Evolutionary Diversity Optimisation in Constructing Satisfying Assignments

Adel Nikfarjam
Optimisation and Logistics
the School of Computer and Mathematical Science
The University of Adelaide
Adelaide, Australia

Ralf Rothenberger
Chair for Algorithm Engineering
Hasso Plattner Institute
University of Potsdam
Potsdom, Germany

Frank Neumann
Optimisation and Logistics
School of Computer and Mathematical Science
The University of Adelaide
Adelaide, Australia

Tobias Friedrich
Chair for Algorithm Engineering
Hasso Plattner Institute
University of Potsdam
Potsdom, Germany

ABSTRACT

Computing diverse solutions for a given problem, in particular evolutionary diversity optimisation (EDO), is a hot research topic in the evolutionary computation community. This paper studies the Boolean satisfiability problem (SAT) in the context of EDO. SAT is of great importance in computer science and differs from the other problems studied in EDO literature, such as KP and TSP. SAT is heavily constrained, and the conventional evolutionary operators are inefficient in generating SAT solutions. Our approach avails of the following characteristics of SAT: 1) the possibility of adding more constraints (clauses) to the problem to forbid solutions or to fix variables, and 2) powerful solvers in the literature, such as minisat. We utilise such a solver to construct a diverse set of solutions.

Moreover, maximising diversity provides us with invaluable information about the solution space of a given SAT problem, such as how large the feasible region is. In this study, we introduce evolutionary algorithms (EAs) employing a well-known SAT solver to maximise diversity among a set of SAT solutions explicitly. The experimental investigations indicate the introduced algorithms' capability to maximise diversity among the SAT solutions.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**.

KEYWORDS

SAT, Evolutionary Diversity Optimisation

ACM Reference Format:
Adel Nikfarjam, Ralf Rothenberger, Frank Neumann, and Tobias Friedrich. 2023. Evolutionary Diversity Optimisation in Constructing Satisfying Assignments. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3583131.3590517

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07…$15.00
https://doi.org/10.1145/3583131.3590517

1 INTRODUCTION

Combining the principle of EAs and diversity mechanisms has received increasing attention in the evolutionary computation community. Diversity is widely believed to be essential for survival in dynamic environments. In recent years, the benefits of having access to diverse solutions have been discussed in several studies such as Neumann et al. [26], Nikfarjam et al. [27]. We can categorise the advantage into three main groups: 1) Robustness against dynamic changes and imperfect modeling, 2) critical information about the solution space, and 3) increasing decision-makers' ability to consider and choose between diverse alternatives.

1.1 Related Studies

Traditionally, diversity is seen as a means to avoid premature convergence or explore niches in fitness landscapes of optimisation problems. Niching is a technique that usually divides the population into sub-populations. This enables the algorithms to explore and cover a broader range of solution space. Li et al. [20] provides a comprehensive review of niching methods. Two other paradigms, Quality Diversity (QD) and EDO, have recently evolved.

QD aims to compute a diverse set of high-quality solutions that differs in terms of some pre-defined behavioural characteristics. In fact, QD sees diversity in exploring best-performing solutions in a behavioural space. QD has been mostly studied in robotics [2, 34, 38], and game designs [15, 16, 36]. Recently, some studies applied QD's principles to combinatorial optimisation problems [7, 29, 33].

EDO is another concept recently developed around the idea of diversity. In contrast to the other paradigms, EDO explicitly maximises the structural diversity of solutions, often subject to a constraint on the solutions' quality. The concept has been defined in [37], which studied an optimisation problem in continuous domains. Afterwards, EDO has been adapted to generating benchmark instances for TSP and a diverse set of images respecting different aesthetics [1, 19]. These studies were followed up by works on the use of star-discrepancy, and multi-objective indicators [25, 26].

Bossek et al. [4] studied the performance of sophisticated mutations at generating a diverse set of benchmark instances for TSP.

Recently, the focus of the literature shifted from generating instances to computing solutions for combinatorial optimisation problems. There are several problems studied in that regard, such as the traveling salesperson problem (TSP) [11, 28], the knapsack problem (KP) [5], the quadratic assignment problem [12], the minimum spanning tree problem [6], the traveling thief problem [32], the optimisation of monotone Sub-modular Functions [24], and the patient scheduling problem [30]. In most of the mentioned papers, it has been assumed that we already know the optimal solution. The case of unknown optimal solutions has been studied in [23, 27, 31] by using co-evolutionary techniques or by dividing the population into two subpopulations.

This paper studies the SAT problem in the context of EDO. SAT is a classical problem in mathematical logic and computer science. The goal of the problem is to determine if there is an assignment to Boolean variables such that a given Boolean formula evaluates to true. The decision variant of SAT is one of the most well-known and well-studied NP-complete problems [9]. One can find many applications for SAT, such as software verification and constraint solving. Davis et al. [10] is one of the earliest studies carried out in SAT and introduced a method to compute a satisfying assignment. Several efficient approaches have been developed for SAT in recent years, like Conflict-Driven Clause-Learning (CDCL) [35] and the Variable State Independent Decaying Sum (VSIDS) branching heuristic [21]. One very versatile solver incorporating those heuristics is minisat [14]. Minisat has been widely adopted and used as the benchmark solver in the literature. To the best of our knowledge, Nadel [22] is the only study focusing on the diversity of SAT solutions. They studied Diverse$k$Set, the problem of finding a given number of diverse solutions to an SAT problem, by adapting the variable ordering strategy. However, there can be found another paradigm in the SAT literature, called uniform solution sampling. They aim to compute different solutions without taking diversity into account directly. UniGen2 [8] and QuickSampler [13] can be cited here.

