a	b	c
Ex. 1	2	3	7
Ex. 2	2	4	8
Ex. 3	2	5	9
Ex. 4	2	1	5

(a) f illustrated with 4 examples

	a	b	c
Ex. 1	2	3	7
Ex. 2	2	4	8
Ex. 3	2	5	9
Ex. 4	2	1	5
Ex. 5	3	3	12

(b) f' illustrated with 5 examples

Table 1: The semantic to learn is embedded in a set of examples: the *semantic fitness* is the ratio of examples a function solves.

in Section 4. Finally, we explain our main contribution, *i.e.,* a social semantic diversity measure.

3.1 Illustrative Example

To illustrate our words through a simple example, consider the problem that consists in finding an arithmetic expression defined by a set of examples. The arithmetic expression takes as input a couple of integers and produces as output one integer. It is thus in the form $f(a, b) = c$, where f is an arithmetic function, and a, b, and c are integers. A possible set of examples that can be used to learn the arithmetic expression is shown in Table 1a. Each row in the table represents an example, *i.e.,* a tuple of integers representing input couples with their expected output. Accordingly, the fitness of a candidate solution is the ratio of examples this candidate solves: a perfect solution will solve them all. A well-defined representation of arithmetic functions, together with genetic programming methodology, will learn relevant solution configurations that solve the example set (when doable).

```
Solution0: f0(a,b) = 2a² - b
Solution1: f1(a,b) =  7a - b
Solution2: f2(a,b) =  -a + b²
Solution3: f3(a,b) =  2a + b
```

Fig. 4: Examples of candidate solutions

In practice, a search-based approach to learning (*e.g.,* genetic programming) will easily find a function that solves examples from Table 1a: for instance, as shown in Fig. 4, solution $f_3(a, b) = 2a + b$. This *easy-to-find* candidate will show a fitness of 1.00 (4 out of 4 examples are solved). Yet, the same algorithm might yield the same solution for the second case f' illustrated in Table 1b, with an additional example to the set of Table 1a. Solution f_3, showing a high, but not perfect, fitness, 0.8 $(4/5)$, will have a fair chance to take part of the reproduction and many other similar *alpha* individuals, sharing a similar behavior, will form the majority of the population. Indeed, during evolutionary computation, solutions showing a high fitness have better chances to breed the individuals of the next generation. *Alphas* grow numerous and their dominance arises against candidates with a lower fitness – whose genetic material is lost through successive generations. A local optima, or *single fitness peak*, is reached.

In our running example, consider the following arrangement where, among five candidate solutions, the one that could bring the necessary genetic material required to solve Example 5 is mistakenly neglected.

Typically, f_0 features an interesting operator (square, in $2a^2$) but exhibits a weak fitness since it only solves the case of Example 5 (*i.e.,* $\{a = 3, b = 3, c = 12\}$). Maintaining diversity shall favor the survival of this genetic material through generation and therefore, potentially help with solving other cases.

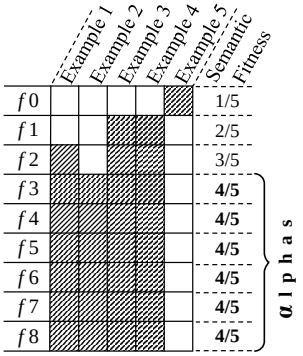


Fig. 5: Overbreeding of alphas during GP run.

Concretely, a phenotypic diversity value represents the distinctiveness of an individual's output (c) among other individuals' outputs, whereas a genotypic diversity represents the distinctiveness of an individual regarding its internal structure (*i.e.,* its *f()* function). We investigate the use of genotypic measurement for diversity in Section 5.

3.2 Social Semantic Diversity

We propose a new diversity measure that prevents selection being "biased towards highly fit individuals" [84] by an explicit consideration of individuals' participation in the population's behavioral diversity. Back to our running example, Solution 0, f_0, solves an example that no other solution solves, Example 5, and therefore it should purposefully be given more attention. In our approach, the weight of an example in the evaluation of an individual's fitness is no longer absolute but varies dynamically with the geometry of the population. In simple terms, the more solutions cor-

rectly handle an example, the less this example counts in the calculation of the solutions' fitness.

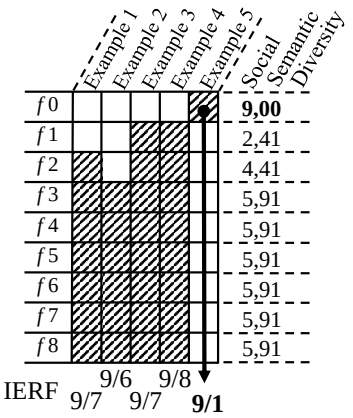


Fig. 6: Inverse Example Resolution Frequency (IERF). Grey cells shows which examples a solution solves (*e.g.,* solution 0 solves only Example 5).

We call *Social Semantic Diversity* a measure that takes into account, not the only individualistic fitness (*i.e.,* how many examples an individual resolves), but considers as well a social dimension (*i.e.,* what does that individual bring to the general fitness of the population).

The computation of the Social Semantic Diversity (SSD), based on the *inverse example resolution frequency* (IERF), is inspired from the *term frequency-inverse document frequency* (TF-IDF) information retrieval technique. In pragmatical words, the SSD of a solution is the sum of IERF of the examples it solves.

Paraphrasing *TFIDF* definition may help the reader to grasp the general idea of SSD. We formulate it as follows: "SSD increases proportionally to the number of examples solved and is offset by the frequency of which an example is solved by the population's individuals, which helps to adjust for the fact that some examples are more frequently solved in general." As a consequence, SSD favors solutions solving *corner cases* by giving importance to candidate solutions solving examples that other do not. In Fig. 6, Example 5 is solved by only one solution and thus gains a stronger IERF than other examples. As a consequence, solutions solving this example gain better SSD and are favored for breeding the next generation.

4 Learning MDE artefacts, the case of Well-Formedness Rules

In this section, we illustrate how social diversity can be implemented in a multi-objective genetic-programming algorithm to learn well-formedness rules (WFRs) from examples. This case is significantly more complex than the example in the former section, yet the method remains the same. As mentioned earlier in this paper, researchers offer to use GP to learn some of MDE artifacts automatically as a substantial alternative to writing them manually. Indeed, we show in this paper that, during the process, which scalability remains at stake [46], an improvement in populations' social diversity will lead to more efficient learning and more generalizable results.

4.1 Genetic Programming adaptation

Our goal is to find the minimal set (*i.e.,* size) of WFRs that best discriminates between the valid and invalid example models (*i.e.,* fitness). Size and fitness objectives being contradictory in nature, we represent the learning of WFRs as a multi-objective optimization problem, and we solve it using the Non-Sorting Genetic Algorithm NSGA-II [32].

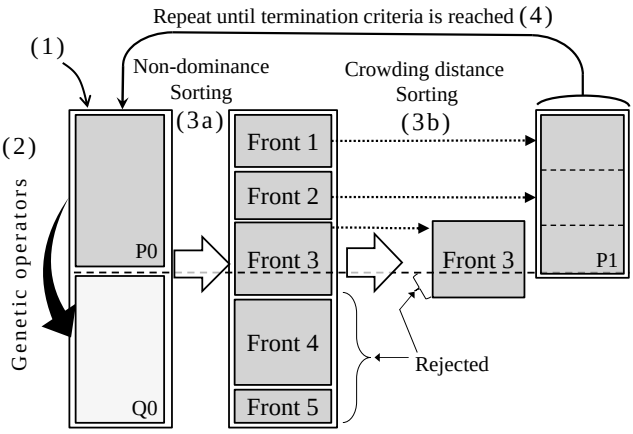


Fig. 7: Non-Sorting Genetic Algorithm NSGA-II [32]

The idea of NSGA-II is to make a population of candidate solutions evolve toward near-optimal solutions in order to solve a multi-objective optimization problem. NSGA-II is designed to find a set of optimal solutions, called non-dominated solutions, also Pareto set. A non-dominated solution provides a suitable compromise between all objectives such that one objective cannot be further improved without degrading another objective. As described in Fig. 7, the first step in NSGA-II is to create randomly a population P_0 of $N/2$ individuals encoded using a specific representation (1). Then, a child population Q_0, of the same size, is generated from the population of parents P_0 using genetic operators such as crossover and mutation (2). Both populations are merged into an initial population R_0 of size N, which is sorted into dominance fronts according to the dominance principle (3a). A solution s_1 dominates a solu-

tion s_2 for a set of objectives $\{O_i\}$ if $\forall i, O_i(s_1) \geq O_i(s_2)$ and $\exists j \mid O_j(s_1) > O_j(s_2)$. (This definition holds for parallel *maximization* and *minimization* of objectives.) The first (Pareto) front includes the non-dominated solutions; the second front contains the solutions that are dominated only by the solutions of the first front, and so on and so forth. The fronts are included in the parent population P_1 of the next generation following the dominance order until the size of $N/2$ is reached. If this size coincides with part of a front, the solutions inside this front are sorted, to complete the population, according to a crowding distance which favors "diversity" in the solutions (3b). This process will be repeated until a stop criterion is fulfilled, *e.g.,* a number of iterations is achieved or a certain value of semantic fitness is reached.

