Synthesising Diverse and Discriminatory Sets of Instances using Novelty Search in Combinatorial Domains

Alejandro Marrero amarrerd@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Eduardo Segredo esegredo@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Coromoto León cleon@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna, San Cristóbal de La Laguna, Spain

Emma Hart e.hart@napier.ac.uk
School of Computing, Engineering and the Built Environment,
Edinburgh Napier University, Edinburgh, United Kingdom

Abstract

Gathering sufficient instance data to either train algorithm-selection models or understand algorithm footprints within an instance space can be challenging. We propose an approach to generating synthetic instances that are tailored to perform well with respect to a target algorithm belonging to a predefined portfolio but are also diverse with respect to their features. Our approach uses a novelty search algorithm with a linearly

weighted fitness function that balances novelty and performance to generate a large set of diverse and discriminatory instances in a single run of the algorithm. We consider two definitions of novelty: (1) with respect to discriminatory performance within a portfolio of solvers; (2) with respect to the features of the evolved instances. We evaluate the proposed method with respect to its ability to generate diverse and discriminatory instances in two domains (knapsack and bin-packing), comparing to another well-known quality diversity method, Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) and an evolutionary algorithm that only evolves for discriminatory behaviour. The results demonstrate that the novelty search method outperforms its competitors in terms of coverage of the space and its ability to generate instances that are diverse regarding the relative size of the "performance gap" between the target solver and the remaining solvers in the portfolio. Moreover, for the Knapsack domain, we also show that we are able to generate novel instances in regions of an instance space not covered by existing benchmarks using a portfolio of state-of-the-art solvers. Finally, we demonstrate that the method is robust to different portfolios of solvers (stochastic approaches, deterministic heuristics and state-of-the-art methods), thereby providing further evidence of its generality.

Keywords

Instance Generation, Quality Diversity, Novelty Search, MAP-Elites, Knapsack Problem, Bin Packing Problem.

1 Introduction

In any optimisation domain, it is well known that no single solver can best solve all instances, necessitating the use of algorithm-portfolios which collectively provide coverage of the problem-space. This leads to the need for per-instance algorithm-selection, i.e. choosing the best solver from a portfolio for a given instance, a topic first proposed by Rice (1976) that has garnered considerable attention over recent years (Kerschke et al., 2019). Many algorithm-selection approaches rely on training a machine-learning algorithm to predict either performance of a given algorithm or the label of the best solver, using a large set of representative instances from the domain for training. However, acquiring sufficient instances that both cover the feature-space of instances and are discriminatory with respect to the solvers in the portfolio is challenging. Previous work in the field of *instance space analysis (ISA)* that provides 2D visualisations of an

instance space (Smith-Miles, 2009) has revealed that when these spaces are constructed using readily available benchmarks, there are regions of the instance space that are completely lacking in instances, while for some solvers, the region in which they perform well is sparsely covered (Smith-Miles et al., 2010; Smith-Miles and Bowly, 2015). Real-world instances can also be used to augment synthetic datasets. However, they can be hard to come by and each one often only covers a small but unique region of a space due to high-levels of structure specific to the dataset (Hains et al., 2012). Where gaps in the space exist, it thus remains unclear whether this is simply due to a lack of real-data or whether the gaps are related to regions that are not representative of real-world instances.

To address this, attention has been given to find new instances to fill these gaps. For example, space-filling approaches such as those described by Smith-Miles *et al.* (Smith-Miles et al., 2010; Smith-Miles and Bowly, 2015) directly attempt to fill gaps in the feature-space in the Travelling Salesman Problem (TSP) and Graph Colouring domains, but do not account for discriminatory behaviour. On the other hand, other research has focused on evolving new instances that are maximally discriminative with respect to a portfolio of solvers (Alissa et al., 2019; Bossek et al., 2019; Plata-González et al., 2019), i.e. maximise the performance-gap between a target and other solver (respectively in the Bin-Packing, TSP and Knapsack domains), but these methods tend not to have explicit mechanisms for creating instances that are diverse with respect to the feature-space.

We argue that in to underpin future work in algorithm-selection and in understanding algorithm strengths and weaknesses, better approaches to filling instance spaces are required. Specifically, we propose that new datasets are needed that: (1) cover a high proportion of the *feature-space* relevant to a domain; (2) contain instances on which the portfolio of solvers of interest exhibit discriminatory performance; (3) contain instances that highlight diversity in the *performance-space*, i.e. highlight a range of values for the performance-gap between the winning solver and the next best solver, in order to gain further insight into the relative performance of different solvers. In particular, the latter point has rarely been addressed: most approaches to evolving dis-

criminatory instances attempt to maximise the gap between a solver and the next-best method, resulting in instances that reflect the extremes of the regions of strength of a solver. However, it is clearly important to also find those all regions in which a solver outperforms another, not just the extremes.

The main contribution of our work is therefore in proposing an approach that is simultaneously capable of generating a set of instances which are diverse with respect to different search spaces (instance or performance) *and* that exhibit discriminatory but diverse performance with respect to a portfolio of solvers (where diversity in the latter case refers to variation in the magnitude of the performance gap). Unlike some previous work where the goal is to discover instances that are specifically *hard* for a portfolio of solvers, our goal is to develop a space-covering method that produces a large set of discriminatory instances that maximises coverage of the instance space and that generalises across both portfolios and domains.

Our approach utilises Novelty Search (NS), proposed by Lehman and Stanley (2011), a type of Evolutionary Algorithm (EA) that in its basic form rewards novelty rather than quality, thus driving a search algorithm to explore large parts of the search space. We evaluate the approach in two domains: the Knapsack Problem (KP) and the Bin Packing (BP) problem. We consider novelty defined with respect to (1) a feature-based descriptor and (2) a performance-based descriptor to evaluate how the search for new instances can best be guided. To ensure the approach generalises across portfolios of solvers, we consider three different portfolios for the KP domain: a portfolio of metaheuristics, a portfolio based on deterministic heuristics, and a portfolio containing three state-of-the-art exact solvers. Results show that both NS approaches succeed not only in providing considerably better coverage of the instance and performance spaces, but also obtaining larger sets of diverse and discriminatory instances in comparison to an approach that does not make use of novelty. Furthermore, in contrast to previous methods, such as that proposed by Alissa et al. (2019), a single run of our method returns a large set of instances that are diverse and discriminatory with respect to a single target-solver. It therefore needs to be run M times, where M is the number of solvers in the portfolio. In contrast, space-filling approaches, such as those described by Smith-Miles

et al. (2010); Alissa et al. (2019), tend to converge to a single solution, so in the worst case need to be run $i \times M$ times to generate i instances that are discriminatory for each of the M solvers.

The paper extends a recent conference paper (Marrero et al., 2022) in which the idea of using NS in the KP domain was first proposed. In this preliminary work, NS was used in conjunction with a feature-based descriptor only, and an empirical investigation into the appropriate tuning of parameters was conducted. This article makes the following new contributions:

- It extends previous work by introducing a new measure of novelty, specifically a performance-based descriptor that describes the performance of each solver in a portfolio, avoiding the need to calculate features. A detailed comparison of both feature and performance-based descriptors is provided.
- It provides a detailed comparison to another Quality Diversity (QD) algorithm, MAP-Elites (Mouret and Clune, 2015), which was recently shown to be able to generate novel instances in the TSP domain (Bossek and Neumann, 2022). In addition, we also compare to an evolutionary algorithm that only evolves for discriminatory ability between solvers, rather than novelty. The superiority of the NS approach is demonstrated in both cases.
- It provides evidence that the method generalises across *domains* through experiments that generate instances for both the KP and BP domains.
- It provides evidence that the method generalises across *portfolios of solvers* by means of experiments that generate instances for meta-heuristic, deterministic heuristics and state-of-the-art solver portfolios.

The remaining of this paper is organised as follows. Section 2 describes previous research carried out in the field of instance generation in multiple domains. We then present a detailed description of our proposed method in Section 3. In Section 4, the approach is evaluated in two domains, KP and BP. We use the KP domain as a first case-study to evaluate the approach in detail, comparing to existing methods of instance-

generation and also to known instance spaces. Having demonstrated its efficacy, we repeat the evaluation of the key experiments in the BP domain. Finally, conclusions and lines of further research are presented in Section 5.

2 Related Work

Several authors have previously applied evolutionary methods to target generation of a set of instances where one solver outperforms others in a portfolio. For example, working in the TSP domain, Smith-Miles et al. (2010) use an EA to evolve new instances which are easy (or hard) for a given solver using an objective function that minimises (or maximises) the search-effort required by a solver to produce a tour. Plata-González et al. (2019) focus on the KP and use an EA to evolve instances for a target algorithm using an objective function that maximises the gap between the performance of the target and the other solvers in the portfolio. Alissa et al. (2019) follow the same approach but working in the BP domain. These approaches require running an EA multiple times to generate instances for a single target algorithm, with each run producing one or more instances, depending on the extent to which the final population has converged. The process must be repeated per target algorithm. Furthermore, these methods cannot guarantee providing a diverse set of instances as each run might converge to similar solutions. As an alternative to using an EA, Dang et al. (2022) propose a constraint-based framework used in conjunction with a racing technique (López-Ibáñez et al., 2016) which provides an automated approach to generate instances that are graded (solvable at a certain difficulty level for a solver) or can discriminate between two solving approaches. This is shown to be successful in generating instances for five problems taken from the MiniZinc domain (Nethercote et al., 2007) but like the EA approaches mentioned above, focuses only on generating discriminatory rather than 'diverse *and* discriminatory' instances.

To address this, other work attempts to explicitly maintain diversity while evolving instances that are easy (or hard) for a target algorithm. In Gao et al. (2016), the selection method of the EA is altered to favour offspring that maintain diversity with respect to a chosen feature, as long as the offspring have a performance gap over a

given threshold, providing promising results for TSP. In Bossek et al. (2019), also working in TSP, novel mutation operators are designed which are able to generate diverse instances with respect to a set of features using a simple algorithm. The operators are also incorporated into an EA to optimise the generation of easy/hard instances for a target algorithm. Their results show that their method is capable of exposing a large difference in algorithm performance (easy for one solver and hard for its contender) while covering a broad spectrum of instance characteristics.

Rather than focusing on evolving discriminatory instances, Smith-Miles and Bowly (2015); Smith-Miles et al. (2021) describe a method for evolving new instances to directly fill gaps in an *instance space* that has been constructed using a large set of known instances. An existing set of instances is first projected onto a 2D plane in which the projection is optimised to reveal pockets of strengths and weaknesses of multiple algorithms in the projected feature-space. An EA is then used to evolve new instances to target gaps in this instance space: overlaying the instance space with performance data helps to reveal where hard instances are for a given portfolio. In Smith-Miles et al. (2021), the gap-filling approach is used to reveal new insights into where the hard problems lie regarding the KP domain, for which the instances proposed by Pisinger (2005) are considered as the initial set. Two approaches are considered: in the first, an EA is used to generate weakly structured instances by evolving an instance definition where fitness is defined as the distance of the instance to the target region. In the second, an EA tunes the parameters of an instance generator to again minimise the gap to the target region. Their analysis reveals new insights into the locations of hard KP problems. This method has also been applied in other domains, e.g. TSP, Graph Colouring (Smith-Miles and Bowly, 2015), and black-box continuous optimisation (Muñoz and Smith-Miles, 2020). Focusing on try to gain new insights into what makes an instance hard for KP solvers, Jooken et al. (2022) proposed a stochastic generator to produce a new class of hard instances for the KP. Although most of the new instances generated via this method are located in the same region of the space as those generated by Smith-Miles et al. (2021), the new instances are much harder to solve than previously known instances, despite being smaller.

Researchers have also appealed to the QD algorithm literature (Pugh et al., 2016) to find better methods for generating diverse but discriminatory instances. This family of algorithms that includes NS (Lehman and Stanley, 2011) and MAP-Elites (Mouret and Clune, 2015) provides mechanisms for simultaneously forcing exploration of a search-space while optimising for quality within each region. The robotics and games literature[1] has witnessed a rapid explosion of application and development of QD methods in recent years which is recently beginning to filter into the combinatorial optimisation domain, specifically as a tool to deliver diverse solutions to an end-user (Urquhart and Hart, 2018). However, it was recently shown that MAP-Elites can be used to evolve instances for TSP that are diverse with respect to two chosen features *and* discriminatory with respect to two solvers (Bossek and Neumann, 2022). At the same time, preliminary work in the KP domain (Marrero et al., 2022) demonstrated that NS can evolve large sets of instances that are discriminatory for a portfolio of four solvers in a single domain. This article pushes further in developing a generalised method of producing diverse instances that broadly cover an instance space, and are discriminatory with respect to a portfolio of solvers chosen by a user. It extends existing work by comparing two different novelty descriptors, as the definition of an appropriate descriptor is key to the functioning of NS (Doncieux et al., 2019). Furthermore, we demonstrate that it generalises over multiple solver portfolios (containing state-of-the-art, deterministic and stochastic solvers), and across two domains (KP and BP), as well as providing comparisons to existing state-of-the art instance-generation methods, including MAP-Elites. Finally, we show that the approach is able to generate new instances that lie in new regions of the instance space not covered by well-known benchmarks, e.g. Smith-Miles et al. (2021).