1.2 Our Contribution

Several characteristics distinguish SAT from other problems studied in the EDO literature. For instance, the other problems contain either no or few constraints, such as the KP and the TSP. SAT, however, is a highly constrained problem, making it extremely difficult to generate a feasible solution with conventional operators and algorithms in the literature of EDO. In other fields such as constrained programming, researchers often forbid some variables or elements of a given problem to construct a diverse set of solutions. This paper makes a bridge between this approach and EDO. Instead of using conventional operators, which are inefficient in SAT, we introduce evolutionary algorithms (EAs) and operators that iteratively modify the original SAT problem by adding clauses. We use a time-efficient solver, such as minisat, to construct new solutions and utilise EDO approaches to maximise the diversity of the solutions. We define two entropy-based diversity measures to quantify the diversity of SAT assignments. The first measure treats all variables equally, while the other takes the frequency of variables in clauses into account. We also conduct a comprehensive experimental investigation, the goal of which is twofold: First, to evaluate the algorithms' performance in constructing diverse assignments. And second, to study the correlation among diversity, solution space, and the number of clauses. For this purpose, we use an SAT generator to construct instances with particular characteristics. Then, we observe how the changes in these characteristics affect the diversity of solutions and algorithms' performances. For example, The introduced mutation outperforms the crossover in the power law SAT instances, while it is the opposite in the uniform instances.

The remainder of the paper is structured as follows: We first define SAT and diversity in Section 2. The diversity algorithms are introduced in Section 3. The Comprehensive experimental investigation is presented in Section 4. Finally, we finish with concluding remarks.

2 SAT AND DIVERSITY

The Boolean Satisfiability Problem (SAT) consists of determining the existence of an assignment (also called model, interpretation, or solution) satisfying a Boolean formula. A Boolean formula is several literals combined by logical connectives, AND ($\wedge$), and OR ($\vee$), and a literal is a Boolean variable or a negation of a variable ($\neg$). A formula that is formed by the conjunction of a number of clauses (a disjunction of literals) is in conjunctive normal form (CNF). A formula in CNF is satisfiable if there is at least one assignment of the variables such that the formula evaluates to true. In other words, a given CNF formula Φ is true if an assignment x satisfies all clauses in Φ; otherwise, Φ is false. This paper aims to compute a diverse set of assignments for a given formula. For this purpose, we require a measure to quantify the diversity of assignments.

2.1 Diversity

We utilise an entropy-based measure of diversity. First, we define some notations. Let X denote the set of Boolean variables, $x = (x_1, \cdots, x_n)$ the assignment, and P a set of assignments, where $|X| = n$, $|P| = \mu$, m is number of the clauses. Also, let $f(x_i)$ be the number of assignments in P, where $x_i = True$. Then, we can calculate the contribution of each variable to diversity as

$$h(x_i) = \begin{cases} 0 & \text{if } f(x_i) = 0 \text{ and} \\ -\left(\frac{f(x_i)}{\mu}\right) \cdot \ln\left(\frac{f(x_i)}{\mu}\right) & \text{if } f(x_i) > 0. \end{cases}$$

In line with EDO literature [28, 30], the entropy of P can be calculated by summation of the variables' contributions:

$$H_1(P) = \sum_{x_i \in X} h(x_i)$$

Nevertheless, some variables appear in clauses more frequently than others. Such variables are likely to be more challenging to diversify, and often play a more important role in the problem. It would be intriguing to give more frequent variables more weight in the entropy calculation such that we first increase such variables' chance to be diverse and second the measure shows the diversity based on the frequency. Therefore, we define the second measure as follows:

$$H_2(P) = \sum_{x_i \in X} r(x_i) \cdot h(x_i),$$

where $r(x_i)$ is the number of occurrences of x_i in the formula. It is beneficial to know the maximum diversity for the measures. It can be used as an upperbound to evaluate a diversity of a set of solutions. We can calculate the optimal $f(x)$ from $\frac{dh(x)}{df(x)} = 0$; Thus, the contribution of a variable is at maximum when:

$$f(x) = \mu \cdot e^{-1}$$

Let denote the optimal $f(x)$ by f^*. Since there is no limitations on the number of true variables in P , H_1 and H_2 are maximum when $\{f(x) = f^* | \forall x \in X\}$. Then, we can calculate H_1^{max} and H_2^{max} form :

$$H_1^{max} = n \cdot f^*$$
$$H_2^{max} = C \cdot f^*$$

where C is the number of the literals in Φ.

3 DIVERSITY ALGORITHMS

In this paper, we compute a diverse set of assignments for a given SAT problem using the well-known SAT solver minisat. A basic approach to compute P for an SAT problem is to forbid the current assignment by adding a clause to the formula and using the solver to generate another one. For constructing the clause, we can easily make a disjunction of the literals where each literal is the flipped associated variable in the assignment. This method only sometimes leads to a diverse set of assignments. Algorithm 1 outlines the steps required for this approach.

Algorithm 1 The basic algorithm

```
while |P| < μ do
    Solve the SAT problem by the solver.
    if A satisfying assignment x was found then
        Add x to P.
        Add a clause forbidding x to Φ.
    else
        Break.
```

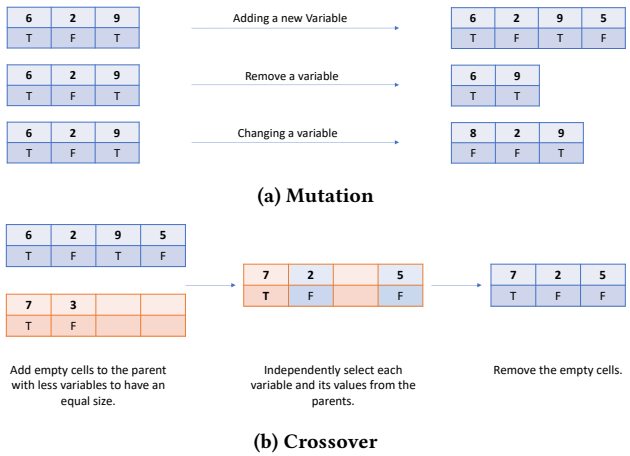
EDO is another method to compute a diverse set of assignments. We can fix some variables to true or false and then use the solver (minisat) to determine a satisfying assignment with those fixed variables. Afterwards, we can employ EDO approaches to maximise diversity. Here, the question is how to choose the fixed variables. In line with most EDO algorithms in the literature, we can randomly select one of the current solutions and, by standard bit flip mutation, flip some of the variable assignments and fix them. In contrast to the standard bit-flip mutation, where the rest of the variables remain unchanged, the solver determines the value for the other variables. Algorithm 2 describes this approach. First, we find the first satisfying assignment for Φ by minisat and add it to P. Then, we select a solution in P uniformly at random and choose and flip some variables by the bit-flip mutation. After adding clauses to Φ that fix the selected variables, we solve Φ by minisat. If a satisfying assignment is found, we add it to P; Then, if $|P| > \mu$, we remove an assignment x with the least contribution to the diversity of P. Finally, we remove the clauses fixing the variables from Φ. We repeat these steps until a termination criterion is met.