We adapted NSGA-II to our problem as follows.

- **Solution Representation and Creation.** A solution to our problem is represented as a set of OCL constraints (previously depicted in Section 2.1), each implementing a WFR represented as a tree. The initial population is created randomly. For each individual, the average number of nodes in the WFR trees, the maximum depth, and the maximum width are configurable;
- **Objectives.** We consider three objectives: *Semantic Fitness* is the rate of negative and positive examples processed accurately by an individual, to be maximized; *Size* is the number of leaves in the constraint tree, the smaller the better, normalized within a range of expected number of leaves; and *Diversity* is SSDM, which can be represented either as an objective or a crowding distance, to be maximized as well.
  The size objective is important. It allows the size of the solutions to be controlled during the course of the evolution. When the evolution is complete, priority is given to solutions that have a high semantic success rate rather than those that are small. In this sense, the two objectives are not taken on an equal level. However, control during evolution is essential so as not to generate unusable solutions (i.e., bloated).
- **Reproduction.** As genetic operators, we use a single-point crossover applied to the tree-root vector, and two kinds of mutations. First, a node from a WFR tree is chosen randomly. If it is a leaf, the pattern instance is either replaced with a new randomly created one or, if applicable, the pattern parameters are replaced randomly with applicable values. If the selected node is a logical operator, it is changed randomly.
- **Termination criteria.** Evolution stops if either a Semantic Fitness of 100% is achieved, or an arbitrary large number of iterations is reached.

4.2 Diversity implementation

De Jong *et al.* show that diversity is not an objective in the conventional sense, since diversity applies to the populations visited during the learning, not to final solutions [29]. As a consequence, we implemented our social diversity metric as part of two different steps in the evolutionary execution of NSGA-II. The first is as an objective of its own, considered together with afore-mentioned size and fitness objectives – as promoted by de Jong *et al.* [29] and applied to the Frequency Assignment problem by Segura *et al.* [75].

Another implementation of a diversity measure builds on peculiar limitation of NSGA-II [39] and acts as an alternative to the computation of a crowding distance. In both cases, SD computation remains the same.

In the following subsections, we start by explaining the implementation of our social semantic diversity.

4.2.1 Social Semantic Diversity implementation

Implementing Social Semantic Diversity (SSD) measure comes to adapting TF-IDF [78] using solutions as documents and examples as words. This is detailed in Listing 1. The first nested loops compute all examples resolutions frequencies.

At a given iteration, SSD is calculated from a binary matrix in which each cell represents the score of an individual against an example of the training set ($sol\_vs\_examples$). The frequency of an example (fq_ex) is the number of times it is solved by individuals (first *for* loop). Finally, individual's SSD value is the sum of *inverse example resolution frequencies* of examples that it processes accurately (last *for* loop). More precisely, variables are:

- $example\_set$, the vector of training examples (size D);
- $sol\_vs\_examples$, contains the result of the comparison between output of individuals and output of the oracle when executed on $example\_set$;
- fq_ex, contains examples frequencies, recording how many solutions solve each example from $example\_set$;
- and $ierfi$, the vector of *inverse example resolution frequencies* of training examples.

5 Genotypic alternatives

In this section, we show how social diversity can also be implemented at the syntactic level, *i.e.*, genotypic. To this respect, we present a definition of two genotypic diversity strategies, one problem-independent and one problem-dependent. These strategies are used for evaluation purpose but can be used as alternatives to or in combination with our social diversity strategy.

Listing 1: Excerpt for SSD weights calculation.

```
\\ Compute frequencies of examples solved
for (i = 0; i < sol_vs_ex.length; i++)
  for (j = 0; j < sol_vs_ex[i].length; j++)
    fq_ex[j] += sol_vs_ex[i][j];

\\ Inverse document frequencies
for (j = 0; j < fq.length; j++)
   ierfi[j] = Math.log10((fq_ex[j]>0) ?
                          D/fq_ex[j] : 0);

\\ Weigthing
weight = 0;
for(j = 0; j < example_set.length; j++)
  if(example_set[j].isAccurate())
    weight += ierfi[j];
```

5.1 Genotypic Diversity

We regard genotypic diversity as two distinct genres. There are strategies that compute diversity at the syntactic level only, whereas others use extra-knowledge specific to the problem domain.

A common approach to address diversity is to use text comparison techniques from research on Natural Language Processing (NLP) to compare between individuals at a syntactic level (or genotypic level) [19, 10]. NLP techniques support the extraction of structured information from raw text written in natural language. Considering a certain degree of naturalness in source code [49], NLP algorithms applied to the source code of candidate solutions shall give us a coherent heuristic to measure the dissimilarity between one another and to measure populations' diversity. These strategies offer therefore the benefit to be completely generic. Since it is only considering the syntax, it is also prone to work independently to the implementation language. In Auerbach words, they are "free from requiring defining dimensions on which to measure diversity and performance" [3]. Other strategies are domain specific and require being finely tailored to the specific problem to solve. This is the case for strategies focusing on genetic operators, or distance evaluation between candidate individuals [24].

We present, in the following paragraphs, two genotypic diversity measures, one generic and one problem specific. This gives us baselines for the evaluation of our contribution.

Rouge Genotypic Diversity. We use ROUGE algorithm to measure genotypic dissimilarity among individuals (Recall Oriented Understudy for Gisting Evaluation) [57]. It includes measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. In our context, it is a convenient tool to compute the distance (*i.e.,* the distinctiveness) of an ideal text (*i.e.,* the individual under scrutiny) among a corpus of texts (*i.e.,* the other individuals in the population). As shown in Fig. 8, Rouge Genotypic Diversity (RGD) is the sum of the comparisons to all individual in a corpus of solutions. With a generic consideration of the source code, one of the main advantage of ROUGE diversity is that it is agnostic of the problem definition.

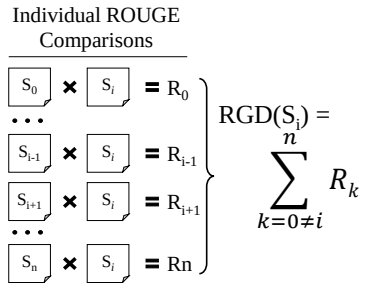


Fig. 8: ROUGE Genotypic Diversity measurement

Social Genotypic Diversity. On the other hand, considering the benefits of extra knowledge – about the expected solution's language for instance, *e.g.*, specific typecasting – might help precise diversity concerns. The idea is to ensure that the algorithm explores a maximum of the grammar elements during evolution. In our example, arithmetic expressions use natural numbers together with a set of operators defined beforehand by a precise grammar. Solution 0, f_0, should be favored for it employs an operator that the perfect solution requires and no other solutions do (namely, square power). To ensure that the evolution process covers at best grammar elements, we weight individuals fitness with the frequency of which the elements of the grammar are used by the population's individuals.

In the context of MDE, grammar elements refer to elements of the grammar itself, plus all typing information usually defined by one or more metamodels. Solutions in the population are MDE artefacts, they instantiate part of the metamodel they conform to. For example, OCL constraints instantiate part of the OCL metamodel, and ATL[4] transformations instantiate part of the ATL metamodel). We call this metamodel a *solution metamodel*. For their part, the artefact-solutions manipulate models. These models instantiate part of the metamodel they are derived from. This is for example `FamilyTree` or `ProjectManager` in this paper. We call this metamodel a *typing metamodel*.