3 Methods

This section provides a detailed description of the general method. It is then rigorously evaluated, first in the context of the KP domain using multiple portfolios, and afterwards in the BP domain to demonstrate its ability to generalise.

[1] See `https://quality-diversity.github.io`.

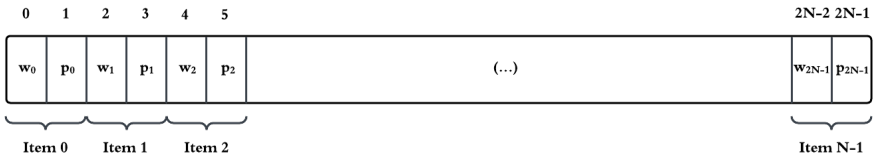


Figure 1: Representation of the instances as stored in memory (genotype).

3.1 Domain and instance representation

We first apply the approach to generating instances for the KP, a commonly studied combinatorial optimisation problem with many practical applications. The KP requires the selection of a subset of items from a larger set of N items, each with profit p and weight w in such a way that the total profit is maximised while respecting a constraint that the weight remains under the knapsack capacity C.

A knapsack instance is described by an array of integers of size $2 \times N$ where N is the dimension (number of items) of the instance of the KP we want to create (see Figure 1), with the weights ($w_{i=0...N-1}$) and profits ($p_{i=0...N-1}$) of the items stored at the even and odd positions of the array, respectively. The capacity C of the knapsack is determined for each new individual generated as 80% of the total sum of weights, as using a fixed capacity would tend to create trivial solutions if the weights of the instances increase significantly. We evolve fixed size instances containing $N = 50$ items, hence each individual describing an instance contains 100 values describing pairs of (profit, weight). In both cases, upper and lower bounds were set to delimit the maximum and minimum values of both profits and weights.

3.2 Novelty descriptors

We compare two NS approaches, in which novelty of an instance is defined either with respect to the features of the instance (referred to from here on as feature-based–NS_f) or to the performance of the portfolio on that instance (referred to as performance-based–NS_p).

In the feature-based version, the novelty descriptor is a vector representing features of an instance being evolved. A set of eight features are used to describe a KP

instance. The chosen features are inspired by those used by Plata-González et al. (2019), thus containing: capacity of the knapsack; minimum weight and profit; maximum weight and profit; average item efficiency (average sum of the ratios profit/weight of each item); mean distribution of values considering profits and weights ($N \times 2$ integer values representing the instance); standard deviation of values considering profits and weights.

The performance-based descriptor is defined as an M-dimensional vector with the average performance of each optimiser considered in a portfolio of size M. Given a portfolio of M algorithms, then the novelty descriptor for an instance is calculated as an M-dimensional vector, where each element $A_{i=0...M-1}$ represents the average performance over R repetitions of algorithm $A_{i=0...M-1}$ on the instance. Searching for novel performance descriptors ensures that the portfolio is discriminatory (as in order to be novel, the entries in the descriptor should differ). However, it should be noted that using this descriptor, it is not possible to guarantee that the differences are significant, or that the search process will find instances that are 'won' by each of the algorithms; i.e. in the worst case, it would be possible to find novel descriptors in which a single algorithm is always the 'winner'.

An additional motivation behind using a performance-based descriptor is that it avoids the definition of a problem-dependent feature-based descriptor. Thus, the method could be applied to other domains where obtaining a feature-based descriptor might be a challenging task.

3.3 Algorithm portfolios

Since in principle, the portfolio can contain any number of types of algorithms that can produce a solution to the problem, we consider, as a first portfolio, a set of differently configured versions of a meta-heuristic approach, specifically an EA. Parameter tuning can significantly impact EA performance on an instance (Nannen et al., 2008). Therefore, it is expected that different configurations of the same approach cover different regions of the instance space. The EA used to produce a solution to each KP instance (EA_{solver}) is a generational elitist algorithm with parameters defined in Ta-

ble 3 in the Appendix. Previous empirical investigation of this algorithm revealed that the crossover rate was one of the parameters with the largest impact over its performance (Marrero et al., 2022). Therefore, we defined four configurations that differ only in the crossover rate. The crossover rate can take one of four possible values commonly used in the literature: 0.7, 0.8, 0.9 or 1.0.

Two additional portfolios are also considered in order to determine if the approach generalises. The first is a portfolio containing a set of four simple deterministic heuristics (Plata-González et al., 2019) commonly used in the field, i.e. Default (Def), which selects the first item available to be inserted into the knapsack; Max Profit (MaP), which sorts the items by profit and selects those items with largest profit first; Max Profit per Weight (MPW), which sorts the items by its efficiency (ratio between the profit and weight of each item) and selects those items with largest ratio first; and Min Weight (MiW), which selects items with the lowest weight first.

Lastly, instances are evolved that are discriminative for a portfolio of state-of-the-art solvers for the KP domain. The portfolio consists of a branch-and-bound technique called *Expknap* (Pisinger, 1995), and two dynamic programming approaches known as *Minknap* (Pisinger, 2000) and *Combo* (Martello et al., 1999). Since a detailed explanation of these methods is out of the scope of this work, the reader is referred to the original publications for more details.

3.4 Instance generation via novelty search

Novelty Search was first introduced by Lehman and Stanley (2011) as an attempt to mitigate the problem of finding the optimal solution in deceptive problems, i.e. a problem in which lower-order building blocks, when combined, do not lead to a global optimum (Lehman and Stanley, 2011). In such problems, the reward signal or fitness function may actively steer search away from exactly the region of the space where the solution lies. The core idea was to replace the objective function in a standard evolutionary search process with a function that rewards behavioural novelty rather than a performance-based fitness: counter-intuitively, they showed that can lead to the discovery of better solutions than using an objective-based search.

Algorithm 1: Novelty Search

```
   Input: N, k, evals, portfolio
1  initialise(population, N);
2  evaluate(population, portfolio);
3  archive = ∅ ;
4  for i = 0 to evals do
5  |  parents = select(population);
6  |  offspring = reproduce(parents);
7  |  offspring = evaluate(offspring, portfolio, archive, k) (see Algorithm 2);
8  |  population = update(population, offspring);
9  |  archive = update_archive(population, archive);
10 |  solution_set = update_ss(population, solution_set);
11 end
12 return solution_set
```

We propose to use NS to discover a diverse set of instances that can be used e.g. to inform algorithm-selection. While the 'vanilla' version of NS would deliver diversity, it does not have the desired property of creating instances that are also tailored to be discriminative for a given algorithm. However, a number of techniques have been proposed in the NS literature to combine the explorative aspect of NS with the more exploitative search performed by objective-based search algorithms. Multiple proposals for achieving the desired balance between the two factors can be found in the literature, e.g. Cuccu and Gomez (2011) propose using a linear scalarisation of novelty and fitness scores, allowing the experimenter to control the relative weight of the novelty and fitness scores, while a Pareto-based multi-objective optimisation has also been proposed by Mouret (2011). A thorough comparison of these methods finds little difference between them (Gomes et al., 2015). Therefore, we adopt the linear scalarisation method from Cuccu and Gomez (2011), defining novelty with respect to either a feature-space or performance-space and objective fitness as the performance difference between a target algorithm and the others in the portfolio considered, herein referred to as the performance-gap.

The NS algorithm ($EA_{instance}$) used is described by Algorithm 1. The algorithm evolves an initial population of randomly generated instances: one execution results in generation of a diverse set of instances that are tailored to a chosen target algorithm belonging to the portfolio. An initial empirical investigation was conducted to set the population size and the number of evaluations parameters to obtain a reasonable trade-

Algorithm 2: Evaluation method

```
Input: offspring, portfolio, archive, k
for instance in offspring do
    for algorithm in portfolio do
        apply algorithm to solve instance R times;
        calculate mean profit of algorithm
    end
    calculate the novelty score(offspring, archive, k) (Equation 1);
    calculate the performance score(offspring) (Equation 2);
    calculate fitness(offspring) (Equation 4);
end
return offspring
```

off between the quality of the results obtained and the amount of time for attaining them. The remaining parameters were set by considering values commonly used in the literature (Nannen et al., 2008). All parameters are given in Table 4 in the Appendix.

3.4.1 Computing the fitness of an instance

To calculate the fitness of an *instance* in the population (see Algorithm 2), two quantities are required: (1) the novelty score measuring the *sparseness* of the instance and (2) the performance score measuring the difference in average performance over R repetitions between the target algorithm and the best of the remaining algorithms.

Given a descriptor x, i.e., typically a multi-dimensional vector capturing relevant information from a solution, the most common approach is to quantify novelty of an individual via a *sparseness* metric which measures the average distance between the individual's descriptor and its *k*-nearest neighbours. The *k* nearest-neighbours are determined by comparing a descriptor to the descriptors of all other members of the current population and to those stored in an external *archive* of past individuals. The sparseness s of an instance (Lehman and Stanley, 2011) is then defined as shown in Equation 1:

$$s(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i) \tag{1}$$

where μ_i is the *i*th-nearest neighbour of x with respect to a user-defined distance metric *dist*.

The performance score ps of an instance is calculated differently depending on the

particular portfolio of solvers for which instances are generated. First, for portfolios based on EA_{solver} or deterministic heuristics, ps is computed as the difference between the mean profit achieved in R repetitions[2] of the target algorithm denoted as t_p and the maximum of the mean profits achieved in R repetitions by the remaining approaches of the portfolio defined as o_p (see Equation 2). Profit is defined as the sum of the profits of items included in the knapsack.

$$ps = t_p - max(o_p) \tag{2}$$

On the other hand, portfolios that include state-of-the-art solvers generally reach the optimal solution for the instance at hand, hence profit cannot be used to distinguish algorithm performance. Instead, performance is measured in terms of the mean CPU time required to achieve the optimal solution in R repetitions. Thus, each solver is run for a maximum of 15 seconds and their run times are collected. If any solver fails to obtain the optimal solution in that time, we set its run time to 15 seconds. This time limit is defined following the proposal of Smith-Miles et al. (2021). Moreover, to ensure that the instances are won by the target algorithm (i.e. it reaches the optimal solution faster than any other solver in the portfolio), the mean run time achieved in R repetitions by the target solver (t_r) must be smaller than the minimum mean run time obtained by the other solvers in the portfolio achieved in R repetitions o_r (see Equation 3).

$$ps = min(o_r) - t_r \tag{3}$$

Finally, the fitness f (Equation 4) used to drive the evolutionary process is calculated as a linear weighted combination of the novelty score s and the performance score ps of an instance, where ϕ is the performance/novelty balance weighting factor, following the method described by Cuccu and Gomez (2011).

[2]In the case of a portfolio based on deterministic heuristics parameter R is set to 1, i.e. no repetitions are required due to the deterministic nature of the heuristics.

$$f = \phi * ps + (1 - \phi) * s \tag{4}$$

It is important to note that to generate discriminatory instances for different algorithms, the target algorithm must vary from one execution to another, i.e. the approach has to be run for each target solver in the portfolio. This is the same for the majority of methods proposed previously which also have to be run per solver (Alissa et al., 2019; Plata-González et al., 2019; Smith-Miles et al., 2010). However, unlike our approach, previously proposed approaches cannot ensure that multiple instances are produced in a single run, since the objective function only tries to maximise the performance gap between the target solver and the next best: as a result, the population can converge to a single instance in the worst case. In contrast, one of the advantages of our proposal based on NS lies in the fact that a single run ensures the generation of multiple instances. Hence, on average, our method is likely to produce larger sets of diverse and discriminatory instances in comparison to those generated by previous methods for the same computational effort.

Parameters associated specifically with the NS algorithm were set according to the results presented in previous work (Marrero et al., 2022) in which we conducted a rigorous exploration of parameters (see Table 4 in the Appendix).

3.4.2 Archive and solution set management

As it can be observed in Algorithm 1, the NS approach has to manage an *archive* and a *solution set*. The archive is supplemented at each generation in two ways. Firstly, a sample of individuals from the current population is randomly added to the archive with a probability of 1% following common practice in the literature (Szerlip et al., 2014). Secondly, any individual from the current generation with sparseness greater than a pre-defined threshold t_a is also added to the archive.

In addition to the archive described above, which is used to calculate the sparseness metric that drives evolution, a separate list of individuals, denoted as the solution set, is incrementally built as the algorithm runs: this constitutes the final set of instances returned when the algorithm terminates (Szerlip et al., 2014). The solution set

is initialised by including the fittest individual of the population at the end of the first generation of the algorithm. Subsequently, at the end of each generation, each member of the current population is scored against the solution set by finding the distance to the nearest neighbour ($k = 1$) in the solution set. Those individuals that score above a particular threshold t_{ss} are added to the *solution set*. The solution set forms the output of the algorithm.

We should note that the solution set does not influence the sparseness metric driving the evolutionary process. Instead, this approach ensures that each solution returned has a descriptor that differs by at least the given threshold t_{ss} from the others in the final collection. Finally, both the archive and the solution set grow randomly on each generation depending on the diversity discovered without any limit on their final size.