Algorithm 2 The bit-flip evolutionary algorithm

```
Solve the SAT problem by the solver, and add x to P.
while A termination criterion is met do
    Select an assignment x from P uniformly at random.
    Select and flip each variable independently with probability 1/n.
    Add clauses fixing the selected variables to Φ
    Solve Φ and determine unfixed variables by the solver.
    if A satisfying assignment x was found then
        if |P| > μ then
            Add x to P.
            Remove one individual x from P, where x = arg max_{x∈P} H(P \ {x}).
    Remove the clauses that fixing the variables from Φ.
```

Algorithm 3 The EDO algorithm

```
while |P| < μ do
    Randomly fix l variables (determine y).
    Add the clauses that fix the variables in y to Φ and solve it by the solver.
    if A satisfying assignment x was found then
        Add x to P and y to Y.
    Remove the clauses fixing the variables from Φ.
while A termination criterion is met do
    Randomly select one (two) parent(s) y_i (y_j) from Y.
    Generate a new solution y_o by mutation or crossover + mutation.
    Add clauses that fix the variables in y_o to Φ and solve the SAT problem.
    if A satisfying assignment x is found then
        Add x to P and y_o to Y.
        Remove one individual x from P, where x = arg max_{x∈P} H(P \ {x}), and the corresponding solution y from Y.
    Remove the clauses fixing the variables from Φ.
```



(a) Mutation

(b) Crossover

Figure 1: The representation of solution y, the mutation, and the crossover in the EDO algorithm 3.

Since minisat is an exact algorithm, we can map from the fixed variables to the actual assignments. Thus, we can save the fixed variables and operate (crossover, mutation) on them. So, we have a solution y consisting of a string $y' = (y'_1, \cdots, y'_l)$ showing the index of fixed variables and a Boolean string y'' showing their values. Let Y be a set of solutions y, where $|Y| = \mu$. Note that from each $y_i \in Y$ we can map to $x_i \in P$, by fixing variables in y_i and solving the problem by the solver.

Algorithm 3 sketches the steps required in this approach. The algorithm consists of two stages, the initialisation and the evolutionary stage. In initialisation, we randomly generate a variable y, where $|y| = l$. We solve ϕ after adding clauses to it. If a satisfying assignment x is found, we add x to P, and y to Y. Afterwards, we remove the clauses fixing the variables from Φ. we continue these steps until $|P| = \mu$.

Having constructed an initial population, we move to the evolutionary stage. We first select a solution y (or two solutions in case of crossover) from Y and generate an offspring y_o by mutation (or first crossover, then mutation). After adding clauses fixing variables in y_o to Φ, we solve it by the solver. If a satisfying assignment x is found, we add x to P and y to Y; then remove a x from P and the corresponding y from Y that has the least contribution to the diversity of P. In last step, we remove the clauses fixing the variables from Φ. We repeat these steps in the evolutionary stage until a termination criterion is met.

We now describe the operators, the mutation and the crossover. For the mutation, we take one of the following three actions uniformly at random: 1) Fix another variable (add a new variable to y), 2) unfix a variable (remove a variable from y), or 3) switch a fixed variable with an unfixed variable, all uniformly at random. The steps are depicted in Figure1a. Turning to the crossover, we add empty cells to the parent with fewer fixed variables to make the sizes equal. Then, we select each variable randomly from the parents with probability 1/2. Figure1b illustrates the steps required by the crossover.

4 EXPERIMENTAL INVESTIGATION

This section empirically studies and compares the introduced algorithms. We examine two variations of Algorithm 3: One solely employs mutation as the operator, while the other first generates an offspring by crossover and then uses mutation on the offspring. To examine the algorithms, we use the SAT generator [3] to generate two sets of CNF formulas. The SAT generator is also used for experimental investigations in [17, 18]. In the first set, the variables appear in clauses based on a power law distribution. The following parameters were used in generating the first set: $n = 100$, $k = 3$, $\beta = 2.75$, and $m = \{210, 220, \cdots, 380\}$, where k and β are the number of literals in a clause and the power law exponent, respectively. In the second set, the variables appear in the clauses based on the uniform distribution. The parameters for the set are: $n = 100, k = 3$, and $m = \{270, 280, \cdots, 440\}$. We set $\mu = 20$ and consider 2000 iterations as the termination criterion for the EAs. Instead of 30 independent runs on one formula, we generate 30 formulas for each configuration and run the algorithms once on each formula. This helps us to comprehend more about SAT instances having the same characteristics. In algorithm 3, l should be set with taking number of clauses into account; a higher number of clauses, a lower l. Based on preliminary investigation, We set $l = 10$ for the lowest number of clauses in each set and gradually decrease it to 4 for the greatest number of clauses. Note that we made sure all formulas were satisfiable ($\Phi = true$).

4.1 Comparison of algorithms employing H_1 as the fitness

In this section, we compare the diversity of SAT assignments obtained by the presented algorithms using H_1 as the fitness function. Table 1 summarises the algorithms' results in the first set of instances (formulas). As expected, the basic algorithm results in assignments with poor diversity; the H_1 values range between 1.71 and 10.01. If we normalise these values, the range is from 5% to 27%. The interesting information is that the increase in the clause-variable ratio $\frac{m}{n}$ has no meaningful impact on the basic algorithm's result. The expectation is that an increase in $\left(\frac{m}{n}\right)$ reduces the feasible region which leads to a decrease in the diversity of assignments; we can observe the trend in the results of the other algorithms.