Fig. 9 illustrates the mechanism. There are two metamodels involved: a solution metamodel, and a typing metamodel – one defines the language of the solutions, the other the dedicated language for the input models of those solutions. The rate a population of solutions *covers* a language

[4] ATL Transformation Language `https://www.eclipse.org/atl/`.

is the rate of elements of that language instantiated at least once in all solutions in the population (1, and hatched section). A set of solutions also *covers* to a certain extent typing metamodels (2, and hatched section). Social Genotypic Diversity measures the distinctiveness of a solution against a population of solutions. For a specific candidate solution, there are elements already instantiated by the population (3), and other that this only solution instantiates (4) and (5). The more elements of the latter, the more a solution is distinct from the population.

Considering this diversity measurement is meant to help the evolution by an augmentation of the breadth of exploration. The main drawback of this kind of genotypic diversity is that the dimension on which to consider diversity must be defined for each problem specifically (e.g., arithmetic language, domain specific language...).

5.1.1 ROUGE Genotypic Diversity implementation

To implement ROUGE in our framework, we used the open-source library proposed by Ganesan *et al.* with ROUGE-L (*i.e.,* summary level Lean Competency System (LCS)) [42]. Saggion *et al.* shows the benefits of using normalized-pairwise LCS to compare similarity between two texts in automatic summarization evaluation [70]. In our adaptation to diversity concern, at each generation, every individual is evaluated against the corpus of all other individuals in the population. To be exact, it is the *OCL code* of the individuals that is compared to the OCL code of others. As illustrated in Fig. 8, we take the sum as a measure for distinctiveness of this individual (or diversity).

5.1.2 Social Genotypic Diversity implementation

Implementing Social Genotypic Diversity (SGD) measure is done by adapting TF-IDF [78], but using solutions as documents and typing elements from metamodels as terms. Therefore, our implementation follows the same reasoning as in Section 3 and comes to overriding Listing 1 with the following changes in variables:

- $example\_set \rightarrow mmelts\_set$, the conjunct vector of all metamodel elements used by at least one solution;
- $sol\_vs\_examples \rightarrow sol\_vs\_mmelts$, contains the lists of metamodel elements used by each solution;
- $fq\_ex \rightarrow fq\_mmelt$, contains metamodel elements frequencies, recording how many solutions use each metamodel element from $mmelts\_set$;
- and $ierfi \rightarrow imufi$, contains a vector of *inverse metamodel elements usage frequencies*.

Table 2: Descriptive statistics of the examples used to train our algorithm in the three alternative cases. Values are averages among the 20 examples.

	`FamilyT.`	`StateM.`	`ProjectM.`
LOC	13.4	13.2	16.3
# classes	10.2	10.4	13
# properties	42.25	67	101.2

6 Evaluation

To assess the improvement brought by our social semantic diversity to the learning strategy, we conducted an empirical evaluation[5]. We formulate our research questions as follows:

- **RQ0** (Sanity check): Are our results a consequence of an effective learning (*i.e.,* an efficient exploration of the search space), or are they due to the vast number of individuals we evaluate during the learning?
- **RQ1**: Does the use of our Genotypic Diversities improves the learning strategy, and, if so, how much?
- **RQ2**: Does the use of Social Semantic Diversity as an objective improves the learning strategy, and, if so, how much?
- **RQ3**: Does the use of Social Semantic Diversity as an alternative crowding distance exhibit better efficiency and generalizability than as an objective?

6.1 Setting

In order to mitigate the influence of a metamodel specific structure on the learning process, we selected three metamodels that demonstrate different levels of structure complexity and require diverse OCL WFR sets: `FamilyTree`, `StateMachine`, and `ProjectManager`. We provided with oracle (*i.e.,* expected WFRs) manually. In more details, `FamilyTree` is the most simple case with 5 classes. Yet, it has been used as an illustrative example in various publications in the MDE research literature, such as [44]. `StateMachine` illustrates structural cardinality restrictions and define a common, widely used language, it has 6 classes. Finally, `ProjectManager` is the most complex case with 11 classes and comes from Hassam *et al.* [47]. As an example, Listing 2 shows the WFR constraining the `StateMachine` metamodel.

6.1.1 Training examples

To provide with example sets of *quality* (*i.e.,* covering at best the modelling space, yet as small as can be), we used

[5] All experiment data is available at `http://geodes.iro.umontreal.ca/publication_material/ssd_ext`

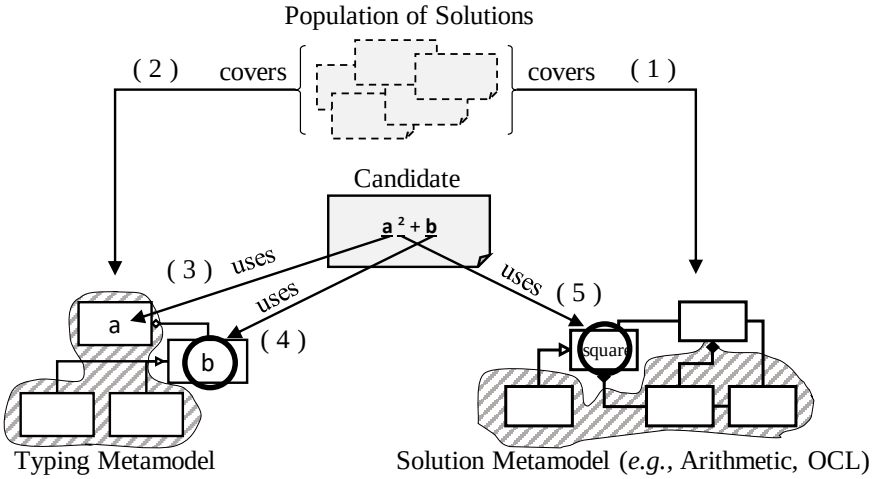


Fig. 9: Social Genotypic Diversity: a candidate solution is "distinct" from other solutions in the sense that it uses different part of the languages (Arithmetic definition and/or application metamodel(s)).

Listing 2: Oracle for the Statemachine metamodel: expected WFRs used to tag the example sets for training and testing.

```
context Choice
inv choice_in : incoming->size() > 1
inv choice_out : outgoing->size() > 1
context Final
inv final_out : outgoing->size() = 0
context Fork
inv fork_in : incoming->size() = 1
inv fork_out : outgoing->size() > 1
context Initial
inv init_in : incoming->size() = 0
context Join
inv join_in : incoming->size() > 1
inv join_out : outgoing->size() = 1
```

a model generator [8]. Size matters since every generated model example must be, in a real setting, tagged manually as *valid* or *invalid*. For the sake of experiment, we use the existing WFRs as oracles to mimic the manual tagging. To run the experiment, we used two sets of examples for each metamodel. On the one hand, 20 models (10 valid, 10 invalid) were required for the learning step (a *training set*). The average size in terms of LOC, number of classes, and number of properties instantiated is shown in Table 2. On the other hand, a *test bench* of 100 models (50 valid, 50 invalid) was used to measure solutions' average accuracy (or generalizability, since models in the test bench were not part of the training set).

6.1.2 Configurations and variables

Eight configurations were considered to illustrate and answer our research questions (see Section 4.1 for implementation details).

Table 3:

Parameter	Range	Min	Max	Choice
Max. number of generations	500	2000	6000	3000
Population size	10	10	50	30
Crossover rate	0.1	0.6	0.9	0.9
Mutation rate	0.1	0.1	0.4	0.3

- *RND* is a random exploration of the potential solutions that takes the best among a given number of solutions randomly generated;
- *STD* is a standard run of NSGA-II [32] with two objectives: size and semantic fitness. This alternative contains a phenotypic measurement for diversity through the crowding distance;
- SGD_{obj} is a run of NSGA-II with three objectives: size, semantic fitness, and SGD diversity; and SGD_{cd} is a run of NSGA-II with size and semantic fitness as objectives, and SGD as crowding distance;
- RGD_{obj} is a run of NSGA-II with three objectives: size, semantic fitness, and RGD diversity; RGD_{cd} is a run of NSGA-II with size and semantic fitness as objectives, and RGD as crowding distance;
- SSD_{obj} is a run of NSGA-II with three objectives: size, semantic fitness, and SSD diversity; and SSD_{cd} is a run of NSGA-II with size and semantic fitness as objectives, and SSD as crowding distance.

We used two dependent variables to quantify experiment results: **#GEN**, the number of generations the evolutionary computation needed to find a solution. A grid search with increased valued revealed that a score of 3000 means that there was no solution with perfect fit found during the learning. And **ACC**, the proportion of examples from the test bench a solution process accurately.