4 Experiments and Results

The primary goal of this work is to evaluate the extent to which NS can be used to generate diverse but discriminatory instances to cover an instance space, and to demonstrate it generalises across domains and solver portfolios. Hence, we:

- Conduct experiments with two different novelty descriptors (feature-based and performance-based) and provide a quantitative and qualitative evaluation of the diversity and discriminative ability of generated instances in two domains: KP and BP.
- Analyse the diversity of each subset of instances 'won' by a target algorithm with respect to the performance-gap, i.e. the magnitude of the difference between the performance of the winning solver and the next-best solver in the portfolio.
- Conduct experiments to compare the performance of the novelty-based approaches against another QD method (MAP-Elites), and against an EA that only attempts to maximise discriminatory ability, following the approaches used by, e.g. Plata-González et al. (2019); Alissa et al. (2019).
- Demonstrate that the method generalises across different solver portfolios: metaheuristics, deterministic heuristics and state-of-the-art approaches.

All algorithms were written in C++, compiled using GNU's GCC 10.0.2 compiler, and included into DIGNEA (Marrero et al., 2023b).[3]

4.1 Generation of instances for a portfolio of EA_{solver} for KP

First, we run $EA_{instance}$ with NS_f and NS_p to compare both approaches using a portfolio of meta-heuristic solvers: a set of EAs in which each has a different EA_{solver} configuration, described in Table 3. The result is a dataset of instances generated from each novelty method (NS_f and NS_p) to favour specific configurations that are diverse with respect to the descriptors detailed in Section 3.2, i.e., a feature-space descriptor for NS_f and a performance space descriptor for the NS_p approach.

Since we are interested in evaluating the distribution of the instances over the respective spaces, we create 2D representations of the instances in both the feature and performance spaces. The procedure to generate these plots is described as follows. For each instance in the dataset, we compute the alternate descriptor; i.e. for the instances generated using NS_f we compute the corresponding performance-descriptor and vice-versa. Therefore, after this step, each instance in the dataset contains the information from both descriptors. To plot instances, we reduce the dimensionality of the dataset using the Principal Component Analysis (PCA) technique. The data used to create the PCA projections contains the descriptor of interest and additional information describing the whole instance, i.e. the capacity C, profits $p_{i=0...N-1}$ and weights $w_{i=0...N-1}$, is also considered for visualisation purposes. As we are evaluating the distribution in two different spaces, we apply PCA in each space separately but are able to project all instances into each space.

Figure 2 provides the representation of the instances in both spaces. On the left-hand side, the instances are distributed over the feature space. Notice how the instances generated by the NS_f approach (red dots) cover the space quite uniformly, while on the contrary, the instances generated by NS_p (green crosses) are clustered in the centre of the region. This is not surprising since NS_f is designed to find diversity in the feature

[3]Supplementary material is available through a Github repository: `https://github.com/PAL-ULL/ns_comb_journal`. The source code of DIGNEA can be found in the following Github repository: `https://github.com/DIGNEA/dignea`.

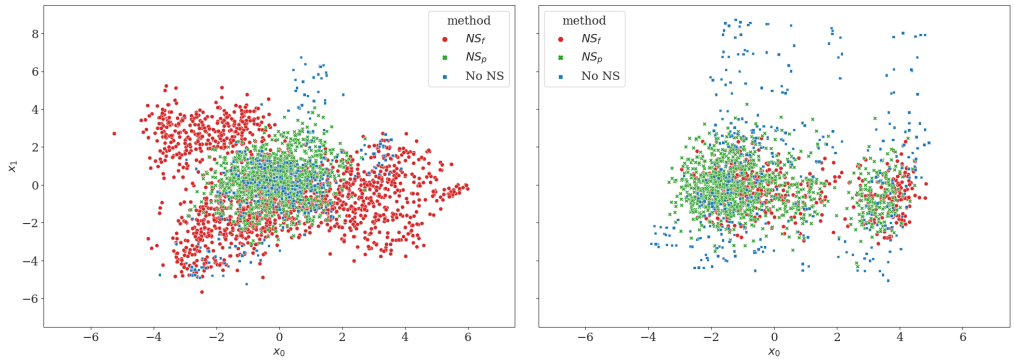


Figure 2: **Left**: Instance representation in the feature space after applying PCA to a dataset containing the feature descriptors and instance information of all instances generated by both NS approaches (NS_f and NS_p), as well as by $EA_{instance}$ without NS. **Right**: Instance representation in the performance space after applying PCA to a dataset containing the performance descriptors and instance information of all instances generated by both NS approaches (NS_f and NS_p), as well as by $EA_{instance}$ without NS. Red dots are used for those instances produced by NS_f, green crosses represent instances generated through NS_p, and blue squares represent instances generated by $EA_{instance}$ without NS. In both cases, the portfolio based on EA_{solver} was considered.

Table 1: Space coverage using the U metric over feature and performance spaces for instances generated with a portfolio of EA_{solver}.

Method	**Feature space**	**Performance space**
NS_f	**0.7880**	0.6805
NS_p	0.7112	**0.7073**
$EA_{instance}$ without NS	0.5995	0.5881

space. However, when we plot the instances over the performance space (right-hand side of Figure 2), there is no obvious difference between NS_p and NS_f in terms of space coverage. Even though NS_p is designed to search for diversity in the performance space, both approaches end up with a similar spatial distribution of instances. Regarding the use of $EA_{instance}$ without NS, instances are scattered in the centre and upper regions of the spaces. Since diversity is not promoted during the evolutionary process that only rewards objective fitness, the instances are located in the regions where the performance gap between the target algorithm and the remaining approaches in the portfolio is maximised.

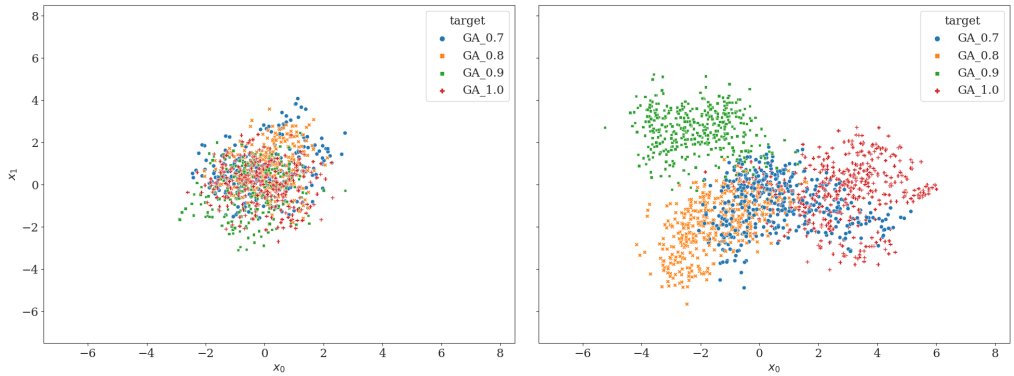
In order to quantitatively evaluate the extent to which the evolved instances cover the instance space, we calculate the exploration uniformity (U) metric, previously pro-

posed by Gomes et al. (2015). This enables a comparison of the distribution of solutions in the space with a hypothetical Uniform Distribution (UD). First, the environment is divided into a grid of 25×25 cells, after which the number of solutions in each cell is counted. Next, the Jensen-Shannon divergence (JSD) (Fuglede and Topsoe, 2004) is used to compare the distance of the distribution of solutions with the ideal UD. The U metric is then calculated according to Equation 5, where δ denotes a $2D$*-descriptor* associated with a solution. This descriptor is defined as the two principal components of each solution extracted after applying PCA to the descriptors described in Section 3.2. A score of 1 denotes a perfectly uniform distribution. Thus, the higher the score the better.

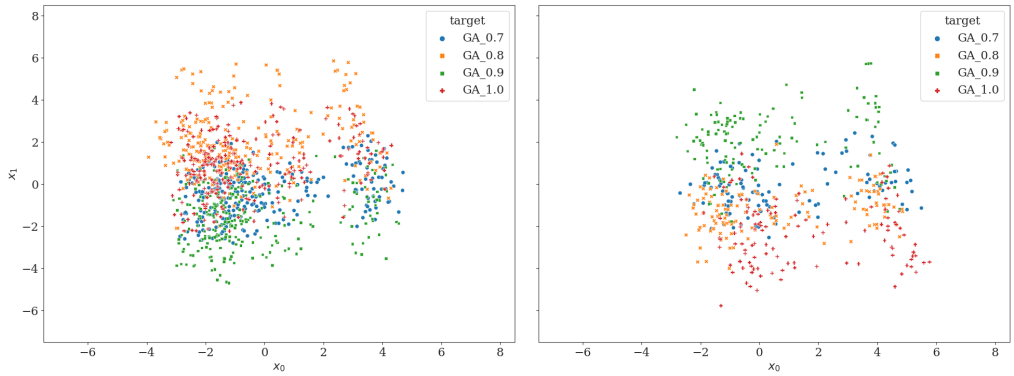
$$U(\delta) = 1 - JSD(P_\delta, UD) \tag{5}$$

Table 1 summarises the coverage metric U per each generation method over the feature and performance spaces. For each space, three different approaches are evaluated; i.e. NS_f, NS_p and $EA_{instance}$ without NS. In terms of coverage of the feature space, NS_f obtains higher quality results compared to the other approaches. This is according to our expectations since NS_f is designed to find diversity in the feature space and therefore it performs a better exploration of the said space. Similarly, NS_p wins in terms of coverage of the performance space, as expected, by following the previous reasoning. Nevertheless, we should note that NS_f obtains good results compared to NS_p in the performance space, in contrast to the feature space where differences between NS_f and NS_p are larger. Finally, running $EA_{instance}$ without NS results in poor (less than 0.6) coverage for both feature and performance spaces. In the light of these results, we conclude that using NS has a substantial impact in improving space coverage and also that NS_f seems to be a preferable choice for obtaining better coverage across spaces.

At this point, it is also relevant to determine the regions of the spaces where the instances for specific solvers are located. Figures 3a and 3b provide an insight into the instance location per target solver in both spaces. Figure 3a shows the instances gener-



(a) Instances generated per solver in the portfolio based on EA_{solver} by NS_p (left) and NS_f (right) considering the feature space.

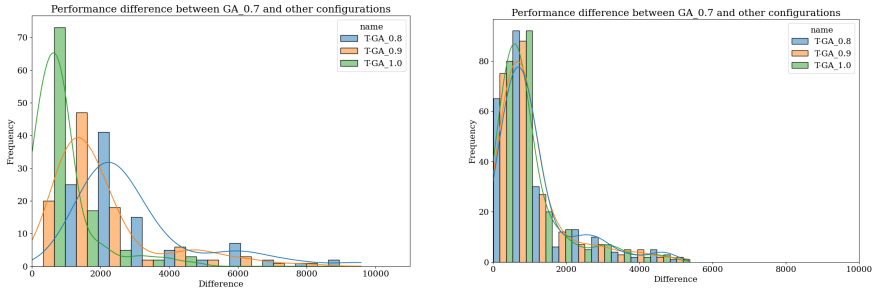


(b) Instances generated per solver in the portfolio based on EA_{solver} by NS_p (left) and NS_f (right) considering the performance space.

Figure 3: Instances generated per solver in the portfolio based on EA_{solver} by both NS_f and NS_p methods projected in the feature and performance spaces. Coloured symbols reflect the 'winning' solver in the said portfolio for a particular instance: red pluses (crossover rate set to 1.0); green squares (0.9), orange crosses (0.8) and blue dots (0.7).

ated using NS_p (left-hand side) and NS_f (right-hand side) with their respective targets over the feature space. Analogously, Figure 3b shows the instances over the performance space. It is noteworthy that in both cases, NS_f generates more differentiated clusters of instances which could be easily classified by an external method.

In the light of these results, we indicate that NS_f is preferable if the goal is to find diversity in the feature space. Conversely, both approaches seem to present similar results when looking for diversity in the performance space. Moreover, from Figures 3a and 3b we could also infer that NS_f produces a better distribution of instances with respect to the associated target solvers.



(a) NS_f (b) NS_p

Figure 4: Distribution of the performance gap between the target approach GA_0.7 (T in the key) and the remaining approaches in the portfolio based on EA_{solver} by considering the instances generated by methods NS_f 4a) and NS_p (4b).

$EA_{instance}$ aims to evolve a diverse set of instances whose performance is tailored to favour a specific target algorithm. Recall however that the performance values obtained are stochastic due to the nature of the solvers used in the portfolio (EA_{solver}). Therefore, considering instances evolved for each of the target algorithms with setting $\phi = 0.85$, we conduct a rigorous statistical evaluation to determine whether the results obtained on the set of instances evolved for a target algorithm show statistically significant differences compared to applying each of the other algorithms in the portfolio to the same set of instances.

We followed a statistical testing procedure, whose detailed description can be found in the Appendix, to support that the instances are in fact discriminatory. Additional results are provided in Table 5 in the Appendix, showing that this is true except in a very small minority of cases where a draw arose. We note that in no case did the target algorithm perform statistically worse on an instance generated for it when compared to another solver in the portfolio.

