

# Looking for novelty in SBSE problems

José Raúl Romero<sup>1</sup>, Aurora Ramírez<sup>1,2</sup>, and Christopher L. Simons<sup>3</sup>

<sup>1</sup> University of Córdoba, Spain\*\*

{jrromero,aramirez}@uco.es

<sup>2</sup> University of Málaga, Spain

<sup>3</sup> University of the West of England, Bristol, United Kingdom

Chris.Simons@uwe.ac.uk

**Abstract.** Search-based software engineering (SBSE) was conceived to support engineers searching for innovative ideas to solve recurrent software engineering problems along the software project lifecycle. However, current approaches require the engineer to formulate and quantify their search objectives, which may be challenging. As SBSE consolidates as a discipline, problems become more demanding, and consequently the definition of the search problem and the characteristics of the search space remain oversimplified. Thus the evaluation of problem solutions by means of a fitness function could be failing to address essential aspects that can cause disappointment for the engineer after reaching final results. This position paper launches the idea that novelty search opens up a new scenario, as it rewards solution novelty, a concept mapping to problem characteristics other than fitness and whose definition might be more intuitive to the engineer. We explore its applicability to SBSE and discuss some preliminary findings of interest to the SBSE community.

**Keywords:** novelty search · search-based software engineering · next release problem

## 1 Introduction and motivation

In the field of Search-based Software Engineering (SBSE), search-based approaches have been applied to a range of activities within the software project lifecycle [4]. The underlying idea has always been to support the engineer in the generation of rigorous, alternative—even unexpected— solutions. A range of search methods have been applied, all conceived to return potential solutions compliant with the software engineering problem. However, they all require deep knowledge on the problem formulation, and evolutionary algorithms have emerged where fitness measures are necessary to select superior individual candidates for the next generation of the population. Moreover, it can be challenging to formulate a suitable fitness function. Even if this is achieved, solutions derived in this way can look ‘artificial’ to the software engineer due to problem

\*\* Partially supported by the Spanish Ministry of Economy and Competitiveness [project TIN2017-83445-P].

---

**Algorithm 1** General procedure of the novelty search algorithm

---

**Input:** BC, population\_size, archive\_size, archive\_threshold**Output:** archive

```
1: initialize population at random
2: evaluate each individual for novelty
3: while termination condition not reached do
4:   select parents based on novelty
5:   recombine pairs of parents
6:   mutate new individuals
7:   evaluate each offspring for novelty
8:   select candidates for novelty archive
9:   replace population with offspring
10: end while
11: return archive
```

---

oversimplification. This mismatch between the real problem and its formulation increases as the industry demands more complex solutions.

SBSE should produce real but ‘unexpected’ solutions, and subsequently present options to the software engineer for their understanding and evaluation. Since its inception, SBSE already required mechanisms to guarantee the ‘novelty’ of solutions compared to those produced by analytical procedures. Thus SBSE can be thought of as an opportunity to discover innovative, ‘realistic’ solutions in feasible time. Approaches have appeared in other problem domains wherein fitness-based optimization is abandoned in favour of the role of diversity as found in natural evolution instead [6]. Indeed, novelty search (NS) seems to fit the precepts of SBSE, since it (1) does not require a fitness function, (2) rewards not-previously-found solutions, and (3) develops an archive of solutions from which a working solution can be determined by human judgment. However, proposals relating to the application of NS in SBSE are not abundant. In this paper, we argue that NS could open up a new scenario, thus we explore its applicability to SBSE and discuss some preliminary findings of interest to the SBSE community.

## 2 Theoretical foundations

Novelty search uses an open-ended discovery process in the search for novel solutions, wherein novelty implies being different from any solution discovered beforehand. The novelty of an individual is derived from its *behavior*, represented as a *behavior characterization* (BC), i.e. a set of features based on the genotype, the phenotype or other aspects of the problem domain [5]. The selected BC thus defines a space of all possible behaviors of interest to be searched. The *novelty score* of an individual is measured with respect to others in the evolving population and its predecessors [6] which are stored in an archive. An example of a novelty score is the average distance to the  $k$ -nearest neighbors of the individual.

For readers aware of classical objective fitness-based evolutionary algorithms, the general structure of a basic NS algorithm might look familiar. Algorithm 1

(inspired by [3]) illustrates NS. After random initialization and evaluation — in terms of novelty — of a population of size *population\_size* (lines 1-2), NS selects individuals based on their solution novelty (line 4). Genetic operators are applied to parents and the newly created solutions are evaluated for their novelty too (lines 5-7). Next, an archive stores individuals whose novelty score exceeds *archive\_threshold* (line 8). Usually, a maximum archive size, *archive\_size*, is set to limit the number of returned solutions. Then in line 9, the entire population is replaced after each generation as a way to promote the search for novelty. While NS is typically open-ended, it is possible to apply a termination criterion. Finally, the *archive* is returned.