6.1.3 Evaluation protocol

For the NSGA-II parameters, we use a maximum number of iterations of 3000 and a population size of 30 solutions. Crossover and mutation probabilities are set to 0.9 and 0.3 respectively. In addition, solutions are created with between 5 to 15 WFRs with each WFR having a maximum depth of 3 and width of 15. We answer RQ0 with a comparison between the results given when using SSD as an objective (SSD_{obj}) in the learning strategy and those of a random exploration (RND). Since our strategy explores 3000*30 solutions, the random exploration explores randomly 90000 solutions as well and considers the best individual so created.

We answer RQ1 with a comparison between the solutions obtained after executions with and without genotypic diversity. First, we compare results from RGD_{obj} and RGD_{cd} with STD, then results from SGD_{obj} and SGD_{cd} with STD. We answer RQ2 with a comparison between the solutions obtained after execution with and without social semantic diversity objective (respectively SSD_{obj} and STD). Finally, we answer RQ3 by comparing the configurations with Social Semantic Diversity objective (SSD_{obj}) and with Social Semantic Diversity crowding distance (SSD_{cd}). We ran each treatment 100 times to tackle GP nondeterminism and we guarantee statistical significance of the findings using the Mann-Whitney U test.

6.2 Results and analysis

6.2.1 RQ0 - Sanity check

As can be seen in Table 4, the RND configuration gives very poor results in comparison with an execution in SSD_{obj} for the two most complex metamodels (average accuracy on test bench is 0.5 vs 0.76 for `ProjectManager` and 0.53 vs. 0.94 for `StateMachine`). The difference in both cases is statistically significant (*p-value*<0,001) and the effect size is large (*Cohen's d* > 5). For the small metamodel `FamilyTree`, although statistically significant, the difference and the effect size are small. ***We can conclude that using SSD_{obj} configuration is more efficient than using a random search.***.

Table 4: Statistical comparison of results between random search and our learning approach on three WFR learning scenarios.

	Average ACC Value		M.-W. p-value	Eff. Size Cohen's d
	RDN	SSD_{obj}		
ProjectManager	0.5	0.76	<0.001	7.35
StateMachine	0.53	0.94	<0.001	5.38
FamilyTree	0.93	0.99	<0.001	0.74

6.2.2 RQ1 - Does Genotypic Diversity foster evolution?

Among the genotypic configurations, only RGD_{cd} shows positive results as can be seen in Table 5. Rows 1 and 3 show small yet significant improvement for two metamodels, `ProjectManager` and `FamilyTree`. These conclusions do not hold for the `StateMchine` metamodel.

Otherwise, Fig. 10b is explicit: other configurations using genotypic diversity (RGD_{obj}, SGD_{cd}, and SGD_{obj}) perform almost all iterations (#GEN 3000) without converging to accurate solutions and show a poor ACC. In other words, both efficiency and accuracy decline when using genotypic diversity (not shown in table).

This behavior was predictable in part because of the nature of *RGD* and *SGD*. When employed as crowding distance, a genotypic diversity may confound the fitness ranking and the algorithm may miss interesting candidate solutions, thereof the convergence is at stake. As a matter of fact, when embedding genotypic diversity in a third objective, solutions overgrow in size and satisfy only low accuracy. Genotypic diversity objective, together with a size objective, seems to eclipse the effort of the accuracy objective. Indeed, solutions grow bigger, and the convergence of fitness is lower with genotypic diversity objective.

Table 5: Statistical comparison of results between *STD* and RGD_{cd} on three WFR learning scenarios.

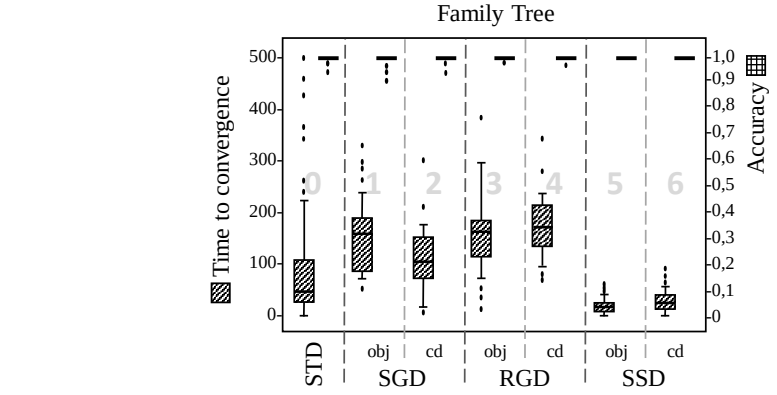
	Average ACC Value		M.-W. p-value	Eff. Size Cohen's d
	STD	RGD_{cd}		
ProjectManager	0.69	0.75	<0.001	0.69
StateMachine	0.92	0.65	<0.001	-2.47
FamilyTree	0.98	0.99	<0.01	0.40

Yet, the reader will as well notice that crowding distance configurations RGD_{cd} and SGD_{cd} outperform their objective equivalences, respectively RGD_{obj} and SGD_{obj} (see respective columns in Fig. 10b).

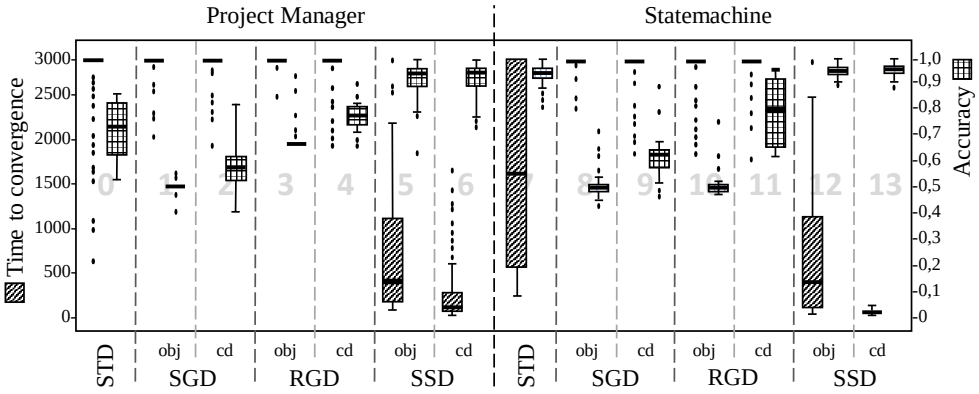
We believe that executions featuring genotypic diversity crowding distance show a very broad variation of candidates solutions during the first iterations of the evolution. Referring to Crepinsek *et al.* statement on exploration and exploitation phases [87], ***we conjecture that genotypic diversity can foster the exploration phase (*i.e.*, in the beginning of the evolution the more genetic material probed, the better) and then one must employ other means to exploit the outcome of that exploration.***

6.2.3 RQ2 - Social Semantic Diversity, an improvement?

Since genotypic configurations show at best inconsistent results depending on the choice of metamodel, we use *STD* executions as a comparison ground to evaluate benefits of



(a) `FamilyTree` metamodel.



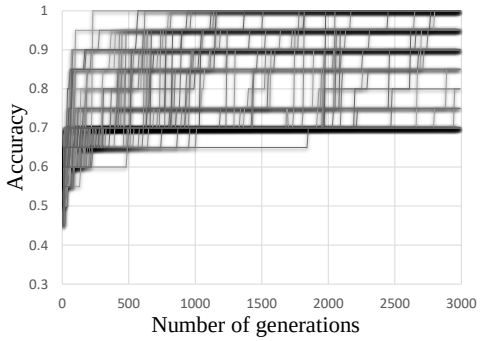
(b) `ProjectManager` and `StateMachine` metamodels.

Fig. 10: Number of generations to find solutions and their accuracy (ACC) on test bench.