In the case of NS_f, for the three algorithms with configuration $\{0.7, 0.8, 0.9\}$, then for the vast majority of instances, the target algorithm outperforms the other algorithms. However, for these three algorithms, it appears harder to find diverse instances where the respective algorithm outperforms the algorithm with configuration 1.0. Thus the results provide insights into the relative strengths and weaknesses of each algorithm (approximated by the number of generated instances).

On the contrary, for NS_p, the results in Table 5 show that even though the target algorithm is able to statistically outperform other approaches in some instances, the number of instances where there is no statistically significant difference is larger in every scenario when compared to the results achieved by NS_f. For example, approach GA_1.0 was able to show statistically better performance than GA_0.8 in 101 out of 209 instances, but there was no significant difference in 108 out of those 209 instances. Note however that the number of instances generated by NS_p was larger in every case in comparison to the number of instances generated by NS_f.

Figures 2, 3a and 3b provided an insight into the diversity of the evolved instances with respect to the feature and performance spaces. We now provide further insight into the diversity of the evolved instances in terms of performance difference (performance gap) among solvers using both NS approaches (see Figure 4). That is, we consider instances that are 'won' by a target algorithm and consider the spread in the magnitude of the performance gap as defined in Equation 2. We note that the approach is able to generate diverse instances in terms of this metric using either NS approximation: while a significant number of instances have a relatively small gap when using NS_p (as seen, for instance, by the left skew to the distribution in Figure 4b), we also find instances spread across the range (see Figure 4a) when NS_f is applied. The instances therefore exhibit diversity in terms of the performance gap, as well as diversity in terms of coverage of the analysed spaces.

4.2 Generation of instances for a portfolio of deterministic tailored heuristics for KP

To understand how the method generalises across different portfolios, we perform a similar experimental evaluation first with a portfolio deterministic KP heuristics, i.e. Def, MaP, MPW and MiW, which were defined in Section 3.3. All the parameters for $EA_{instance}$ are the same as shown in Table 4 in the Appendix, setting the maximum evaluations to perform at 10,000 and parameter $R = 1$. We run $EA_{instance}$ with the new portfolio using both variants, i.e. NS_f and NS_p, as in the previous section.

Figure 5 provides an insight into the distribution of the instances over the feature

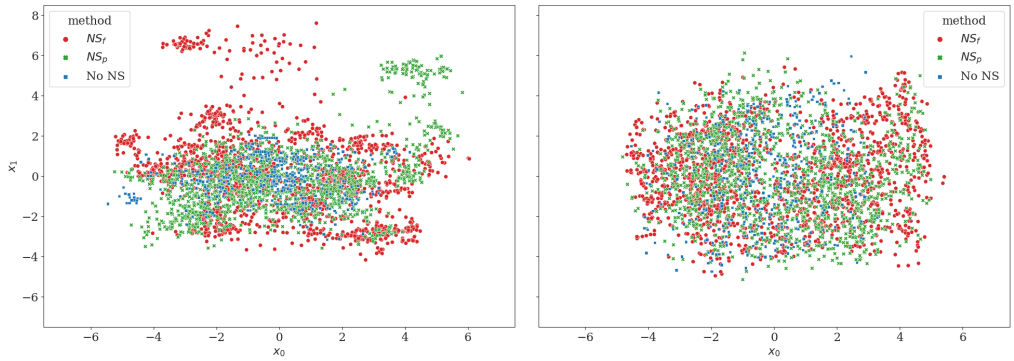


Figure 5: **Left**: Instance representation in the feature space after applying PCA to a dataset containing the feature descriptors and instance information of all instances generated by both NS approaches (NS_f and NS_p), as well as by $EA_{instance}$ without NS. **Right**: Instance representation in the performance space after applying PCA to a dataset containing the performance descriptors and instance information of all instances generated by both NS approaches (NS_f and NS_p), as well as by $EA_{instance}$ without NS. Red dots are used for those instances produced by NS_f, green crosses represent instances generated through NS_p, and blue squares represent instances generated by $EA_{instance}$ without NS. In both cases, the portfolio based on deterministic heuristics was considered.

and performance spaces with respect to the NS approach used to generated them. Red dots represent the instances generated by NS_f and green crosses the instances generated by NS_p. In contrast to the results obtained with the portfolio of EA_{solver}, the instances share a similar distribution across both spaces, with some NS_f instances occupying larger regions in the performance space. Either way, there is no significant difference in this aspect. Even though, for this portfolio, $EA_{instance}$ without NS covers approximately the same regions as NS_f and NS_p, notice how the instances are gathered more in the centre of both spaces.

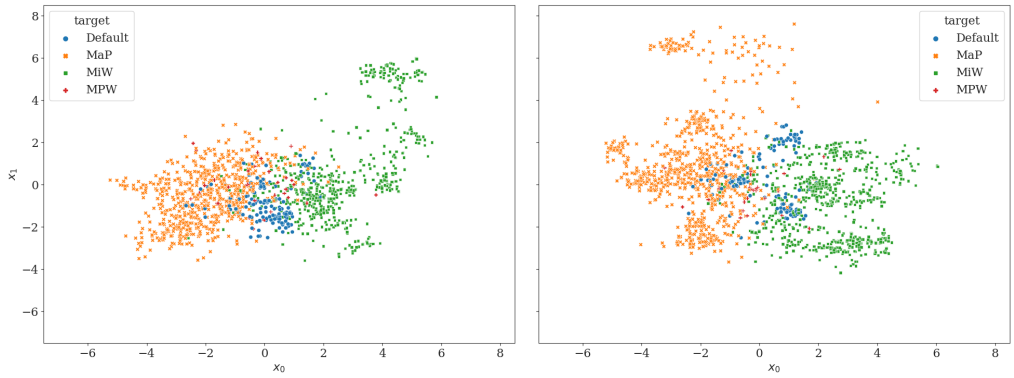
In terms of the coverage metric U, Table 2 provides an insight per each generation method over the feature and performance spaces. Results show analogous behaviour to using a portfolio of EA_{solver}. That is, NS_f and NS_p obtain better coverage than other approaches over their respective spaces. Although $EA_{instance}$ without using NS seems to get higher coverage values with this new portfolio, notice that the difference between this approach and the 'winner' approach remains similar to those results shown in Table 1 for both spaces.

Table 2: Space coverage using the U metric over the feature and performance spaces for instances generated with a portfolio of heuristics.

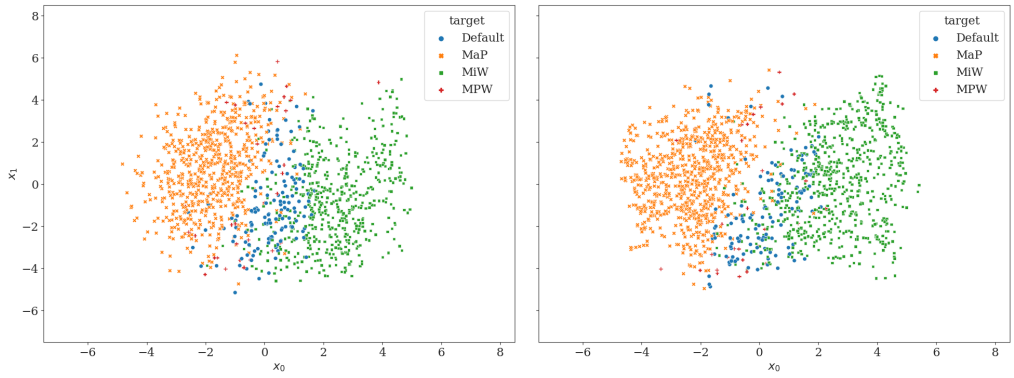
Method	**Feature space**	**Performance space**
NS_f	**0.7863**	0.7297
NS_p	0.7340	**0.8233**
$EA_{instance}$ without NS	0.6233	0.7003

In addition, it seems that the approach is able to generate a better distribution of the instances with respect to the target solver in this new portfolio for both spaces and NS approaches (see Figures 6a and 6b). It is worth discussing the difference between these figures and the previous ones. Take, for instance, Figure 3a and its analogous Figure 6a. While the instances are stacked over each other when using a portfolio of EA_{solver} configurations and NS_p, considering a portfolio of heuristics results in a more widely spread distribution of instances and better grouping by target solvers. Moreover, comparing both portfolios over the performance space (Figures 3b and 6b) reveals that regions of the instances 'won' by the algorithms in the portfolio of heuristics are better differentiated compared to the portfolio of EA_{solver}. Besides, as in the previous experiment, NS_f is able to generate good distributions in both spaces. NS_p fails to generate good space distributions when the solvers in a portfolio behave similarly, in that, there is little difference between their performances (as in the EA_{solver} portfolio). In this case, it is difficult to find novel descriptors as all solvers return similar values.

At this point, it is interesting to see how the distribution of the values of the eight features selected for the KP domain are related to the different four deterministic heuristics selected for this particular portfolio. To do so, we plotted each feature of the 8D descriptor per instance generated by NS_f (see Figure 15 in the Appendix). It can be observed how MaP and MiW tend to have bigger ranges for a significant number of the features in comparison to MPW and Default. This is not unexpected since they are the two solvers for which the largest number of instances is produced. In addition to the above, we also calculated a pairwise correlation matrix using the Pearson correlation. We note that all but one pair has a coefficient < 0.29 which is deemed a weak correlation. The pair consisting of the features 'average item efficiency' and 'min-



(a) Instances generated per solver in the portfolio based on deterministic heuristics by NS_p (left) and NS_f (right) considering the feature space.

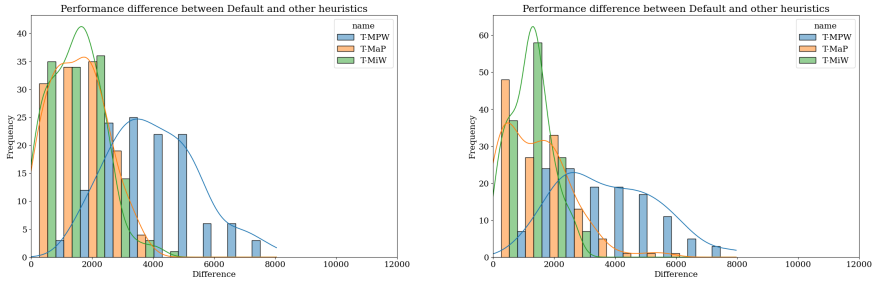


(b) Instances generated per solver in the portfolio based on deterministic heuristics by NS_p (left) and NS_f (right) considering the performance space.

Figure 6: Instances generated per solver in the portfolio based on deterministic heuristics by both NS_f and NS_p methods projected in the feature and performance spaces. Coloured symbols reflect the 'winning' solver in the said portfolio for a particular instance: red pluses (MPW); green squares (MiW), orange crosses (MaP) and blue dots (Default).

imum weight' has a coefficient of 0.48, for which we show the relation regarding the particular target solver in Figure 16 in the Appendix.

We now follow the same statistical procedure as described previously to evaluate the new portfolio of heuristics. Considering the deterministic nature of the new set of algorithms, the comparison was carried out using the profit achieved by each approach after solving each instance generated. The results can be found in Table 6 in the Appendix. Moreover, it is important to highlight that the results show that the instances generated are completely biased to the target solver. In contrast to the results from the



(a) NS_f (b) NS_p

Figure 7: istribution of the performance gap between the target approach Def (T in the key) and the remaining approaches in the portfolio based on deterministic heuristics by considering the instances generated by methods NS_f 7a) and NS_p (7b).

portfolio based on EA_{solver} (see Table 5 in the Appendix), where the performance of the algorithms on some instances show no significant differences, the results from this experiment demonstrate that, in each case, the target algorithm presents statistically significant better performance than its competitors in all instances. As a consequence, we demonstrate that our method is able to evolve completely tailored yet diverse instances when the algorithms in the portfolio are substantially different in terms of their design.

Finally, we evaluate the diversity of the evolved instances with respect to the performance gap among solvers in the portfolio (see Figure 7). Again, results show that the approach is able to generate diverse instances in terms of Equation 2. Nevertheless, we should note that the performance gap among algorithms in the heuristic-based portfolio is considerably higher for some cases with respect to the performance gap detected in the previous experiment. For instance, in Figure 7, the performance gap between Def and MPW is considerably larger in comparison to the previous results attained by the different configurations of EA_{solver} (Figure 4).

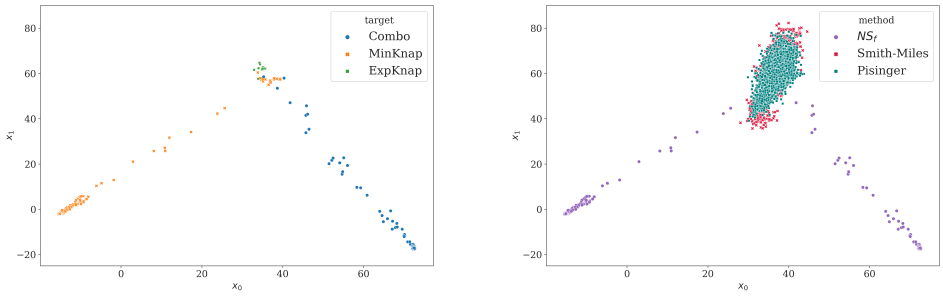
4.3 Generation of instances for a portfolio of state-of-the-art solvers for KP

The naïve heuristics in the previous portfolio are not expected to provide high-performing results when dealing with large or real-world instances. Therefore, in order to provide more evidence that our method is able to generalise across portfolios, we perform another experimental evaluation with a portfolio of state-of-the-art solvers, i.e.