As Table 1 shows, the bit-flip brings about considerably more diverse assignments than the basic algorithm. The observation can be confirmed by the Kruskal-Wallis statistical test at a 5% significance level and with Bonferroni correction. The mean of diversity ranges from 44% to 82%. Although there are also fluctuations in the bit-flip algorithm's results, we can observe a general decrease in diversity by an increase in $\left(\frac{m}{n}\right)$, especially when m is larger than 290. However, if we only consider the first half of the table, it is exactly the other way around; there is a slight increase in diversity obtained. One plausible reason is that the minisat solver is an exact algorithm, and bit-flip mutation does not impose as significant changes as required. On the other hand, an increase in $\left(\frac{m}{n}\right)$ makes even minor changes significantly impact the assignments. In fact, the feasible regain and the maximum achievable diversity decrease in instances with medium values of $\left(\frac{m}{n}\right)$ compared to small ones, but the bit-flip algorithm performs better in these instances.

Table 1 indicates the superiority of EDO algorithms in constructing diverse sets of SAT assignments. Both algorithm variants yield decent results and statistically outperform the basic and the bit-flip algorithms in all instances. Here, we can observe a more static downward trend in diversity with increasing $\frac{m}{n}$. It results in sets with more than 90% diversity (normalised H_1) for instances with $\frac{m}{n} \leq 3$. For example, the mean of diversity is 96% in cases where $m = 210$. Interestingly, the variant using only mutation results in slightly higher diversity. Although, it is not statistically significant.

Table 2 draws a similar comparison between the algorithms on the set of uniform formulas. Almost all our observations in Table 1 are still valid. Table 2 shows that: 1) Algorithm 1 results in solutions with poor diversity ranging from 6% to 29%. Nevertheless, the diversity obtained in the uniform instances is higher compared to the power law formulas. 2) Bit-flip performs better than the basic algorithm but worse than the EDO variants. The average H_1 obtained by the bit-flip algorithms ranges from 0.31 to 0.86. 3) We can observe a descending trend in diversity for increasing $\frac{m}{n}$, especially in the EDO algorithms' results.

The most interesting part of the table is comparing the two EDO variants. In contrast to the power law instances, the variant using

Table 1: The diversity obtained from the algorithms using H_1 as the fitness function in 30 independent runs. Stat shows the results of Kruskal-Wallis statistical test at a 5% significance level with Bonferroni correction. In row Stat, the notation X^+ means the median of the measure (H_1) is better than the one for variant X, X^- means it is worse, and X^* indicates no significant difference.

	Basic 1			Bit-flip 2			EDO 3 Mutation			EDO 3 Crossover+Mutation		
m	H_1	H_2	Stat (1)	H_1	H_2	Stat (2)	H_1	H_2	Stat (3)	H_1	H_2	Stat (4)
210	0.055	0.016	$2^-3^-4^-$	0.753	0.839	$1^+3^-4^-$	0.962	0.959	$1^+2^+4^*$	0.953	0.955	$1^+2^+3^*$
220	0.052	0.011	$2^-3^-4^-$	0.721	0.818	$1^+3^-4^-$	0.945	0.938	$1^+2^+4^*$	0.932	0.933	$1^+2^+3^*$
230	0.055	0.019	$2^-3^-4^-$	0.738	0.823	$1^+3^-4^-$	0.937	0.932	$1^+2^+4^*$	0.925	0.925	$1^+2^+3^*$
240	0.046	0.007	$2^-3^-4^-$	0.731	0.808	$1^+3^-4^-$	0.933	0.927	$1^+2^+4^*$	0.921	0.924	$1^+2^+3^*$
250	0.171	0.135	$2^-3^-4^-$	0.774	0.851	$1^+3^-4^-$	0.928	0.918	$1^+2^+4^*$	0.911	0.915	$1^+2^+3^*$
260	0.114	0.075	$2^-3^-4^-$	0.765	0.832	$1^+3^-4^-$	0.925	0.909	$1^+2^+4^*$	0.914	0.904	$1^+2^+3^*$
270	0.089	0.061	$2^-3^-4^-$	0.757	0.823	$1^+3^-4^-$	0.911	0.893	$1^+2^+4^*$	0.896	0.886	$1^+2^+3^*$
280	0.172	0.143	$2^-3^-4^-$	0.76	0.828	$1^+3^-4^-$	0.907	0.897	$1^+2^+4^*$	0.886	0.885	$1^+2^+3^*$
290	0.14	0.083	$2^-3^-4^-$	0.826	0.842	$1^+3^-4^-$	0.912	0.878	$1^+2^+4^+$	0.9	0.874	$1^+2^+3^-$
300	0.272	0.235	$2^-3^-4^-$	0.825	0.825	$1^+3^-4^-$	0.902	0.856	$1^+2^+4^*$	0.895	0.857	$1^+2^+3^*$
310	0.191	0.156	$2^-3^-4^-$	0.776	0.777	$1^+3^-4^*$	0.862	0.814	$1^+2^+4^*$	0.844	0.806	$1^+2^*3^*$
320	0.099	0.051	$2^-3^-4^-$	0.478	0.424	$1^+3^-4^*$	0.611	0.489	$1^+2^+4^*$	0.591	0.478	$1^+2^*3^*$
330	0.169	0.135	$2^-3^-4^-$	0.544	0.503	$1^+3^-4^*$	0.666	0.56	$1^+2^+4^*$	0.643	0.547	$1^+2^*3^*$
340	0.182	0.129	$2^-3^-4^-$	0.627	0.562	$1^+3^-4^-$	0.73	0.611	$1^+2^+4^*$	0.717	0.603	$1^+2^+3^*$
350	0.157	0.113	$2^-3^-4^-$	0.534	0.496	$1^+3^-4^*$	0.61	0.532	$1^+2^+4^*$	0.605	0.531	$1^+2^*3^*$
360	0.089	0.047	$2^-3^-4^-$	0.531	0.501	$1^+3^-4^*$	0.606	0.537	$1^+2^+4^*$	0.6	0.535	$1^+2^*3^*$
370	0.156	0.11	$2^-3^-4^-$	0.425	0.339	$1^+3^-4^-$	0.535	0.394	$1^+2^+4^*$	0.529	0.392	$1^+2^+3^*$
380	0.161	0.121	$2^-3^-4^-$	0.437	0.344	$1^+3^-4^*$	0.498	0.375	$1^+2^+4^*$	0.491	0.372	$1^+2^*3^*$