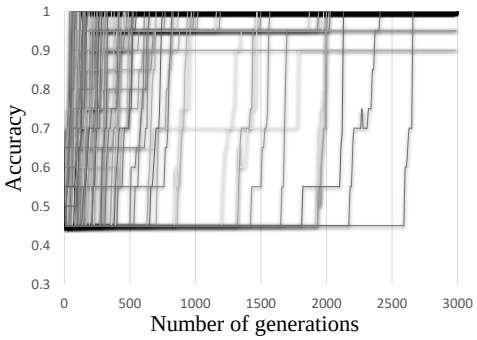
using a Social Semantic Diversity. Efficiency shows a significant improvement when SSD_{obj} is used, as can be seen in respective columns (5 and 12) of Fig. 10b. The number of generations required to find a solution when employing SSD_{obj} is a lot smaller than when employing *STD*. With `ProjectManager` metamodel, a *STD* run hardly finds solutions solving all training examples within the fixed 3000 generations (col. 0) whereas SSD_{obj} does it in 260 generations on average (col. 5). More, solutions were found with significantly better accuracy than *STD* (respectively 0.76 against 0.69) and thus strengthen solutions' generalizability likewise. This success is also noticed, if of lesser magnitude, during executions on the `StateMachine` metamodel (col. 7 to 13). Here, if solutions are found in both configurations, SSD_{obj} is significantly faster (with 481 generations, when *STD* requires more than 1782). As for the `FamilyTree` metamodel, solutions given by SSD_{obj} executions output a similar ACC (0.98), with 25 generations (resp. 79 with STD) – see Fig. 10a. ***We can conclude that injecting our social semantic diversity significantly improves the learning efficiency***.

6.2.4 RQ3 - Social Semantic Diversity as an alternative crowding distance, any better yet?

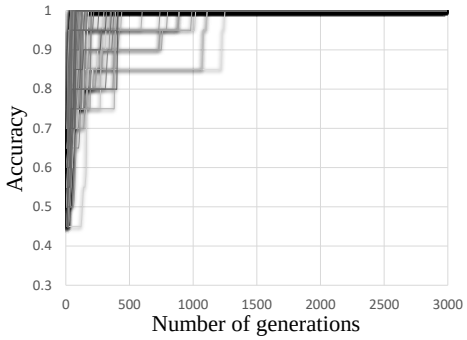
Results of RQ3 are flagrant (see the last configuration for both metamodels in Fig. 10b). In Fig. 11, a hundred runs show together how using *SSD* (11c and 11b) surges the learning curves and fosters solution exploration compared to a standard run (11a). As for generalizability, choosing between *SSD* as an objective (SSD_{obj}) or in the crowding distance (SSD_{cd}) does not have any significant impact on the accuracy of solutions on test bench found (Mann Witney p-value > 0.01; see even columns in Fig. 10b for an illustration). Thence, the main difference lies in the smaller average number of iterations SSD_{cd} needs to converge, compared to SSD_{obj} runs. Note that this analysis is the strongest with `ProjectManager` and `FamilyTree` metamodels. With the `StateMachine` metamodel results are slightly mitigated but remain significant. In that case, WFRs are more generally focused on structural cardinality than WFRs of the two other metamodels. We conceive this might be a factor for slightly different results. ***We can conclude that social semantic diversity as a crowding distance is more efficient than as an objective***.



(a) Standard evolution (STD)



(b) SSD as an objective (SSD_{obj})



(c) SSD as crowding distance (SSD_{cd})

Fig. 11: Evolution of individuals' average accuracy value during runs on `ProjectManager` metamodel, with a hundred runs a plot. (a) shows a standard run without diversity measure. (b) shows a steep, but randomly delayed rise of the Semantic Fitness when SSD_{obj} is used; (c) figures how unbelievably quick the Semantic Fitness converges when its crowding distance counter part SSD_{cd} is used.

6.2.5 Conclusion

In conclusion, as shown in Fig. 10b and Fig. 11 and certified with statistical analysis, both semantic diversity strategies significantly surpass a standard exploration of solutions. Convergence is faster and output more generalizable (*i.e.,* confronting solutions to a test bench gives better results). Conversely, genotypic diversity may at times hamper convergence in both crowding distance and third objective alternatives, and should be used with care.

A reason for these results might come from the way size is controlled. As recognized in the literature, we implemented it as a Pareto-based Multi-objective Parsimony Pressure [29].

Finally, solutions found are similar in structure and size - or at least, we did not find any particular trend between them. The rules found look like rules written "by hand", short and legible, although often very simple (which proves to be sufficient to satisfy the given problems). Solutions' size was indeed the one expected (*i.e.,* legible by a human), and the learning, passed a few generations, relied mainly on Semantic Fitness. The examples have been all solved (few runs did not converge toward a 1.0 ACC individual) and with no statistical peculiarities. As a presumed consequence,

Listing 3:

```
package statemachine
context Final
inv Cst_Final_40177 :
  not ( self.outgoing->size() > 0 )
context Choice
inv Cst_Choice_64113 :
  not ( self.outgoing->size() < 1 )
context Initial
inv Cst_Initial_64121 :
  not ( self.outgoing->size()
  = self.incoming->size()) AND
  self.incoming->size() = 0
context Fork
inv Cst_Fork_48580 :
  not ( ( self.outgoing->size() = 1 and
  not ( Fork.allInstances()->size() = 1 )
  ))
context Statemachine
inv Cst_Statemachine_15350 :
  not ( self.id.oclIsUndefined() )
endpackage
```

when putting SSD as an alternative to crowding distance, results were breathtaking on the three metamodels. Finally, using Social Semantic Diversity Measure as an alternative crowding distance outperforms its use as an additional objective. Convergence is boosted, and generalizability is kept at its maximum. We hope these results are generalizable and claim the need to explore other applications, with SSD_{obj} and SSD_{cd} alike, as well as other hybrid approaches featuring genotypic diversities.

6.3 Threat to validity

Although our approach produced good results on three metamodels, a threat to validity resides in the generalization of our approach to other scenarios. Still, the metamodels used show different characteristic and origin, and while our sample does not cover all learning scenarios, we believe that it is representative enough of a wide range of metamodels.

A known limitation of our approach lies in the avoidance of string attribute. We did not target WFRs constraining Strings for their derivation is still a consequent open topic, out of reach of this contribution, that would hamper its comprehension.

In our study, we performed multiple statistical tests, which may alter the significance of each of them, i.e., error-rate threat to validity. The significance values are, however, very small (< 0.001), and remain significant after using Benjamini *et al.* correction procedure [14].

The way we implemented the genotypic diversity measure for WFR learning can be challenged. Other implementation strategies could be experimented. For instance, since the learning of WFR uses OCL patterns, the coverage of these patterns could be an alternative measure instead of metamodel elements.

Another threat to the validity of our results relates to the use of a single set of (20) models to learn each WFR set. Characterization of example sets is an ongoing investigation, and different sets might show different results. Yet, to mitigate what specificities the manual design of models can bring and encourage replication of our work, we used a generator [8]. Also, using the same set in every configuration ensures a difference in sets do not interfere in the experiment.

The suitability of ROUGE algorithm for text comparison is supported by previous empirical evidence. Yet, different kinds of ROUGE as well as stemming (for instance, using Porter *et al.* algorithm [67]) or other preprocessing of solutions' syntax may yield different results. We envision further work, with characterized preprocessings, to explore the impact of lexical variations. In the same vein, there exist a plethora of approaches to apprehend textual representations. We chose ROUGE for it fits our primary goal of comparing between text fragments, but other NLP techniques could potentially be used.

Additionally, the capability of a solution to solve a given example is binary by nature for WFR learning, i.e., a model example is considered as valid or invalid by a WFR. For learning other MDE artifacts, such as model transformations, the evaluation of an example by a solution is not binary as what is produced can be valid to a certain degree. A candidate transformation can produce an output model that matches partially the expected one according to the considered example. Yet, our semantic diversity measure can be adapted using the conformance of solution to examples, as defined by Baki *et al.* [5], or model-to-model distance metric [38].

Finally, we evaluated only one implementation for each diversity type. Many others have introduced genotypic diversities in the past. Yet, most of these approaches are intended to optimize genetic algorithms execution whereas we target genetic programming and specifically search-based software engineering techniques for evolutionary learning. This is a specific context since objectives do not have the same importance: fitness is the goal, size an impediment to usable solutions, and diversity matters during evolution, not for the output. Our experiment must be replicated with different genotypic measurements. Yet, state-of-the art literature shows a small impact of different genotypic diversities compared to phenotypic equivalent. We conclude on the same note. No general conclusion is drawn, but so far this case with Rouge and phenotypic domain specific diversities perform (significantly and strongly) poorer on the three scenarios.