Expknap, *Minknap* and *Combo* and attempt to evolve much larger instances to demonstrate that the method scales. We set the parameters according to those used in recent work by Smith-Miles et al. (2021), which focused on filling gaps in an instance space to understand where hard knapsack problems exist. Note however that our goal is different in that we aim to generate a space-covering set of discriminative instances. Thus, the number of items is set to $N = 1000$, $R = 10$, and the stopping criterion is switched to CPU time instead of maximum evaluations to perform. For this state-of-the-art portfolio, ps is calculated using Equation 3. The expected scores from these solvers can be extremely low (just a few milliseconds for Minknap and Combo), hence to avoid discarding large sets of solutions, the archive and solution set thresholds have been updated to $t_a = t_{ss} = 1e - 7$. The remaining parameters are the same as shown in Table 4 in the Appendix.

We run $EA_{instance}$ 10 times per solver in the portfolio using only NS_f. In order to evaluate the results in the context of existing work in the KP domain, we situate our results in an instance space that includes the large set of instances discussed in Smith-Miles et al. (2021), which are either collected from the literature or generated to fill gaps in the space, enabling new insights into where hard problems lie. We reiterate that the goal of our approach is to cover as much space as possible in a single run: this contrasts with approaches that target specific gaps that would have to be run repeatedly with different targets to cover the whole space. It is constructive, however, to analyse the resulting space containing our newly generated instances and those from Smith-Miles et al. (2021).

Thus, we create a dataset combining the instances produced by NS_f, Pisinger's original instances (Pisinger, 2005) gathered by Smith-Miles, and the new instances evolved by Smith-Miles et al. (2021) to fill gaps in the existing benchmarks. To provide a fair comparison, we calculate the 8D feature descriptor defined in Section 3.2 for each instance in the dataset and reduce the dataset by means of a PCA model fed with the feature descriptors and instance information. Figure 8 provides an insight of the distribution of the instances over the feature space. On the left-hand side (Figure 8a), NS_f instances are represented for each target solver, i.e. blue dots represent



(a) Instances per solver

(b) Instances per method

Figure 8: **Left (8a)**: Instances generated by NS_f for each target approach in the portfolio based on state-of-the-art solvers. Blue circles are used to represent those instances generated for Combo, orange crosses represent instances produced for Minknap, and green squares represent instances generated for Expknap. **Right (8b)**: Instances produced by each generation method: NS_f (purple dots) and the method proposed by Smith-Miles et al. (2021) (crimson crosses), which starts from the original instances provided by Pisinger (teal squares). To obtain each of both figures, PCA is applied to a dataset containing the feature-based descriptors, as well as the information (weights and profits), of the corresponding instances.

the instances generated for Combo, orange crosses are used for instances produced for Minknap, and green squares the instances generated for Expknap. There is a notable disparity in the quantity of instances generated for each solver, with Expknap exhibiting a relatively lower number of instances. While Combo/Minknap find the optimal solution in a few milliseconds, Expknap may take more than 15 seconds to reach the optimal for $N = 1000$ items. Consequently, the task of generating instances for Expknap becomes notably challenging when it is part of the same solver portfolio as Combo and Minknap. Moreover, the right-hand side (Figure 8b) provides a visualisation of the instances produced by each generation method, i.e. NS_f, instances evolved by Smith-Miles, and instances from the original Pisinger's generation methods. As expected, the original Pisinger instances and the instances generated by Smith-Miles et al. (2021) occupy the same region of the space. This is obvious given that the goal of Smith-Miles *et. al.* was to fill in gaps in an instance space containing the Pisinger dataset. In contrast, NS_f (purple dots) is able to find diverse and discriminatory instances for the portfolio using state-of-the-art solvers in distant regions of the space not covered by the instances described in Smith-Miles et al. (2021). With the exception of Expknap instances (green

squares in Figure 8a), which overlap with the existing benchmarks, the vast majority of the instances are located in two corners of the space. Specifically they highlight new regions in which either Combo or Minknap perform well.

4.4 Comparison against alternative generation methods: MAP-Elites

We now turn our attention to evaluating how our NS method compares against another QD method, i.e. MAP-Elites (Mouret and Clune, 2015), which also returns a set of solutions and was recently demonstrated to be an effective tool for instance generation in the TSP domain (Bossek and Neumann, 2022). We use the MAP-Elites version described by Mouret and Clune (2015) to search for diverse and high-performing solutions in the 8D feature space of the KP domain. Pseudo-code and parameter setting is provided in Algorithm 3 and Table 7, respectively, in the Appendix.[4] MAP-Elites requires the user to define a multi-dimensional archive in which each axis represents a feature and is discretised into cells. In this case, we use an 8D grid where each axis corresponds to one of the eight features already defined. The boundaries of each axis must therefore be defined (unlike in NS). We calculate these boundaries using the instances of size $N = 50$ generated by NS_f for the portfolio of deterministic heuristics (Section 4.2). For each of the eight features considered, we calculate $b_i = (l_i, u_i)$. As the manner in which the grid is discretised is a parameter of the method, we evaluate different resolutions: $r \in \{3, 5, 10, 15, 20, 25\}$. An in-depth description of MAP-Elites can be found in Mouret and Clune (2015).

We run MAP-Elites at each resolution to generate diverse and discriminatory instances for a portfolio of deterministic KP heuristics. In order to provide a fair comparison, we ensure that the computational budget of MAP-Elites is identical to NS_f; i.e. both methods perform the same total number of evaluations. Then, PCA is applied to a dataset containing the 8D feature descriptor and instance information of all instances generated by MAP-Elites and NS_f. Figure 9 shows the distribution of instances generated by NS_f (results from Section 4.2) and MAP-Elites with $r = 15$. While most of the instances generated by MAP-Elites are located in the same region as NS_f, the method

[4]The source code of MAP-Elites can also be found in the aforementioned GitHub repository.

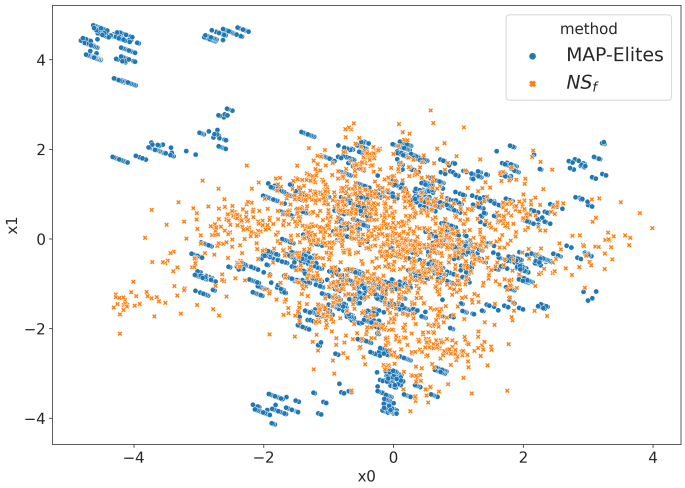


Figure 9: Instances generated by NS_f and $15r$-MAP-Elites considering the portfolio based on deterministic heuristics. Orange crosses are used to represent those instances generated by NS_f, while those instances produced by $15r$-MAP-Elites are represented through blue dots.

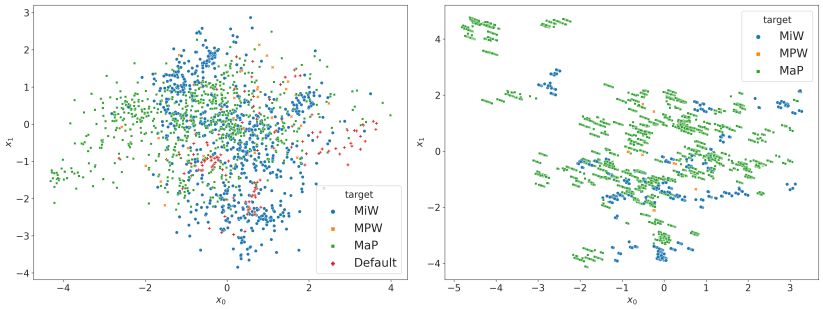


Figure 10: Instances generated by NS_f (left) and $15r$-MAP-Elites (right) for each solver in the portfolio based on deterministic heuristics. In both figures, blue points are used to represent instances generated for MiW, orange crosses considering MPW as the target, green squares for instances produced for MaP and red pluses for Default.

is able to find a number of instances in other regions such as the top left corner of the space.

Figure 10 shows the distribution of instances per solver in the portfolio. The left-hand side presents the instances generated by NS_f, while the right-hand side shows instances generated by MAP-Elites with $r = 15$. Note that even though we only present the results for $r = 15$, MAP-Elites was not able to generate instances for the 'Default'

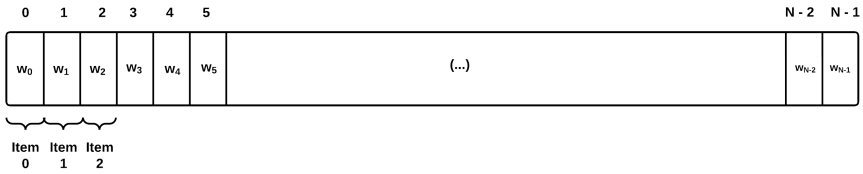


Figure 11: Representation of the BP instances as stored in memory (genotype).

heuristic in any run. The remaining plots with different values for parameter r can be found in the Appendix. The space coverage in terms of the U metric is $U(NS_f) = 0.6943$ and $U(15r\text{-MAP-Elites}) = 0.6443$. These scores indicate that NS_f is able to provide better coverage of the feature space than this MAP-Elites configuration. In fact, NS_f scores higher than any MAP-Elites configurations evaluated (U scores for each r-MAP-Elites configuration can be found in Table 8 in the Appendix).

4.5 Generation of instances for the BP domain

In order to demonstrate that the NS_f method can be generalised to other optimisation domains, we now apply it to the BP domain (Coleman and Wang, 2013). The goal of BP is to find a packing that minimises the number of bins B of fixed capacity C required to fit a set of N items of weights w_i, while enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity C.

A BP instance is described by an array of integer numbers of size N where N is the dimension (number of items) of the instance of the BP we want to create (see Figure 11), with the weights of each item stored. The instances are described following the Falkenauer U class (Falkenauer, 1996) setting $N = 120$, the capacity of each bin $C = 150$, and the weights distributed in the range $[20, 100]$. As in KP, the novelty descriptor for the BP domain is a vector representing features of an instance being evolved. A set of 10 features is used to describe a BP instance, following Alissa et al. (2019): mean of the weights of the items; median of the weights of the items; standard deviation of the weights, maximum and minimum weights; proportion of items i that have a weight $w_i > \frac{1}{2}$ denoted as huge; proportion of items that have a weight $\frac{1}{3} < w_i \leq \frac{1}{2}$ denoted

as large; proportion of items that have a weight $\frac{1}{4} < w_i \leq \frac{1}{3}$ denoted as medium; proportion of items that have a weight $w_i \leq \frac{1}{4}$ denoted as small, and proportion of items that have a weight $w_i \leq \frac{1}{10}$ denoted as tiny. All weights of the items are normalised, so $C = 1$.