Table 2: The diversity obtained from the algorithms using H_1 as the fitness function. The variables appear in clauses based on a uniform distribution with $n = 100$ and $k = 3$. The notations are in line with Table 1

	Basic 1			Bit-flip 2			EDO 3 Mutation			EDO 3 Crossover+Mutation		
m	H_1	H_2	Stat (1)	H_1	H_2	Stat (2)	H_1	H_2	Stat (3)	H_1	H_2	Stat (4)
270	0.295	0.28	$2^-3^-4^-$	0.859	0.889	$1^+3^-4^-$	0.942	0.947	$1^+2^+4^*$	0.94	0.948	$1^+2^+3^*$
280	0.241	0.217	$2^-3^-4^-$	0.867	0.879	$1^+3^-4^-$	0.944	0.943	$1^+2^+4^*$	0.944	0.946	$1^+2^+3^*$
290	0.202	0.186	$2^-3^-4^-$	0.834	0.848	$1^+3^-4^-$	0.937	0.938	$1^+2^+4^*$	0.939	0.941	$1^+2^+3^*$
300	0.183	0.175	$2^-3^-4^-$	0.877	0.888	$1^+3^-4^-$	0.943	0.943	$1^+2^+4^*$	0.946	0.946	$1^+2^+3^*$
310	0.09	0.078	$2^-3^-4^-$	0.875	0.893	$1^+3^-4^-$	0.943	0.946	$1^+2^+4^*$	0.945	0.948	$1^+2^+3^*$
320	0.062	0.051	$2^-3^-4^-$	0.884	0.894	$1^+3^-4^-$	0.936	0.939	$1^+2^+4^*$	0.937	0.94	$1^+2^+3^*$
330	0.157	0.137	$2^-3^-4^-$	0.885	0.895	$1^+3^-4^-$	0.927	0.927	$1^+2^+4^*$	0.932	0.934	$1^+2^+3^*$
340	0.135	0.117	$2^-3^-4^-$	0.898	0.905	$1^+3^-4^-$	0.928	0.927	$1^+2^+4^*$	0.933	0.933	$1^+2^+3^*$
350	0.073	0.062	$2^-3^-4^-$	0.895	0.903	$1^+3^-4^-$	0.916	0.918	$1^+2^+4^*$	0.918	0.92	$1^+2^+3^*$
360	0.08	0.067	$2^-3^-4^-$	0.866	0.875	$1^+3^-4^-$	0.893	0.896	$1^+2^+4^*$	0.898	0.903	$1^+2^+3^*$
370	0.084	0.07	$2^-3^-4^-$	0.851	0.862	$1^+3^-4^-$	0.884	0.886	$1^+2^+4^*$	0.891	0.895	$1^+2^+3^*$
380	0.058	0.042	$2^-3^-4^-$	0.846	0.855	$1^+3^-4^-$	0.876	0.879	$1^+2^+4^*$	0.877	0.88	$1^+2^+3^*$
390	0.178	0.178	$2^-3^-4^-$	0.822	0.822	$1^+3^*4^-$	0.832	0.829	$1^+2^*4^*$	0.835	0.832	$1^+2^+3^*$
400	0.226	0.215	$2^-3^-4^-$	0.637	0.622	$1^+3^-4^-$	0.648	0.63	$1^+2^+4^*$	0.647	0.629	$1^+2^+3^*$
410	0.105	0.098	$2^-3^-4^-$	0.674	0.669	$1^+3^-4^-$	0.693	0.685	$1^+2^+4^*$	0.693	0.684	$1^+2^+3^*$
420	0.125	0.118	$2^-3^-4^-$	0.603	0.592	$1^+3^*4^-$	0.612	0.599	$1^+2^*4^*$	0.613	0.6	$1^+2^+3^*$
430	0.153	0.146	$2^-3^-4^-$	0.311	0.299	$1^+3^*4^-$	0.326	0.309	$1^+2^*4^*$	0.326	0.309	$1^+2^+3^*$
440	0.059	0.047	$2^-3^-4^-$	0.352	0.335	$1^+3^-4^-$	0.366	0.346	$1^+2^+4^*$	0.366	0.347	$1^+2^+3^*$

Table 3: The diversity obtained from the algorithms using H_2 as the fitness function on the same instances in Table 1. The Kruskal-Wallis statistical test is conducted on H_2. The notations are in line with Table 1