7 Related Work

The present study lies at the intersection of two different fields of investigation. On the one hand, we contribute to the advance of a more efficient learning of MDE artefacts, and we will present some of the latest works in the domain to sketch an up-to-date portrait of its potential benefits and pitfalls. On the other hand, the need for a control of population diversity during the execution of GPs is not new and we will introduce state-of-the-art works on diversity for genetic programming.

7.1 Learning MDE Artefacts

Research investigation on learning automatically MDE artefacts has started more than a decade ago. The artifact under prime scrutiny was, and remains model transformation. As early as 2006, Varro *et al.* [85] proposed a semi-automatic graph-based approach to derive transformation rules using interrelated source and target models. Balogh *et al.* [7] have

built an extension that derives n-m (instead of 1-1) rules using Inductive Logic Programming (ILP) and Wimmer *et al.* propose a similar approach with mappings defined in a concrete rather than an abstract syntax [89, 81, 80, 43]. Later on, teams started to automate the learning of model transformations by analogy. They do not try to abstract the transformation. They derive the corresponding target model from a source model by considering model transformation as an optimization problem [53, 54, 55]. More recently, Faunes *et al.* [37] proposed an approach to learn directly the code of transformations from examples. Genetic programming (GP) is used to learn n-m transformation rules starting from source and target examples without additional knowledge. To learn a transformation, application examples are employed (model pairs: potential inputs with their corresponding transformed version). The approach is enhanced by Baki *et al.* [6, 5] to learn the rules' execution control of model transformations. We foresee challenging future work in the replication of the present study featuring model transformation as objects.

The second MDE artifact under strong scrutiny is the well-formedness rules set. Faunes *et al.* [36] shows that it is feasible to learn well-formedness rules (OCL constraints) from examples and counter examples with ultimately no extra knowledge. Faunes' approach consists in using the set of examples and counter examples to participate in the evaluation of candidate solutions during a mono-objective GP run. In this scenario, a good solution differentiates between valid and invalid models accordingly. The results are promising, but the use of mono-objective seems problematic here. In fact, multi-objective GP allows dissociating between semantic and syntactic objectives. This permits the consideration of semantic diversity and thus promotes the generalization power of solutions [84]. Moreover, authors used oracles (the expected solution) to produce examples – which is irrelevant for an execution *in vivo*. More recently, Dang *et al.* [28] proposed a framework to infer OCL invariants from examples. Their work consists of translating OCL in a constraint logic program and using a solver to identify candidate solutions. The originality of this work is the involvement of the user during the process of learning. The process is iterative, and users are asked to check whether examples are sound and to rebuke the extravagant ones. They are also asked to point out the specific problematic part of malformed examples when applicable to help with providing examples in the next iteration. Later, the user is asked to assess the relevance of the solution(s) given by the algorithm. To our best knowledge, this is the first time that there is a concrete evaluation of solutions relevance. In the same vein, Clariso *et al.* [25] show the complexity of expressing and repairing specific misconceptions such as thresholds and complex select OCL expressions. The authors show that mutation analysis is a good candidate to tackle this issue.

7.2 Diversity Measure for Genetic Programming

Since the earliest work on genetic programming [56, 50], and more generally on evolutionary computation [33], researchers pointed out the importance of maintaining (and controlling) diversity of individuals in a population of candidate solutions during the process of evolution. This can be achieved during the crowding evaluation or during the selection of candidate reproducers [61]. Diversity has been considered on the syntax (or genotype) level as well as on the semantics (or phenotype) level of solutions. In this study, we offer to study both.

Among other striking works on diversity, Crepinsek *et al.* [87] conducted a major meta-study. Referring to exploration and exploitation viewpoints, authors see diversity as an important characteristic to address the drawbacks of the evolution process. They classify and name diversities depending on their nature (genotypic, phenotypic, or a mix of both), on their mode of application (measuring distance or entropy among a population, taking into account statistics of genes and/or behaviors in one generation or with a historical perspective). In this work, authors consider the control of diversity as a way to achieve balance between phases of exploration and exploitation during the evolution. For an in-depth investigation on diversity, we strongly recommend the reader to consider reading this seminal work.

State-of-the-Art investigations on diversity focus on "Dimensionality Reduction" [48, 1, 83, 92]. In the same vein, NSGA-III has been designed to foster diversity control on *Many*-Objective Optimization Problem (MaOOP) [30, 31]. We target a specific kind of problem in which only two or three objectives must be considered at maximum. More, the example-based configuration specifically provides with a distinct objective dedicated to the semantic evaluation of candidate solutions. Therefor, we consider that NSGA-II is more appropriate in our case.

The pressure toward which candidates will be elected for reproduction is another mechanism to control diversity [4]. As a matter of fact, the closest work to ours was performed by Byron *et al.* [19]. In their study, authors evaluate the importance of phenotypic diversity versus an estimated Hamming diversity applied on *selection factors* during the choice of reproducers. Along with a syntactic heuristic, they use a weighted value for diversity that considers the resolution rate of training elements. They conclude that when applied as selection factor, genotypic diversity seems more suitable than phenotypic diversity. The nature and the results of the experiment unveil a need for more investigation on the topic since they illustrate their approach on the only King-Rook-King problem. In the same manner, Eshelman *et al.* [35] offer, for instance, to prevent incest and more generally to favor parents that are at a distance from each other, and Szubert *et al.* [82] favor antagonist behaviors for reproduction.

The history of evolution has also been used to help decide whether parents are fit for reproduction or not [62, 40]. In all cases, these approaches require an important adaptation that is difficult to apprehend with complex MDE artifacts.

Diversity can be controlled through the choice of genetic operators [27]. Burks *et al.* [18] propose a new crossover method that acts on the root of tree-like solution representations to augment (deepen) diversity. As a consequence, a broader exploration of the solution space takes place (instead of a common random pick crossover leading to superficial changes). Adapting this kind of diversity to an MDE artifact, such as a WFR set or transformation rules, is a complex operation though and must be tailored in a *ad hoc* manner to every new problem. We plan to explore this possibility in the future.

As emphasized by Spears *et al.* [79], decomposing populations reveals to being also beneficial for diversity maintenance and the same year, Ryan *et al.* [68] propose "disassortative mating" which involves two populations: one has its individuals ranked depending on their fitness, the other has its individuals ranked on the sum of their size and weighted fitness. In the same fashion, Benbassat *et al.* [13] separate the population into subpopulations (buckets) based on similarity to bias selection (*i.e.,* individuals from all buckets will be used for reproduction). Muller-Bady *et al.* [65] offer to inject populations during the evolution to stir genetic material and augment diversity. We experimented with similar methods, but the small benefits were drowned by the results of the Social Semantic Diversity.

Decomposition-based MOEAs are based on transforming the multi objective problem into a set of single-objective optimization problems that are tackled simultaneously [23]. The difficulty of these kinds of approaches lies in the selection of proper weights that may depend on the form of the Pareto front. Decomposition-based approaches investigate the use of the star discrepancy measure in evolutionary diversity optimization [66]. The work is empirically evaluated with two settings studied in the literature, namely diversity optimization for images and Traveling Salesperson Problem instances. In the context of learning WFR artifacts though objectives are specific (*i.e.,* for a semantic goal, size control, and optionally diversity improvement). This context avoid the possibility to put them into a single weighted objective.

Along these lines, Seada *et al.* [74] describe an approach to capture dynamic changes in the relationship between selection, genotype-phenotype mapping and loss of population diversity. Finally, Galvan Lopez *et al.* [41] reminds us to be careful about the impact and *locality* of genetic operators.

Hitherto, there is no convergence regarding the benefits of diversity. As Affenzeller *et al.* [2] concluded recently, "diversity needs to be considered in the context of fitness improvement, and that more diversity is not necessarily beneficial in terms of solution quality." We address this issue with the use of SSD_{cd} which helps evolutionary computation converge and fosters generalizability of the solutions. Moreover, the dimension on which *SSD* is measured is directly related to the end purpose of a solution and no more precision need be detailed and implemented. Finally, our approach (with SSD_{cd} or SSD_{obj}) is independent of the domain definition and comes at no costs since examples' resolution (solutions' execution) must be, nevertheless, performed for fitness evaluation.