We use a portfolio of four simple deterministic solvers for the BP domain, following previous work in algorithm-selection for BP (Alissa et al., 2019, 2023): First Fit (FF), which places each item into the first feasible bin that will accommodate it; Best Fit (BF), that allocates each item into the feasible bin that minimises the residual space; Worst Fit (WF), where each item is inserted into the feasible bin with the most available space; and Next Fit (NF), which inserts each item into the current bin (Garey and Johnson, 1981). Performance is calculated according to Falkenauer's performance metric (Falkenauer, 1996) (see Equation 6). In this study, we fixed $z = 2$, fill_j is the sum of the weights of the items inside bin_j, and B is the total number of bins used.

$$Fa = \frac{\sum_{j=1}^{B} (\frac{\text{fill}_j}{C})^z}{B} \tag{6}$$

Therefore, the performance score ps of a solver in the BP domain is calculated following Equation 7.

$$ps = t_{Fa} - max(o_{Fa}) \tag{7}$$

$EA_{instance}$ is run 10 times for each solver in the portfolio using only NS_f. We compare the results with the instances generated by Alissa et al. (2019) using the same portfolio. Alissa *et. al.* created a set containing 1,000 discriminatory instances best solved by one of each heuristic in the portfolio (4,000 instances in total). The parameter configuration of $EA_{instance}$ can be found in Table 9 in the Appendix. We combine all the instances into a single dataset and calculate the 10D feature descriptor for the BP domain previously detailed. As before, the resulting dataset is reduced by means of a PCA model fed with the feature descriptors and instance information. Figure 12 provides an insight into the distribution of the instances over the BP feature space. It can be observed that the Alissa *et. al.* instances are clustered on top of each other on

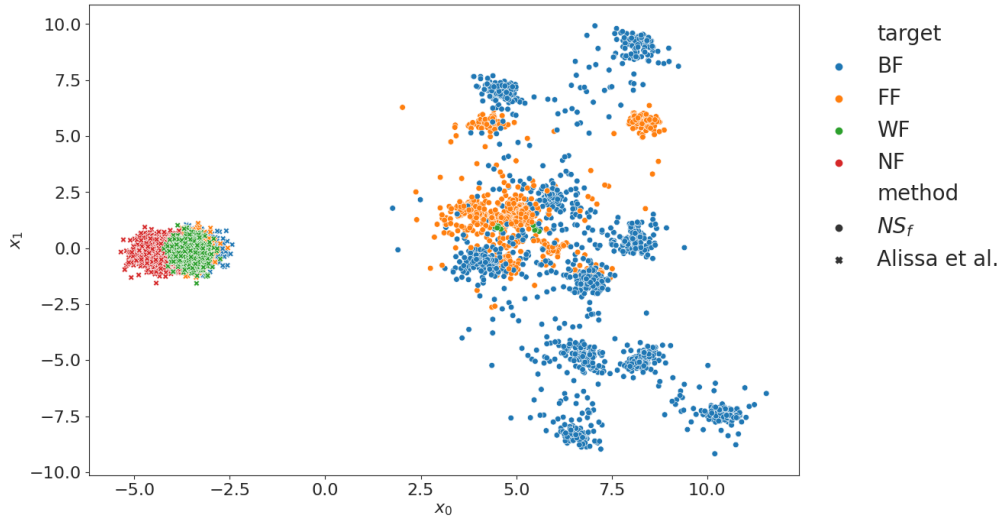


Figure 12: Instance distribution by 'wining' solver across each generation method. Dots represent the instances generated by NS_f while crosses represent the instances generated by Alissa et al. (2019). Colours are then used to differentiate instances generated for each target solver in the portfolio.

the left-hand side of the space, while the instances generated by NS_f have a wider spread on the right-hand side of the space. The distribution of instances per solver is somewhat imbalanced: NS_f generated 2,342 instances from which 1,476 instances are discriminatory for BF, 855 for FF, and 11 for WF. The approach failed to generate instances for NF.[5] Calculating the space coverage in terms of the U metric over the feature space results in the following scores: $NS_f = 0.5583$ and Alissa et al. $= 0.3536$. These findings align with the data distribution illustrated in Figure 12, and confirm the capability of NS_f in creating BP instances that offer improved coverage of the feature space. Finally, an analysis of the feature distribution of the newly generated instances using NS_f revealed that all instances have a value 0 for the medium and tiny features. Inspecting the features of the instances from Alissa *et. al.*, we note that the huge, small and tiny features all have value 0. The NS_f approach thus appears to lead to more variability in terms of feature descriptor for BP (see Figure 22 in the Appendix).

[5]Personal communication with the authors of Alissa et al. (2019) indicated that significant computational effort was necessary using their method to find instances in which NF outperformed the other solvers.

5 Conclusions and Further Research

The main objective of this work was to propose an instance-generation algorithm based on NS that can be used to generate diverse instances for combinatorial optimisation domains, where the instances are also discriminatory for a given portfolio of solvers. We proposed two variants of an NS algorithm that uses a weighted combination of performance and diversity to drive selection. In the first variant, the novelty descriptor that drives the search is defined with respect to the features of an instance (NS_f); in the second, the descriptor is defined with respect to a vector denoting the performance of each solver in a portfolio (NS_p).

We evaluated these methods using two domains (KP, BP) to demonstrate the generality of the approach across domains, and also using different portfolios of solvers (meta-heuristics, deterministic heuristics, state-of-the-art exact methods) to demonstrate robustness against the choice of portfolio. We show that the method is capable of generating a large set of diverse, discriminatory instances in one run, in contrast to many previously proposed methods (Alissa et al., 2019; Plata-González et al., 2019; Smith-Miles et al., 2010) and also results in better coverage of the feature-space than another QD algorithm, MAP-Elites, that was used to generate instances for TSP in previous work (Bossek and Neumann, 2022). Furthermore, in contrast to previous approaches that tend to maximise the performance gap between a target solver and the next best solver in a portfolio, our method generates discriminatory instances with respect to a target solver that has broad variation in terms of the magnitude performance gap. We believe this provides better data with which to train more robust algorithm selection methods. Finally, in order to demonstrate that the method scales, in the KP domain, instances were generated with 50 items, following Marrero et al. (2022), and 1000 items, as per recent work in the literature, e.g. Smith-Miles et al. (2021), where the goal was to discover hard KP instances.

In the KP domain, results demonstrate that both the NS_f and NS_p approaches generate discriminatory instances for a range of portfolios, providing better coverage than either the EA approach proposed by e.g. Plata-González et al. (2019) and the MAP-

Elites algorithm, adapted by Bossek and Neumann (2022). Recent work has shown that an *ensemble* consisting of a number of NS_f approaches is beneficial in terms of space coverage and uniform distribution of the instances over the KP features space (Marrero et al., 2023a). Given that MAP-Elites locates some instances that are different from those generated by NS_f , including MAP-Elites in an ensemble of instance-generating methods could beneficial in future. Visualising the coverage obtained by both variants NS_f and NS_p (see Sections 4.1 and 4.2), it is clear that NS_f locates instances in some part of the space that NS_p does not, and vice versa: this suggests that an *ensemble* approach to instance-generation that combines instances generated using both novelty descriptors would be beneficial going forward. NS_f is evaluated across three different portfolios. Results show that it is robust with respect to the type of solver in the portfolio, generating diverse instances for all solvers contained in three different portfolios: meta-heuristics, deterministic heuristics and exact solvers. Note however that the number of generated instances won by each solver varies. For example, for the exact solver portfolio, it is most difficult to generate instances that are won by *Expknap*, while for the deterministic portfolio, it is hardest to generate instances won by the *Default* solver.

The results above also generalise to the BP domain, where the NS_f variant is used to generate new instances for a portfolio of deterministic heuristics. A total number of 2,342 new instances were generated, covering more of the feature-space than the instances generated by Alissa et al. (2019). However, we note that in this case, no instances were generated for the NF heuristic. Personal communication with the authors of Alissa et al. (2019) revealed that significant computational effort was devoted to finding instances that were won by this heuristic using their method. Further inspection of the features of the generated instances also showed that the 'medium' and 'tiny' features have value zero for each instance, providing some insight into the distributions that lead to discriminatory performance.

Finally, although the objective of this study is to provide broad coverage of a space in any domain, rather than specifically generate hard instances for a domain of interest, it is instructive to examine our results in the context of existing results in the KP domain where there has been significant recent efforts to understand where the hard instances

generated for a portfolio of state-of-the-art solvers are. Although an exact comparison to recent work such as Smith-Miles et al. (2021) is not possible as their method attempted to fill gaps in an instance space that already contained a very large number of instances, we used NS_f to try and generate new instances that were discriminatory for the same portfolio of solvers used in Smith-Miles et al. (2021). This additional study also demonstrates that the method scales, given the instance size is now 1,000 rather than 50 as used in the remaining KP experiments. Given that the 1,300 instances gathered by Smith-Miles et al. (2021) already occupy a large part of the instance-space, it would be expected to be challenging to find new instances.Nonetheless, NS_f finds 87,492 new instances with the vast majority of them occupying a different region of the space than the existing dataset. These instances are mainly discriminatory with respect to the Combo and Minknap heuristics. NS_f struggles to find instances in which Expknap is the winner. Hence our results can be used to augment existing datasets and provide new insights in to the feature distributions of instances that Combo/Minknap perform well on.

Although outside the scope of this study, an interesting angle for future work would be to adapt the NS methods proposed in this article to generate *hard* and diverse instances for a domain (defined with respect to a chosen solver). This could easily be achieved by adapting the performance-related element of the fitness function to maximise the time taken for an exact method such as Minknap to find a solution, rather than optimising for discriminatory ability within a portfolio. This could potentially bring new insights into where the hard instances lie in a domain. It is important to note that the time taken to generate diverse and hard instances depends on the type of solvers in the portfolio: using exact methods might significantly increase the running time.

Multi-objective methods that trade-off the performance/diversity metrics would also be worth exploring. Finally, another fruitful angle could be to focus on the features used to define the novelty descriptor and therefore the search-space. There is much existing work in the field of feature-generation that could be drawn on, for example in constructing new features as linear (Tran et al., 2019; Marrero et al., 2023a) or

non-linear combinations of existing features to discover new feature-spaces in which to search. Additionally, several works exist that propose different features for the KP domain (Smith-Miles et al., 2021; Jooken et al., 2023). However, we note that some of these features require a significant amount of time to compute and could therefore impact running time. Nevertheless, future work could investigate the use of different novelty descriptors that make use of some or all of these features.

With respect to both domains considered, this could also include adapting the representation of the instance that is used to define the search-space to consider additional information, for example the capacity constraint in KP or the bin-size in BP.

Acknowledgements

This work was partially funded by the Spanish "Ministerio de Ciencia, Innovación y Universidades" as well as by the "Universidad de La Laguna (ULL)", as part of the project with contract number 2022/0000580.

The work of **Alejandro Marrero** was funded by "Agencia Canaria de Investigación, Innovación y Sociedad de la Información de la Consejería de Universidades, Ciencia e Innovación y Cultura y por el Fondo Social Europeo Plus (FSE+) Programa Operativo Integrado de Canarias 2021-2027, Eje 3 Tema Prioritario 74 (85%)" with grant TESIS2020010005.

The work has been performed under Project HPC-EUROPA3 (INFRAIA-2016-1-730897), with the support of the EC Research Innovation Action under the H2020 Programme; in particular, the authors gratefully acknowledge the computer resources and technical support provided by Edinburgh Parallel Computing Centre (EPCC).

This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk) as well as the TeideHPC infrastructure from ITER (https://www.iter.es/).

Prof. Emma Hart is supported by the EPSRC grant Keep Learning EP/V026534/1.

References

Alissa, M., Sim, K., and Hart, E. (2019). Algorithm Selection Using Deep Learning Without Feature Extraction. *In Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Re- public. ACM, New York, NY, USA*.

Alissa, M., Sim, K., and Hart, E. (2023). Automated algorithm selection: from feature-based to feature-free approaches. *Journal of Heuristics*, 29(1):1–38.

Bossek, J., Kerschke, P., Neumann, A., Wagner, M., Neumann, F., and Trautmann, H. (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*, pages 58–71.

Bossek, J. and Neumann, F. (2022). Exploring the Feature Space of TSP Instances Using Quality Diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 186–194, New York, NY, USA. Association for Computing Machinery.

Coleman, N. and Wang, P. (2013). *Bin-Packing*, pages 116–126. Springer US, Boston, MA.

Cuccu, G. and Gomez, F. (2011). When novelty is not enough. In *European Conference on the Applications of Evolutionary Computation*, pages 234–243. Springer.

Dang, N., Akgün, Ö., Espasa, J., Miguel, I., and Nightingale, P. (2022). A framework for generating informative benchmark instances. *arXiv preprint arXiv:2205.14753*.

Doncieux, S., Laflaquière, A., and Coninx, A. (2019). Novelty search: A theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 99–106, New York, NY, USA. Association for Computing Machinery.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30. ID: Falkenauer1996.

Fuglede, B. and Topsoe, F. (2004). Jensen-Shannon divergence and Hilbert space embedding. In *International Symposium on Information Theory, 2004. ISIT 2004. Proceedings.*, pages 31–.

Gao, W., Nallaperuma, S., and Neumann, F. (2016). Feature-based diversity optimization for problem instance classification. In *International Conference on Parallel Problem Solving from Nature*, pages 869–879. Springer.

Garey, M. R. and Johnson, D. S. (1981). *Approximation Algorithms for Bin Packing Problems: A Survey*, pages 147–172. Springer Vienna, Vienna.

Gomes, J., Mariano, P., and Christensen, A. L. (2015). Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 943–950, New York, NY, USA. Association for Computing Machinery.

Hains, D., Whitley, D., and Howe, A. (2012). Improving Lin-Kernighan-Helsgaun with crossover on clustered instances of the TSP. In *International Conference on Parallel Problem Solving from Nature*, pages 388–397. Springer.

Jooken, J., Leyman, P., and De Causmaecker, P. (2022). A new class of hard problem instances for the 0-1 knapsack problem. *European Journal of Operational Research*, 301(3):841–854.

Jooken, J., Leyman, P., and De Causmaecker, P. (2023). Features for the 0-1 knapsack problem based on inclusionwise maximal solutions. *European Journal of Operational Research*, 311(1):36–55.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–222.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Marrero, A., Segredo, E., Hart, E., Bossek, J., and Neumann, A. (2023a). Generating diverse and discriminatory knapsack instances by searching for novelty in variable dimensions of feature-space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 312–320.

Marrero, A., Segredo, E., León, C., and Hart, E. (2022). A novelty-search approach to filling an instance-space with diverse and discriminatory instances for the knapsack problem. In *Parallel Problem Solving from Nature – PPSN XVII*, pages 223–236, Cham. Springer International Publishing.