	Basic 1			Bit-flip 2			EDO 3 Mutation			EDO 3 Crossover+Mutation		
m	H_2	H_1	Stat (1)	H_2	H_1	Stat (2)	H_2	H_1	Stat (3)	H_2	H_1	Stat (4)
210	0.016	0.055	$2^-3^-4^-$	0.849	0.732	$1^+3^-4^-$	0.965	0.944	$1^+2^+4^*$	0.957	0.912	$1^+2^+3^*$
220	0.011	0.052	$2^-3^-4^-$	0.83	0.709	$1^+3^-4^-$	0.95	0.928	$1^+2^+4^*$	0.939	0.892	$1^+2^+3^*$
230	0.019	0.055	$2^-3^-4^-$	0.836	0.719	$1^+3^-4^-$	0.941	0.911	$1^+2^+4^+$	0.932	0.882	$1^+2^+3^-$
240	0.007	0.046	$2^-3^-4^-$	0.825	0.706	$1^+3^-4^-$	0.936	0.896	$1^+2^+4^*$	0.929	0.874	$1^+2^+3^*$
250	0.135	0.171	$2^-3^-4^-$	0.859	0.757	$1^+3^-4^-$	0.927	0.909	$1^+2^+4^*$	0.918	0.874	$1^+2^+3^*$
260	0.075	0.114	$2^-3^-4^-$	0.845	0.751	$1^+3^-4^-$	0.919	0.906	$1^+2^+4^*$	0.911	0.872	$1^+2^+3^*$
270	0.061	0.089	$2^-3^-4^-$	0.838	0.737	$1^+3^-4^-$	0.907	0.89	$1^+2^+4^+$	0.895	0.844	$1^+2^+3^-$
280	0.143	0.172	$2^-3^-4^-$	0.842	0.74	$1^+3^-4^-$	0.906	0.877	$1^+2^+4^*$	0.895	0.84	$1^+2^+3^*$
290	0.083	0.14	$2^-3^-4^-$	0.861	0.807	$1^+3^-4^-$	0.895	0.887	$1^+2^+4^+$	0.887	0.864	$1^+2^+3^-$
300	0.235	0.272	$2^-3^-4^-$	0.835	0.813	$1^+3^-4^-$	0.865	0.884	$1^+2^+4^*$	0.865	0.867	$1^+2^+3^*$
310	0.156	0.191	$2^-3^-4^-$	0.786	0.762	$1^+3^-4^*$	0.824	0.845	$1^+2^+4^*$	0.816	0.815	$1^+2^*3^*$
320	0.051	0.099	$2^-3^-4^-$	0.434	0.468	$1^+3^-4^*$	0.494	0.599	$1^+2^+4^*$	0.481	0.558	$1^+2^*3^*$
330	0.135	0.169	$2^-3^-4^-$	0.513	0.534	$1^+3^-4^*$	0.562	0.647	$1^+2^+4^*$	0.551	0.61	$1^+2^*3^*$
340	0.129	0.182	$2^-3^-4^-$	0.57	0.617	$1^+3^-4^*$	0.616	0.718	$1^+2^+4^*$	0.606	0.69	$1^+2^*3^*$
350	0.113	0.157	$2^-3^-4^-$	0.505	0.524	$1^+3^-4^*$	0.537	0.598	$1^+2^+4^*$	0.534	0.587	$1^+2^*3^*$
360	0.047	0.089	$2^-3^-4^-$	0.504	0.524	$1^+3^-4^*$	0.541	0.602	$1^+2^+4^*$	0.536	0.589	$1^+2^*3^*$
370	0.11	0.156	$2^-3^-4^-$	0.342	0.42	$1^+3^-4^-$	0.396	0.53	$1^+2^+4^*$	0.392	0.521	$1^+2^+3^*$
380	0.121	0.161	$2^-3^-4^-$	0.347	0.432	$1^+3^-4^*$	0.377	0.494	$1^+2^+4^*$	0.374	0.484	$1^+2^*3^*$

Table 4: The diversity obtained from the algorithms using H_2 the fitness on the same instances in Table 2. The notations are in line with Table 3.

	Basic 1			Bit-flip 2			EDO 3 Mutation			EDO 3 Crossover+Mutation		
m	H_2	H_1	Stat (1)	H_2	H_1	Stat (2)	H_2	H_1	Stat (3)	H_2	H_1	Stat (4)
270	0.28	0.295	$2^-3^-4^-$	0.891	0.849	$1^+3^-4^-$	0.95	0.933	$1^+2^+4^*$	0.946	0.923	$1^+2^+3^*$
280	0.217	0.241	$2^-3^-4^-$	0.881	0.863	$1^+3^-4^-$	0.946	0.937	$1^+2^+4^*$	0.947	0.934	$1^+2^+3^*$
290	0.186	0.202	$2^-3^-4^-$	0.849	0.83	$1^+3^-4^-$	0.939	0.931	$1^+2^+4^*$	0.942	0.93	$1^+2^+3^*$
300	0.175	0.183	$2^-3^-4^-$	0.891	0.871	$1^+3^-4^-$	0.945	0.939	$1^+2^+4^*$	0.947	0.936	$1^+2^+3^*$
310	0.078	0.09	$2^-3^-4^-$	0.897	0.874	$1^+3^-4^-$	0.945	0.933	$1^+2^+4^*$	0.95	0.937	$1^+2^+3^*$
320	0.051	0.062	$2^-3^-4^-$	0.895	0.881	$1^+3^-4^-$	0.938	0.929	$1^+2^+4^*$	0.94	0.928	$1^+2^+3^*$
330	0.136	0.157	$2^-3^-4^-$	0.897	0.882	$1^+3^-4^-$	0.93	0.923	$1^+2^+4^*$	0.934	0.923	$1^+2^+3^*$
340	0.117	0.135	$2^-3^-4^-$	0.907	0.895	$1^+3^-4^-$	0.932	0.927	$1^+2^+4^*$	0.934	0.925	$1^+2^+3^*$
350	0.062	0.073	$2^-3^-4^-$	0.904	0.892	$1^+3^-4^-$	0.919	0.911	$1^+2^+4^*$	0.922	0.913	$1^+2^+3^*$
360	0.067	0.08	$2^-3^-4^-$	0.879	0.865	$1^+3^-4^-$	0.897	0.889	$1^+2^+4^*$	0.904	0.894	$1^+2^+3^*$
370	0.07	0.084	$2^-3^-4^-$	0.866	0.851	$1^+3^-4^-$	0.892	0.882	$1^+2^+4^*$	0.896	0.884	$1^+2^+3^*$
380	0.042	0.058	$2^-3^-4^-$	0.858	0.843	$1^+3^-4^-$	0.878	0.867	$1^+2^+4^*$	0.881	0.869	$1^+2^+3^*$
390	0.178	0.178	$2^-3^-4^-$	0.824	0.818	$1^+3^*4^-$	0.832	0.829	$1^+2^*4^*$	0.834	0.83	$1^+2^+3^*$
400	0.214	0.226	$2^-3^-4^-$	0.623	0.636	$1^+3^-4^-$	0.631	0.646	$1^+2^+4^*$	0.631	0.645	$1^+2^+3^*$
410	0.098	0.105	$2^-3^-4^-$	0.672	0.673	$1^+3^-4^-$	0.684	0.688	$1^+2^+4^*$	0.684	0.686	$1^+2^+3^*$
420	0.118	0.125	$2^-3^-4^-$	0.593	0.601	$1^+3^-4^*$	0.602	0.611	$1^+2^+4^*$	0.601	0.61	$1^+2^*3^*$
430	0.146	0.153	$2^-3^-4^-$	0.3	0.311	$1^+3^*4^-$	0.309	0.325	$1^+2^*4^*$	0.309	0.326	$1^+2^+3^*$
440	0.047	0.059	$2^-3^-4^-$	0.336	0.352	$1^+3^-4^-$	0.347	0.366	$1^+2^+4^*$	0.347	0.366	$1^+2^+3^*$

both crossover and mutation slightly outperforms the other one in terms of H_1. We can get diverse sets of SAT assignments with more than 90% diversity in terms of H_1 with the EDO algorithm in cases $m \leq 360$.