8 Conclusion

In this paper, we propose an innovative social semantic diversity measure for multi-objective genetic programming, and we study its impact on the search process for learning model well-formedness rules from examples and counter examples. The *Social Semantic Diversity* (SSD) is measured in a way that does not take into account the only individualistic fitness (*i.e.,* how many learning examples an individual resolves) but considers as well a social dimension (*i.e.,* what does that individual bring to the general fitness of the population). We integrated SSD in a genetic-programming version of NSGA-II algorithm as (i) an additional objective, and (ii) as an alternative to the crowding distance. For the sake of completeness and comparison, we also defined two genotypic diversity measures, one problem-domain agnostic, and the other dependent on the problem domain.

We evaluated the three different diversity measures, as both an additional objective or an alternative to the crowding distance, on the problem of learning well-formedness rules for three metamodels. Our results are compelling evidence that injecting our social semantic diversity in the learning process, especially as an alternative to the crowding distance, improves the convergence and the quality of the learned artifacts. The two genotypic diversity measures, however, did not bring substantial improvement to standard learning with diversity boost.

For problems with binary decisions, i.e., a solution does or does not solve a learning example, the proposed measure and its integration in the multi-objective genetic programming algorithm are agnostic with respect to the learned artifact, and the input/output examples used to guide the learning. This allows our social semantic diversity measure to be applied to a wide range of problems. The social diversity measure can be adapted, for non binary-decision problems, by integrating the degree with which a solution solves an example.

In the specific category of learning model-driven engineering artifacts, other studies are necessary to confirm the important gain in convergence and quality of the learned artifacts. We expect, in particular, to conduct some of these

studies, especially for model transformation learning. Finally, we encourage further replication of our work to determine whether different multi-objective GP algorithms could benefit as well from our discovery, outside the MDE problems.

References

1. S. F. Adra and P. J. Fleming. Diversity management in evolutionary many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 15(2):183–195, 2011.
2. Michael Affenzeller, Stephan M. Winkler, Bogdan Burlacu, Gabriel Kronberger, Michael Kommenda, and Stefan Wagner. Dynamic observation of genotypic and phenotypic diversity for different symbolic regression gp variants. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '17 Companion, pages 1553–1558. ACM, 2017.
3. Joshua E. Auerbach, Giovanni Iacca, and Dario Floreano. Gaining insight into quality diversity. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16 Companion, pages 1061–1064. ACM, 2016.
4. T. Back. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proc. of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 57–62, 1994.
5. Islem Baki and Houari Sahraoui. Multi-step learning and adaptive search for learning complex model transformations from examples. *ACM Trans. on Soft. Eng. and Methodology*, X:36, 2015.
6. Islem Baki, Houari Sahraoui, Quentin Cobbaert, Philippe Masson, and Martin Faunes. Learning implicit and explicit control in model transformations by example. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, volume 8767, pages 636–652. 2014.
7. Zoltan Balogh and Dániel Varró. Model transformation by example using inductive logic programming. *Int. J. on Soft. and Systems Modeling*, 8(3):347–364, 2009.
8. Edouard Batot and Houari Sahraoui. A generic framework for model-set selection for the unification of testing and learning mde tasks. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*. ACM, 2016.
9. Edouard Batot and Houari Sahraoui. Injecting social diversity in multi-objective genetic programming: The case of model well-formedness rule learning. In *International Symposium on Search Based Software Engineering*, pages 166–181, 2018.
10. Edouard Batot, Wael Kessentini, Houari A. Sahraoui, and Michalis Famelis. Heuristic-based recommendation for metamodel - OCL coevolution. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*. ACM, 2017.
11. Benoit Baudry and Martin Monperrus. The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Comput. Surv.*, 48(1): 16:1–16:26, 2015.
12. Lawrence Beadle and Colin G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3): 307–337, March 2009.
13. Amit Benbassat and Yuri Shafet. A simple bucketing based approach to diversity maintenance. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '17, pages 1559–1564. ACM, 2017.
14. Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A Practical and powerful approach to multiple testing. *J. Roy. Statist. Soc.*, 57:289–300, 1995.
15. Tommaso F. Bersano-Begey. Controlling exploration, diversity and escaping local optima in gp: Adapting weights of training sets to model resource consumption. In John R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 7–10, 1997.
16. P. A. N. Bosman and D. Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):174–188, 2003.
17. E. K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, Feb 2004.
18. Armand R. Burks and William F. Punch. An efficient structural diversity technique for genetic programming. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '15, pages 991–998. ACM, 2015.
19. James Byron and Wayne Iba. Population diversity as a selection factor: Improving fitness by increasing diversity. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16, pages 953–959, 2016.
20. Juan José Cadavid, Benoit Baudry, and Houari A. Sahraoui. Searching the boundaries of a modeling space to test metamodels. In *Proc. of the Int. Conf. on Software Testing Verification and Validation*, pages 131–140, 2012.
21. Juan Jose Cadavid, Benoit Combemale, and Benoit Baudry. Ten years of Meta-Object Facility: an Analysis of Metamodeling Practices. Research Report RR-7882, AtlanMod, 2012.

22. Juan Jose Cadavid, Benoit Combemale, and Benoit Baudry. An analysis of metamodeling practices for MOF and OCL. *Computer Languages, Systems and Structures*, 41:42 – 65, 2015.
23. Joel Chacón Castillo, Carlos Segura, Arturo Hernández Aguirre, Gara Miranda, and Coromoto León. A multi-objective decomposition-based evolutionary algorithm with enhanced variable space diversity control. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '17 Companion, pages 1565–1571. ACM, 2017.
24. G. Chen, C. P. Low, and Z. Yang. Preserving and exploiting genetic diversity in evolutionary programming algorithms. *IEEE Transactions on Evolutionary Computation*, 13(3):661–673, 2009.
25. R. Clariso and J. Cabot. Fixing defects in integrity constraints via constraint mutation. pages 74–82, Sep. 2018.
26. Vipul K. Dabhi and Sanjay Chaudhary. A survey on techniques of improving generalization ability of genetic programming solutions. *CoRR*, abs/1211.1119, 2012.
27. Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16, pages 645–652. ACM, 2016.
28. Duc-Hanh Dang and Jordi Cabot. On automating inference of OCL constraints from counterexamples and examples. In *Proc of the Sixth Int. Conf. on Knowledge and Systems Engineering KSE*, pages 219–231. Springer Berlin Heidelberg, 2014.
29. Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proc. of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, pages 11–18, 2001.
30. K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.
31. Kalyanmoy Deb and Dhish Kumar Saxena. On finding pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems.
32. Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In *Int. Conf. on Parallel Problem Solving from Nature - PPSN*, 2000.
33. A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundam. Inf.*, 35(1-4):35–50, 1998.
34. Anikó Ekárt and S. Z. Németh. A metric for genetic programs and fitness sharing. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming*, pages 259–270, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
35. Larry J. Eshelman and J. David Schaffer. Crossover's niche. In Stephanie Forrest, editor, *Proc of the 5th Int. Conf. on Genetic Algorithms*, pages 9–14. Morgan Kaufmann, 1993.
36. Martin Faunes, Juan Cadavid, Benoit Baudry, Houari Sahraoui, and Benoit Combemale. Automatically searching for metamodel well-formedness rules in examples and counter-examples. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, pages 187–202, 2013.
37. Martin Faunes, Houari Sahraoui, and Mounir Boukadoum. Genetic-programming approach to learn model transformation rules from examples. In *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, volume 7909, pages 17–32. 2013.
38. Adel Ferdjoukh, Florian Galinier, Eric Bourreau, Annie Chateau, and Clémentine Nebut. Measuring Differences To Compare Sets Of Models And Improve Diversity In MDE. In *ICSEA: International Conference on Software Engineering Advances*, Athenes, Greece, October 2017.
39. Félix-Antoine Fortin and Marc Parizeau. Revisiting the nsga-ii crowding-distance computation. In *Proc. of Int. Conf. on Genetic and Evolutionary Computation*, GECCO. ACM, 2013.
40. Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '17 Companion, pages 1572–1577. ACM, 2017.
41. Edgar Galván-López, James McDermott, Michael O'Neill, and Anthony Brabazon. Towards an understanding of locality in genetic programming. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '10, pages 901–908. ACM, 2010.
42. Kavita Ganesan. Rouge 2.0: Updated and improved measures for evaluation of summarization tasks. 2015.
43. Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. Model transformation by-example: An algorithm for generating many-to-many transformation rules in several model transformation languages. In Richard F. Paige, editor, *Theory and Practice of Model Transformations*, pages 52–66. Springer Berlin Heidelberg, 2009.