Marrero, A., Segredo, E., León, C., and Hart, E. (2023b). DIGNEA: A tool to generate diverse and discriminatory instance suites for optimisation domains. *SoftwareX*, 22:101355.

Martello, S., Pisinger, D., and Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424.

Mouret, J.-B. (2011). Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*, pages 139–154. Springer.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

Muñoz, M. A. and Smith-Miles, K. (2020). Generating new space-filling test instances for continuous black-box optimization. *Evolutionary computation*, 28(3):379–404.

Nannen, V., Smit, S. K., and Eiben, A. (2008). Costs and Benefits of Tuning Parameters of Evolutionary Algorithms. *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer.

Pisinger, D. (1995). An expanding-core algorithm for the exact 0–1 knapsack problem.

Pisinger, D. (2000). A minimal algorithm for the bounded knapsack problem. *INFORMS journal on computing*, 12(1):75–82.

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers and Operations Research*, 32(9):2271–2284.

Plata-González, L. F., Amaya, I., Ortiz-Bayliss, J. C., Conant-Pablos, S. E., Terashima-Marín, H., and Coello Coello, C. A. (2019). Evolutionary-based tailoring of synthetic instances for the Knapsack problem. *Soft Computing*, 23(23):12711–12728.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, page 40.

Rice, J. R. (1976). The Algorithm Selection Problem. *Advances in Computers*, 15(C):65–118.

Smith-Miles, K. and Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers and Operations Research*, 63:102–113.

Smith-Miles, K., Christiansen, J., and Muñoz, M. A. (2021). Revisiting where are the hard knapsack problems? via instance space analysis. *Computers & Operations Research*, 128:105184.

Smith-Miles, K., van Hemert, J., and Lim, X. Y. (2010). Understanding TSP difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280. Springer.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):1–25.

Szerlip, P. A., Morse, G., Pugh, J. K., and Stanley, K. O. (2014). Unsupervised Feature Learning through Divergent Discriminative Feature Accumulation. *AAAI'15: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Tran, B., Xue, B., and Zhang, M. (2019). Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition*, 93:404–417.

Urquhart, N. and Hart, E. (2018). Optimisation and illumination of a real-world workforce scheduling and routing application (wsrp) via map-elites. In *International Conference on Parallel Problem Solving from Nature*, pages 488–499. Springer.

A Computational Resources

The total time taken to run the whole experimental assessment was 1,120 hours approximately, by using diverse computational resources including High-Performance Computing (HPC) infrastructures:

- HPE Cray EX (5,860 nodes) with two AMD EPYC 7742 64-core 2.25GHz processors and 256 GB (standard), 512 GB (high memory) RAM.
- Fujitsu Primergy CX250 (1,052 nodes) with two Intel Xeon ES-2670 CPUs with 8 cores (16 slots) and 32/64 GB RAM each node.
- Node with two AMD EPYC 7502 64-core at 2.5 GHz processors with 128 GB RAM.

B Parameter Setting of EA_{solver} for the KP domain

Table 3: Parameter settings for EA_{solver}. The crossover rate is the distinguishing feature for each configuration.

Parameter	Value
Population size	32
Max. Evaluations	1e5
Mutation rate	1 / N
Crossover rate	0.7, 0.8, 0.9, 1.0
Crossover	Uniform
Mutation	Uniform
Selection	Binary Tournament

C Parameter Setting of $EA_{instance}$ for the KP domain

Table 4: Parameter settings for $EA_{instance}$ which evolves the diverse population of discriminatory instances.

Parameter	Value
Knapsack items (N)	50, 1000
Weight and profit upper bound	1,000
Weight and profit lower bound	1
Population size	10
Crossover rate	0.8
Mutation rate	1 / ($N \times 2$)
Stop criteria	CPU time (state-of-the-art) Evaluations (others)
Evaluations	2,500, 5,000, 10,000, 15,000
Repetitions (R)	1 (deterministic heuristics) 10 (EA_{solver}, state-of-the-art)
Distance metric	Euclidean Distance
Neighbourhood size (k)	3
Thresholds (t_a, t_{ss})	3.0, 1e-7

D Statistical Analysis

We followed a statistical testing procedure to support the conclusions. First, a *Shapiro-Wilk test*[6] was performed to check whether the values of the results followed a normal (Gaussian) distribution. If so, the *Levene test*[7] checked for the homogeneity of the variances. If the samples had equal variances, an ANOVA *test*[8] was done; if not, a *Welch test*[9] was performed. For non-Gaussian distributions, the non-parametric *Kruskal-Wallis*[10] test was used. For every test, a significance level $\alpha = 0.05$ was considered. The comparison was carried out considering the mean profits achieved by each approach at the end of 10 runs for each instance generated.

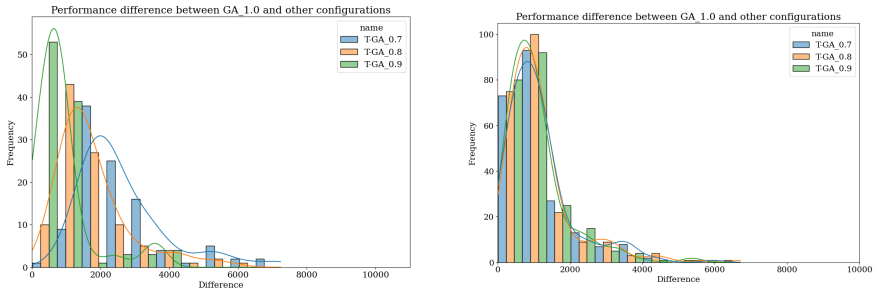
An example of the above procedure applied to a set of results is shown in Table 5. For each target approach A in the first column, the number of 'wins' ($\uparrow$) and 'draws' ($\leftrightarrow$) of the said target algorithm with respect to other approach B is shown. A 'win' means that approach A was able to present statistically better performance in comparison to approach B, by following the aforementioned statistical comparison procedure, when solving a particular instance. A 'draw' means that both approaches A and B did not present statistically significant differences in performance when solving a particular instance. For instance, when running $EA_{instance}$ using NS_f, approach GA_0.7 was able to provide statistically better performance than approach GA_0.8 in 87 out of 90 instances, which in fact were generated by considering GA_0.7 as the target approach. Both GA_0.7 and GA_0.8 did not present statistically significant differences in performance when solving 3 out of 90 instances generated. Finally, note that in no case did the target algorithm lose on an instance to another algorithm.

[6]Shapiro-Wilk test in Python: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html`
[7]Levene test in Python: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html`
[8]ANOVA test in Python: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html`
[9]Welch test in Python: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html`
[10]Kruskal-Wallis test in Python: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html`

E Generation of Instances for a Portfolio of EA_{solver} for KP

Table 5: Wins and draws between the target configuration and other configurations after applying the statistical analysis. A win (↑) indicates statistically significant difference between two configurations and that the mean performance value of the target approach was higher. A draw (↔) indicates no statistically significant difference between both configurations. The number of instances generated for each target approach was 90 for approach GA_0.7, 101 for GA_0.8, 110 for GA_0.9 and 80 instances for GA_1.0 when using NS_f. On the other hand, running $EA_{instance}$ with NS_p produced 196 instances for GA_0.7, 224 for GA_0.8, 221 for GA_0.9 and 209 for GA_1.0.

NS Approach		GA_0.7	GA_0.8	GA_0.9	GA_1.0
NS_f	**GA_0.7**		↑ 87 ↔ 3	↑ 69 ↔ 21	↑ 25 ↔ 65
	GA_0.8	↑ 100 ↔ 1		↑ 77 ↔ 24	↑ 21 ↔ 80
	GA_0.9	↑ 107 ↔ 3	↑ 87 ↔ 23		↑ 18 ↔ 92
	GA_1.0	↑ 21 ↔ 59	↑ 61 ↔ 19	↑ 76 ↔ 4	
NS_p	**GA_0.7**		↑ 73 ↔ 123	↑ 74 ↔ 125	↑ 58 ↔ 141
	GA_0.8	↑ 95 ↔ 129		↑ 82 ↔ 142	↑ 85 ↔ 139
	GA_0.9	↑ 93 ↔ 128	↑ 85 ↔ 136		↑ 82 ↔ 139
	GA_1.0	↑ 99 ↔ 110	↑ 101 ↔ 108	↑ 92 ↔ 117	
No NS	**GA_0.7**		↑ 44 ↔ 26	↑ 48 ↔ 22	↑ 41 ↔ 29
	GA_0.8	↑ 50 ↔ 20		↑ 52 ↔ 18	↑ 42 ↔ 28
	GA_0.9	↑ 53 ↔ 17	↑ 57 ↔ 13		↑ 59 ↔ 11
	GA_1.0	↑ 64 ↔ 6	↑ 65 ↔ 5	↑ 60 ↔ 10	



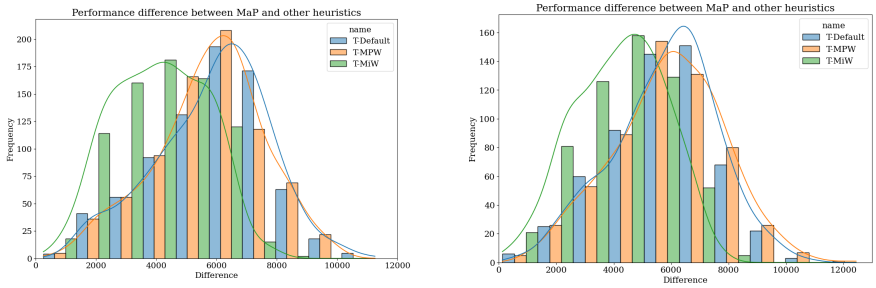
(a) NS_f (b) NS_p

Figure 13: Distribution of performance gap between the approach GA_1.0 (T in the key) and other approaches by considering the instances generated for the former when running $EA_{instance}$ using NS_f (13a) and NS_p (13b).

F Generation of Instances for a Portfolio of Deterministic Tailored Heuristics for KP

Table 6: Wins and draws between the target configuration and other configurations after applying the statistical analysis. A win (↑) indicates statistically significant difference between two configurations and that the mean performance value of the target was higher. A draw (↔) indicates no statistically significant difference between both configurations. The number of instances generated for each target approach was 123 for approach Def, 774 for MaP, 22 for MPW and 687 instances for MiW when using NS_f. On the other hand, running $EA_{instance}$ with NS_p produced 129 instances for approach Def, 572 for MaP, 22 for MPW and 488 for MiW. Moreover, $EA_{instance}$ without NS obtains a significant lower amount of instances, 54 instances for Def, 100 for MaP, 40 for MPW and 92 for MiW.

NS Approach		Def	MaP	MPW	MiW
NS_f	**Def**		$\uparrow 123 \leftrightarrow 0$	$\uparrow 123 \leftrightarrow 0$	$\uparrow 123 \leftrightarrow 0$
	MaP	$\uparrow 774 \leftrightarrow 0$		$\uparrow 774 \leftrightarrow 0$	$\uparrow 774 \leftrightarrow 0$
	MPW	$\uparrow 22 \leftrightarrow 0$	$\uparrow 22 \leftrightarrow 0$		$\uparrow 22 \leftrightarrow 0$
	MiW	$\uparrow 687 \leftrightarrow 0$	$\uparrow 687 \leftrightarrow 0$	$\uparrow 687 \leftrightarrow 0$	
NS_p	**Def**		$\uparrow 129 \leftrightarrow 0$	$\uparrow 129 \leftrightarrow 0$	$\uparrow 129 \leftrightarrow 0$
	MaP	$\uparrow 572 \leftrightarrow 0$		$\uparrow 572 \leftrightarrow 0$	$\uparrow 572 \leftrightarrow 0$
	MPW	$\uparrow 22 \leftrightarrow 0$	$\uparrow 22 \leftrightarrow 0$		$\uparrow 22 \leftrightarrow 0$
	MiW	$\uparrow 488 \leftrightarrow 0$	$\uparrow 488 \leftrightarrow 0$	$\uparrow 488 \leftrightarrow 0$	
No NS	**Def**		$\uparrow 54 \leftrightarrow 0$	$\uparrow 54 \leftrightarrow 0$	$\uparrow 54 \leftrightarrow 0$
	MaP	$\uparrow 100 \leftrightarrow 0$		$\uparrow 100 \leftrightarrow 0$	$\uparrow 100 \leftrightarrow 0$
	MPW	$\uparrow 40 \leftrightarrow 0$	$\uparrow 40 \leftrightarrow 0$		$\uparrow 40 \leftrightarrow 0$
	MiW	$\uparrow 92 \leftrightarrow 0$	$\uparrow 92 \leftrightarrow 0$	$\uparrow 92 \leftrightarrow 0$	



(a) NS_f (b) NS_p

Figure 14: Distribution of performance gap between the approach MaP (T in the key) and other approaches by considering the instances generated for the former when running $EA_{instance}$ with NS_f (14a) and NS_p (14b).