4.2 Comparison of algorithms employing H_2 as the fitness

We examine the algorithms' performance when H_2 is incorporated as the fitness function. The H_2 differs from H_1 in focusing on the variables with more appearances in Φ. Table 3 and 4 summarise the algorithms' results in the power law and uniform instances, respectively. Since Algorithm 2 does not use any diversity measures inside of the algorithm, the results are the same as those of Table 1 and 2. Nevertheless, the other algorithms' results in Table 3 and 4 are different to those in Table 1 and 2. As expected, the diversity of assignments slightly increases in terms of H_2, while there is a minor drop in H_1 values. The change is plausible since we incorporated H_2 into the algorithms as the fitness function instead of H_1.

One may observe that increasing $\frac{m}{n}$ affects the capability of the introduced algorithms in terms of H_2 more than it does in terms of H_1. This is because, in a limited feasible region, the more frequent variables are more likely to be fixed at true or false. Since those variables have a higher weight in the diversity calculation, increases in $\frac{m}{n}$ make it challenging to diversify solutions in terms of H_2. For instance, Table 3 indicates that the H_2 values drop from 0.96 to 0.38 for the EDO algorithm using mutation, while the same sets of solutions result in less severe decreases in H_1 values (from 0.94 to 0.5).

Table 3 also indicates that the gap between the results of the EDO algorithms' variants is more profound when H_2 is used as the fitness function. The statistical tests also confirm the difference in favour of the variant employing the mutation in instances where $m = \{230, 270, 290\}$. However, it is the other way around in the uniform instances; the variant that benefits from the crossover performs slightly better, although the difference is statically insignificant. The same observation we had when H_1 was incorporated into the algorithm as the fitness function.

4.3 Investigation on Unsatisfiablity

This subsection studies the correlation between the obtained diversity and the number of unsatisfiable formulas generated during the search. The introduced algorithms, as mentioned, modify the formula Φ to generate a new assignment in each iteration. Although Φ is a true formula, it is likely to make it false via modifications during the search. We consider Algorithm 2 for this purpose since the algorithm does not have any hyper-parameters affecting the results.

Figure 2 depicts the trajectories of diversity and the false Φ generated by Algorithm 2. Note that we normalise the values to plot them in a figure. As expected, the algorithm generates the minimum number of false formulas (false Φ) when $\frac{m}{n}$ is low. Low values of $\frac{m}{n}$ often lead to large feasible regions and, consequently, a larger room to diversify the solutions. In such cases, the modifications of Algorithm 2 are not large enough to cause unsatisfiability for Φ. If $\frac{m}{n}$ gets sufficiently large, so does the feasible region get more limited, affecting both the diversity and satisfiability rate. Although

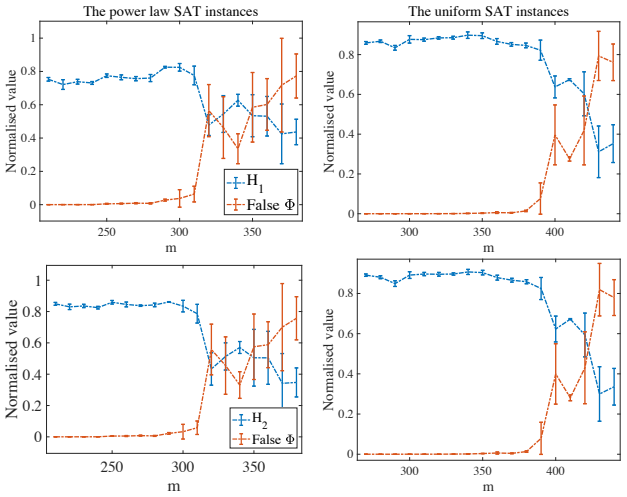


Figure 2: The representative trajectories of the bit-flip algorithm's the diversity and the number of false Φ. In the first row, H_1 serves as the fitness function, while it is H_2 in the second row.

a disproportional relationship between diversity and unsatisfiability is expected, the figure interestingly depicts a symmetric behaviour. The trajectories are pretty similar for H_1 and H_2. The sole difference is the range of H_1 and H_2 in the power law instances, where H_2 starts and finishes at slightly higher values.

5 CONCLUSION

This study presented evolutionary approaches to construct a diverse set of solutions in SAT using the well-known SAT solver, minisat. We first defined two measures to quantify the diversity of solutions. One, which considers and one, which dismisses the frequency of variable appearances in clauses. Then, we introduced two EAs, employing the EDO principle to construct a diverse set of SAT assignments. The EAs iteratively make modifications on a given SAT instance, then solve it with a well-known solver, minisat. Finally, we conducted a comprehensive experimental investigation to assess the algorithms' performance and study the solution space and unsatisfiability rate. The results indicate the capability of the introduced algorithm to compute highly diverse sets of SAT solutions.

For future studies, it is intriguing to study more complicated EAs, like $(\mu + \lambda)$-EAs. Although it is challenging in diversity problems to select the next generation when λ is larger than one, an increase in λ can potentially improve the algorithms' performance. Another possible extension is to study other related problems, such as MaxSAT.

ACKNOWLEDGEMENTS

This work was supported by the Australian Research Council through grants DP190103894 and FT200100536.

REFERENCES

[1] Bradley Alexander, James Kortman, and Aneta Neumann. 2017. Evolution of artistic image variants through feature based diversity optimisation. In *GECCO*. ACM, 171–178.

[2] Maxime Allard, Simón C. Smith, Konstantinos I. Chatzilygeroudis, and Antoine Cully. 2022. Hierarchical quality-diversity for online damage recovery. In *GECCO*. ACM, 58–67.

[3] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. 2009. Towards Industrial-Like Random SAT Instances. In *IJCAI*. 387–392.