44. Martin Gogolla, Antonio Vallecillo, Loli Burgueno, and Frank Hilken. Employing classifying terms for testing model transformations. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, pages 312–321, 2015.
45. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.
46. M. Harman, Y. Jia, and Y. Zhang. Achievements, open problems and challenges for search based software testing. In *Proc. of the Int. Conf. on Software Testing Verification and Validation*, pages 1–12, 2015.
47. Kahina Hassam, Salah Sadou, and Régis Fleurquin. Adapting ocl constraints after a refactoring of their model using an mde process. In *9th ed. of the BElgian-NEtherlands software eVOLution seminar*, pages 16–27, 2010.
48. Z. He and G. G. Yen. Many-objective evolutionary algorithm: Objective space reduction and diversity improvement. *IEEE Transactions on Evolutionary Computation*, 20(1):145–160, 2016. Comparaison with NSGA-III (Niching for many objective.
49. Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *Proc. of the Int. Conf. on Software Engineering*, ICSE '12, pages 837–847, 2012.
50. John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992.
51. David Jackson. Phenotypic diversity in initial genetic programming populations. In Anna Isabel Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Şima Uyar, editors, *Genetic Programming*, pages 98–109. Springer Berlin Heidelberg, 2010.
52. M. Kessentini, W. Kessentini, H. Sahraoui, M. Boukadoum, and A. Ouni. Design defects detection and correction by example. In *Proc. of the Int. Conf. on Program Comprehension*, pages 81–90, 2011.
53. Marouane Kessentini, Houari A. Sahraoui, and Mounir Boukadoum. Model transformation as an optimization problem. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, pages 159–173. Springer, 2008.
54. Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Omar Ben Omar. Search-based model transformation by example. *Int. J. on Soft. and Systems Modeling*, 11(2):209–226, 2010.
55. Marouanne Kessentini, Houari Sahraoui, Mounir Boukadoum, and Omar Ben Omar. Search-based model transformation by example. *Int. J. on Soft. and Systems Modeling*, 11(2):209–226, 2012.
56. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
57. Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10, 2004.
58. H. Liu, L. Chen, K. Deb, and E. D. Goodman. Investigating the effect of imbalance between convergence and diversity in evolutionary multiobjective algorithms. *IEEE Transactions on Evolutionary Computation*, 21 (3):408–425, 2017.
59. Jesús J. López-Fernández, Esther Guerra, and Juan de Lara. Example-based validation of domain-specific visual languages. In *Proc. of the Int. Conf. on Software Language Engineering*, SLE 2015, pages 101–112, 2015.
60. Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evol. Comput.*, 14(3):309–344, September 2006.
61. R. Manner, Samir Mahfoud, and Samir W. Mahfoud. Crowding and preselection revisited. In *Parallel Problem Solving From Nature*, pages 27–36. North-Holland, 1992.
62. Nicholas Freitag McPhee and Nicholas J. Hopper. Analysis of genetic diversity through population history. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO'99, pages 1112–1120. Morgan Kaufmann Publishers Inc., 1999.
63. Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic building blocks in genetic programming. In *Genetic Programming*, pages 134–145. Springer Berlin Heidelberg, 2008.
64. Chihab eddine Mokaddem, Houari Sahraoui, and Eugene Syriani. Recommending model refactoring rules from refactoring examples. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, MODELS '18, pages 257–266, 2018.
65. Robin Mueller-Bady, Martin Kappes, Inmaculada Medina-Bulo, and Francisco Palomo-Lozano. Maintaining genetic diversity in multimodal evolutionary algorithms using population injection. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16 Companion, pages 95–96. ACM, 2016.
66. Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. Discrepancy-based evolutionary diversity optimization. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '18, pages 991–998. ACM, 2018.
67. M. F. Porter. Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. 1997.

68. Conor Ryan. Racial harmony in genetic algorithms. 1994.
69. Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Generation of operational transformation rules from examples of model transformations. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, pages 546–561, 2012.
70. Horacio Saggion, Simone Teufel, Dragomir Radev, and Wai Lam. Meta-evaluation of summaries in a cross-lingual environment using content-based metrics. In *Proc. of the 19th Int. Conf. on Computational Linguistics - Volume 1*, COLING '02, pages 1–7. Ass. for Computational Linguistics, 2002.
71. Jesús Sanchez-Cuadrado, Juan de Lara, and Esther Guerra. Bottom-up meta-modelling: An interactive approach. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, volume 7590, pages 3–19. 2012.
72. J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.
73. Douglas C Schmidt. Model-driven engineering. *IEEE Computer Society*, 39(2), 2006.
74. Haitham Ahmed Seada, Mohamed Abouhawwash, and Kalyanmoy Deb. Towards a better diversity of evolutionary multi-criterion optimization algorithms using local searches. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16 Companion, pages 77–78. ACM, 2016.
75. C. Segura, A. Hernández-Aguirre, F. Luna, and E. Alba. Improving diversity in evolutionary algorithms: New best solutions for frequency assignment. *IEEE Transactions on Evolutionary Computation*, 21(4):539–553, 2017.
76. Bran Selic. What will it take? A view on adoption of model-based methods in practice. *Int. J. on Soft. and Systems Modeling*, 11(4):513–526, 2012.
77. Terence Soule and James A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4): 293–309, 1998.
78. Karen Sparck Jones. Document retrieval systems. chapter A Statistical Interpretation of Term Specificity and Its Application in Retrieval, pages 132–142. 1988.
79. William M. Spears. Simple subpopulation schemes. In *In*, pages 296–307. World Scientific, 1994.
80. Michael Strommer and Manuel Wimmer. A framework for model transformation by-example: Concepts and tool support. In Richard F. Paige and Bertrand Meyer, editors, *Objects, Components, Models and Patterns*, pages 372–391. Springer Berlin Heidelberg, 2008.
81. Michael Strommer, Marion Murzek, and Manuel Wimmer. Applying model transformation by-example on business process modeling languages. In Jean-Luc Hainaut, Elke A. Rundensteiner, Markus Kirchberg, Michela Bertolotto, Mathias Brochhausen, Yi-Ping Phoebe Chen, Samira Si-Saïd Cherfi, Martin Doerr, Hyoil Han, Sven Hartmann, Jeffrey Parsons, Geert Poels, Colette Rolland, Juan Trujillo, Eric Yu, and Esteban Zimányie, editors, *Advances in Conceptual Modeling – Foundations and Applications*, pages 116–125. Springer Berlin Heidelberg, 2007.
82. Marcin Szubert, Anuradha Kodali, Sangram Ganguly, Kamalika Das, and Josh C. Bongard. Reducing antagonism between behavioral diversity and fitness in semantic genetic programming. In *Proc. of the Proc. of the Genetic and Evolutionary Computation Conf.*, GECCO '16, pages 797–804. ACM, 2016.
83. Y. Tian, R. Cheng, X. Zhang, Y. Su, and Y. Jin. A strengthened dominance relation considering convergence and diversity for evolutionary many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 23(2):331–345, 2019.
84. Leonardo Vanneschi, Mauro Castelli, and Sara Silva. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2): 195–214, 2014.
85. Dániel Varró. Model transformation by example. In *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, pages 410–424, 2006.
86. Dániel Varró, Oszkár Semeráth, Gábor Szárnyas, and Ákos Horváth. *Towards the Automated Generation of Consistent, Diverse, Scalable and Realistic Graph Models*, pages 285–312. Springer International Publishing, Cham, 2018.
87. Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3):35:1–35:33, July 2013.
88. J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *Software, IEEE*, 31:79–85, 2014.
89. Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards model transformation generation by-example. In *40th Hawaii Int. Conf. on Systems Science*, page 285, 2007.
90. Hao Wu. Generating metamodel instances satisfying coverage criteria via smt solving. In *Proc. of the Int. Conf. on Model-Driven Eng. and Soft. Development*, pages 40–51, 2016.
91. Bart Wyns, Peter De Bruyne, and Luc Boullart. Characterizing diversity in genetic programming. In *Proceedings of the 9th European Conference on Genetic Pro-*

gramming, EuroGP'06, pages 250–259, Berlin, Heidelberg, 2006. Springer-Verlag.

92. Y. Yuan, H. Xu, B. Wang, B. Zhang, and X. Yao. Balancing convergence and diversity in decomposition-based many-objective optimizers. *IEEE Transactions on Evolutionary Computation*, 20(2):180–198, 2016.