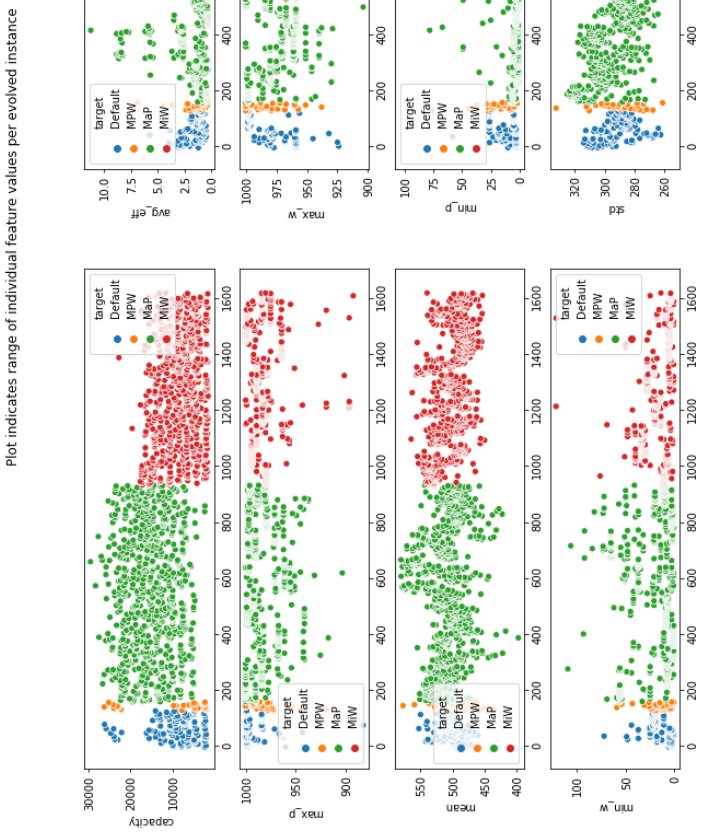


Figure 15: Feature distribution among instances for each solver in the portfolio of KP deterministic heuristics.

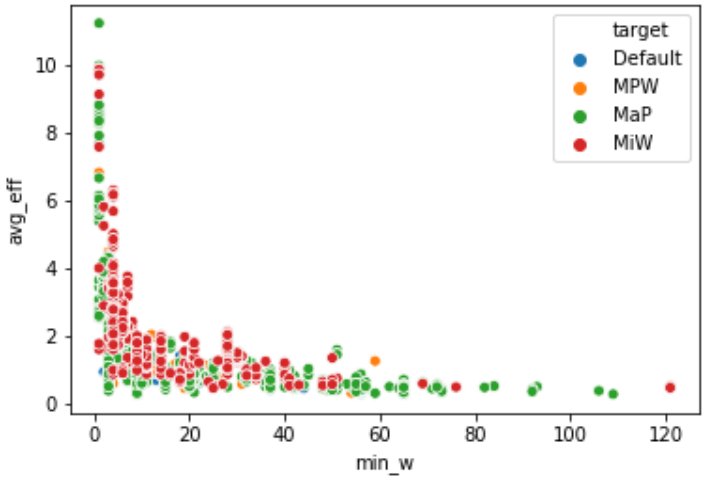


Figure 16: Relation between avg_eff / min_w

G Comparison with MAP-Elites

MAP-Elites creates an empty, N-dimensional map of elites with X being the solutions and P being their performances. After that, it generates G random solutions. Then, all subsequent solutions are generated from elites in the map by randomly selecting an elite x from the map X, creating a copy x' via mutation and recording its feature descriptor b' and performance p'. If the appropriate cell is empty or its occupants's performance is $\leq p'$, then store the performance of x' in the map of elites according to its feature descriptor b' and store the solution x' in the map of elites according to its feature descriptor b'.

Algorithm 3: MAP-Elites

Input: $P \leftarrow \emptyset$, $X \leftarrow \emptyset$
for *iter* $= 1 \rightarrow I$ **do**
  if *iter* $< G$ **then**
    $x' \leftarrow$ random_solution();
  end
  $x \leftarrow$ random_selection(X) ;
  $x \leftarrow$ random_variation(x) ;
  $b' \leftarrow$ feature_descriptor(x') ;
  $p' \leftarrow$ performance(x') ;
  if $P(b') = \emptyset$ *or* $P(b') < p'$ **then**
    $P(b') \leftarrow p'$;
    $X(b') \leftarrow x'$;
  end
end
return P and X

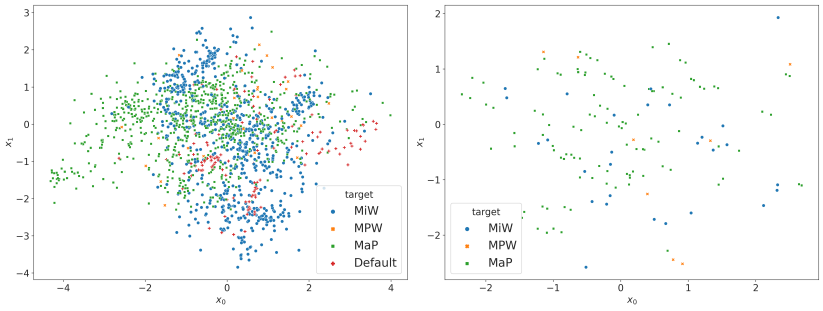


Figure 17: Instances generated per solver in the deterministic heuristics portfolio by NS_f (left) and $3r$-MAP-Elites (right).

Table 7: Parameter settings for MAP-Elites

Parameter	Value
Knapsack items (N)	50
Weight and profit upper bound	1,000
Weight and profit lower bound	1
Population size	10
Mutation rate	$1 \,/\, (N \times 2)$
Evaluations	10,000
Repetitions (R)	1
Distance metric	Euclidean Distance
Neighbourhood size (k)	3
Thresholds (t_a, t_{ss})	3.0, 1e-7
Resolution (r)	3, 5, 10, 15, 20, 25

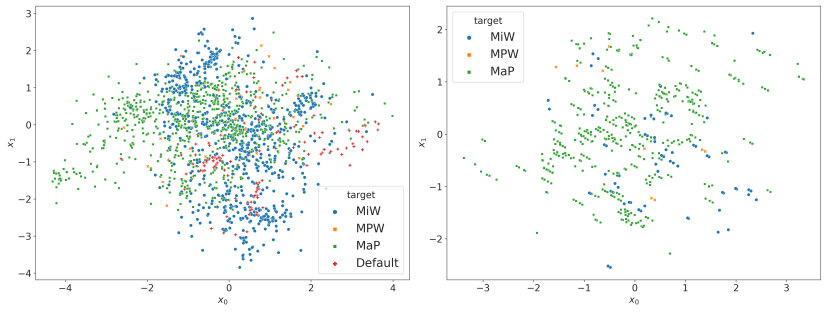


Figure 18: Instances generated per solver in the deterministic heuristics portfolio by NS_f (left) and $5r$-MAP-Elites (right).

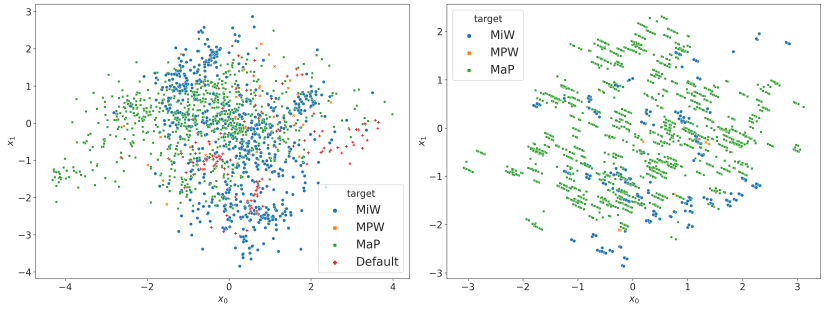


Figure 19: Instances generated per solver in the deterministic heuristics portfolio by NS_f (left) and $10r$-MAP-Elites (right).

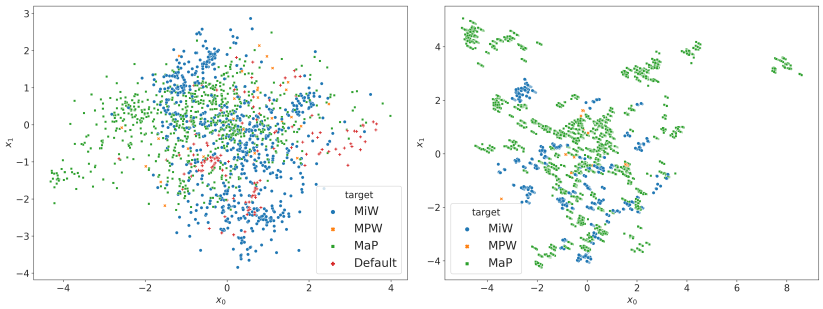


Figure 20: Instances generated per solver in the deterministic heuristics portfolio by NS_f (left) and $20r$-MAP-Elites (right).

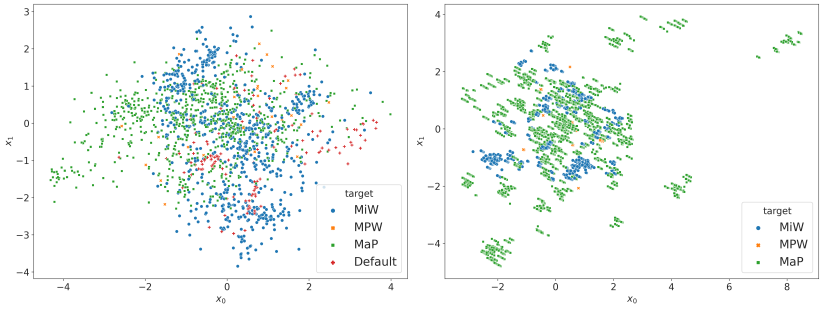


Figure 21: Instances generated per solver in the deterministic heuristics portfolio by NS_f (left) and $25r$-MAP-Elites (right).

Table 8: Space coverage using the U metric over the feature space for instances generated with a portfolio of heuristics.

Method	**U**
NS_f	0.6943
3r-MAP-Elites	0.5049
5r-MAP-Elites	0.5628
10r-MAP-Elites	0.6051
15r-MAP-Elites	0.6443
20r-MAP-Elites	0.6877
25r-MAP-Elites	0.6523

H Generation of Instances for the BP Domain

Table 9: Parameter settings for $EA_{instance}$ which evolves the diverse population of discriminatory BP instances.

Parameter	**Value**
Bin items (N)	120
Weight lower bound	20
Weight upper bound	100
Maximum bin capacity	150
z	2
Population size	10
Crossover rate	0.8
Mutation rate	1 / (N)
Evaluations	10,000
Repetitions (R)	1
Distance metric	Euclidean Distance
Neighbourhood size (k)	3
Thresholds (t_a, t_{ss})	1e-7

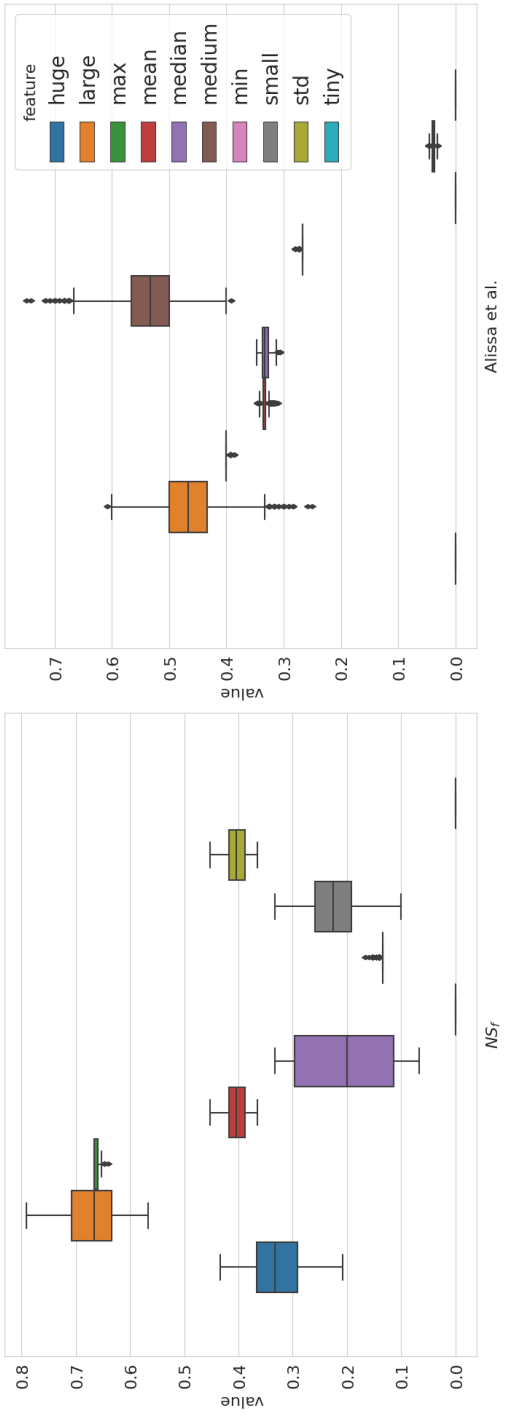


Figure 22: Feature distribution among instances for each generation method: NS_f (left) and Alissa et al. (2019) (right).