As for the application of NS (or variants) to SBSE, only a limited number of preliminary, domain-specific proposals can be found in the literature so far, focused on search-based testing (e.g. [2]) and Genetic Improvement (e.g. [7]).

### 3 An illustrative example: the Next Release Problem

This section explores how novelty search can be applied to the next release problem (NRP), which looks for determining the software features that should be added to the system as part of the following release [1]. The first step is to identify and encode the optimization problem. Here,  $n$  requirements need to be selected by considering customers' preferences. Every requirement  $r_j$  has a cost, while each customer expresses interest in some requirements,  $R_i$ , and indicates a weight to represent their importance. A solution  $x$  is then encoded as a binary array of size  $n$ , where  $x_j$  indicates whether  $r_j$  appears in the release.

Secondly, at least one behavior characterization should be formulated, depending on the problem elements becoming an indicator of solution novelty. In the NRP, the focus could be put on the two most relevant elements for the problem: requirements and customers. Consequently, two distinct BCs could be defined to explore novelty: promoting the selection of pairs of requirements that are not often combined (Eq. 1); and preferably meeting the needs of customers with smaller weights or demanding requirements with a lower demand (Eq. 2).

$$BC_{req}(r_j, r_k) = \{1 \text{ if } x_j = 1 \wedge x_k = 1; 0 \text{ otherwise}\}. \quad (1)$$

$$BC_{cust}(i) = \{1 \text{ if } \exists r_j \in R_i \rightarrow x_j = 1; 0 \text{ otherwise}\} \quad (2)$$

Next, we need to provide a measure to evaluate the solution novelty. For our example, given the binary nature of the encoding structure, Hamming distance seems appropriate. Following with the procedure, we should determine the termination criteria. Goal states (e.g. finding a particular behaviour) and/or computational budgets (e.g. number of iterations) are valid. Thus, for the NRP, reaching 100 iterations will terminate the execution. Note that within the loop of NS, and similarly to other evolutionary approaches, genetic operators are required for crossover and mutation. In this case, basic operators are used (uniform crossover and single-point mutation, respectively).

## 4 Experimentation and Discussion

As for the implementation and execution of a preliminary approach, note that a genetic algorithm (GA) has been also implemented and executed for comparison. Due to space restrictions, numerical results are omitted to promote discussion in this position paper. Experiments reveal that NS makes less predictable decisions during the search compared to other metaheuristics. While fitness-oriented algorithms will converge to an optimum once a promising search path is found, NS is endlessly jumping from one solution to another. Consequently, there is no assumption of how the ‘best solution’ looks like, as no objective fitness-based optimum exists. In the NRP, this is reflected in how the search space is explored. While the GA focuses on specific paths, NS significantly diversifies the exploration, leading to equal chances for each requirement to appear in the release and no particular customer being prioritized. Another effect is that the NS archive contains a more diverse set of distinct solutions. In fact, the definition of the BC is crucial as this determines how the search behaves relating to exploring the search space. It cannot be translated in finding better (optimal) solutions than GA, which is adjusted to the goal expressed as an objective measure (in case the problem can be effectively represented by an objective measure).

Conversely, the NS strategy faces challenges that should nevertheless be taken into consideration. The first challenge relates to the size of the behavioral space. In the NRP, the number of requirement combinations could drastically grow, making it difficult to inspect them all. If so, NS behaves similarly to a GA, which cannot converge so quickly. A second challenge relates to the presence of hard constraints, since feasible regions are more difficult to locate. Finally, both the dimension of the BC and the distance selected to compute novelty scores have been identified as key issues effecting the execution time.

## References

1. Bagnall, A., Rayward-Smith, V., Whittle, I.: The next release problem. *Information and Software Technology* **43**(14), 883–890 (2001)
2. Boussaa, M., Barais, O., Sunyé, G., Baudry, B.: A novelty search approach for automatic test data generation. In: *Proceedings of the 8th International Workshop on Search-Based Software Testing*. pp. 40–43. IEEE Press (2015)
3. Eiben, A.E., Smith, J.E., et al.: *Introduction to evolutionary computing*, vol. 53. Springer (2003)
4. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys* **45**(1), 11 (2012)
5. Kistemaker, S., Whiteson, S.: Critical Factors in the Performance of Novelty Search. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. pp. 965–972. ACM (2011)
6. Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the Search for Novelty Alone. *Evolutionary Computation* **19**(2), 189–223 (2011)
7. López-López, V.R., Trujillo, L., Legrand, P.: Novelty Search for Software Improvement of a SLAM System. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. pp. 1598–1605. ACM (2018)