[4] Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. 2019. Evolving diverse TSP instances by means of novel and creative mutation operators. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. 58–71. https://doi.org/10.1145/3299904.3340307

[5] Jakob Bossek, Aneta Neumann, and Frank Neumann. 2021. Breeding diverse packings for the knapsack problem by means of diversity-tailored evolutionary algorithms. In *GECCO*. ACM, 556–564.

[6] Jakob Bossek and Frank Neumann. 2021. Evolutionary diversity optimization and the minimum spanning tree problem. In *GECCO*. ACM, 198–206.

[7] Jakob Bossek and Frank Neumann. 2022. Exploring the feature space of TSP instances using quality diversity. In *GECCO*. ACM, 186–194.

[8] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *TACAS (Lecture Notes in Computer Science, Vol. 9035)*. Springer, 304–319.

[9] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *STOC*. ACM, 151–158.

[10] Martin Davis, George Logemann, and Donald W. Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.

[11] Anh Viet Do, Jakob Bossek, Aneta Neumann, and Frank Neumann. 2020. Evolving diverse sets of tours for the travelling salesperson problem. In *GECCO*. ACM, 681–689.

[12] Anh Viet Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. 2022. Analysis of Evolutionary Diversity Optimization for Permutation Problems. *ACM Trans. Evol. Learn. Optim.* 2, 3 (2022), 11:1–11:27.

[13] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient sampling of SAT solutions for testing. In *ICSE*. ACM, 549–559.

[14] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *SAT (Lecture Notes in Computer Science, Vol. 2919)*. Springer, 502–518.

[15] Matthew C. Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K. Hoover, and Stefanos Nikolaidis. 2021. Illuminating Mario Scenes in the Latent Space of a Generative Adversarial Network. In *AAAI*. AAAI Press, 5922–5930.

[16] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 2020. Covariance matrix adaptation for the rapid illumination of behavior space. In *GECCO*. ACM, 94–102.

[17] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, Thomas Sauerwald, and Andrew M. Sutton. 2017. Bounds on the Satisfiability Threshold for Power Law Distributed Random SAT. In *ESA (LIPIcs, Vol. 87)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 37:1–37:15.

[18] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, and Andrew M. Sutton. 2017. Phase Transitions for Scale-Free SAT Formulas. In *AAAI*. AAAI Press, 3893–3899.

[19] Wanru Gao, Samadhi Nallaperuma, and Frank Neumann. 2021. Feature-Based Diversity Optimization for Problem Instance Classification. *Evol. Comput.* 29, 1 (2021), 107–128.

[20] Xiaodong Li, Michael G. Epitropakis, Kalyanmoy Deb, and Andries P. Engelbrecht. 2017. Seeking Multiple Solutions: An Updated Survey on Niching Methods and Their Applications. *IEEE Trans. Evol. Comput.* 21, 4 (2017), 518–538.

[21] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference DAC*. ACM, 530–535.

[22] Alexander Nadel. 2011. Generating Diverse Solutions in SAT. In *SAT (Lecture Notes in Computer Science, Vol. 6695)*. Springer, 287–301.

[23] Aneta Neumann, Denis Antipov, and Frank Neumann. 2022. Coevolutionary Pareto diversity optimization. In *GECCO*. ACM, 832–839.

[24] Aneta Neumann, Jakob Bossek, and Frank Neumann. 2021. Diversifying greedy sampling and evolutionary diversity optimisation for constrained monotone submodular functions. In *GECCO*. ACM, 261–269.

[25] Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. 2018. Discrepancy-based evolutionary diversity optimization. In *GECCO*. ACM, 991–998.

[26] Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. 2019. Evolutionary diversity optimization using multi-objective indicators. In *GECCO*. ACM, 837–845.

[27] Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. 2021. Computing diverse sets of high quality TSP tours by EAX-based evolutionary diversity optimisation. In *FOGA*. ACM, 9:1–9:11.

[28] Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. 2021. Entropy-based evolutionary diversity optimisation for the traveling salesperson problem. In *GECCO*. ACM, 600–608.

[29] Adel Nikfarjam, Anh Viet Do, and Frank Neumann. 2022. Analysis of Quality Diversity Algorithms for the Knapsack Problem. In *PPSN (2) (Lecture Notes in Computer Science, Vol. 13399)*. Springer, 413–427.

[30] Adel Nikfarjam, Amirhossein Moosavi, Aneta Neumann, and Frank Neumann. 2022. Computing High-Quality Solutions for the Patient Admission Scheduling Problem Using Evolutionary Diversity Optimisation. In *PPSN (1) (Lecture Notes in Computer Science, Vol. 13398)*. Springer, 250–264.

[31] Adel Nikfarjam, Aneta Neumann, Jakob Bossek, and Frank Neumann. 2022. Co-evolutionary Diversity Optimisation for the Traveling Thief Problem. In *PPSN (1) (Lecture Notes in Computer Science, Vol. 13398)*. Springer, 237–249.

[32] Adel Nikfarjam, Aneta Neumann, and Frank Neumann. 2022. Evolutionary diversity optimisation for the traveling thief problem. In *GECCO*. ACM, 749–756.

[33] Adel Nikfarjam, Aneta Neumann, and Frank Neumann. 2022. On the use of quality diversity algorithms for the traveling thief problem. In *GECCO*. ACM, 260–268.

[34] Nemanja Rakicevic, Antoine Cully, and Petar Kormushev. 2021. Policy manifold search: exploring the manifold hypothesis for diversity-based neuroevolution. In *GECCO*. ACM, 901–909.

[35] João P. Marques Silva and Karem A. Sakallah. 1999. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers* 48, 5 (1999), 506–521.

[36] Kirby Steckel and Jacob Schrum. 2021. Illuminating the space of beatable lode runner levels produced by various generative adversarial networks. In *GECCO Companion*. ACM, 111–112.

[37] Tamara Ulrich and Lothar Thiele. 2011. Maximizing population diversity in single-objective optimization. In *GECCO*. ACM, 641–648.

[38] Enrico Zardini, Davide Zappetti, Davide Zambrano, Giovanni Iacca, and Dario Floreano. 2021. Seeking quality diversity in evolutionary co-design of morphology and control of soft tensegrity modular robots. In *GECCO*. ACM, 189–197.