Learning Descriptors for Novelty-Search Based Instance Generation via Meta-evolution

Alejandro Marrero
Universidad de La Laguna
San Cristóbal de La Laguna, Spain
amarrerd@ull.edu.es

Eduardo Segredo
Universidad de La Laguna
San Cristóbal de La Laguna, Spain
esegredo@ull.edu.es

Coromoto León
Universidad de La Laguna
San Cristóbal de La Laguna, Spain
cleon@ull.edu.es

Emma Hart
Edinburgh Napier University
Edinburgh, UK
e.hart@napier.ac.uk

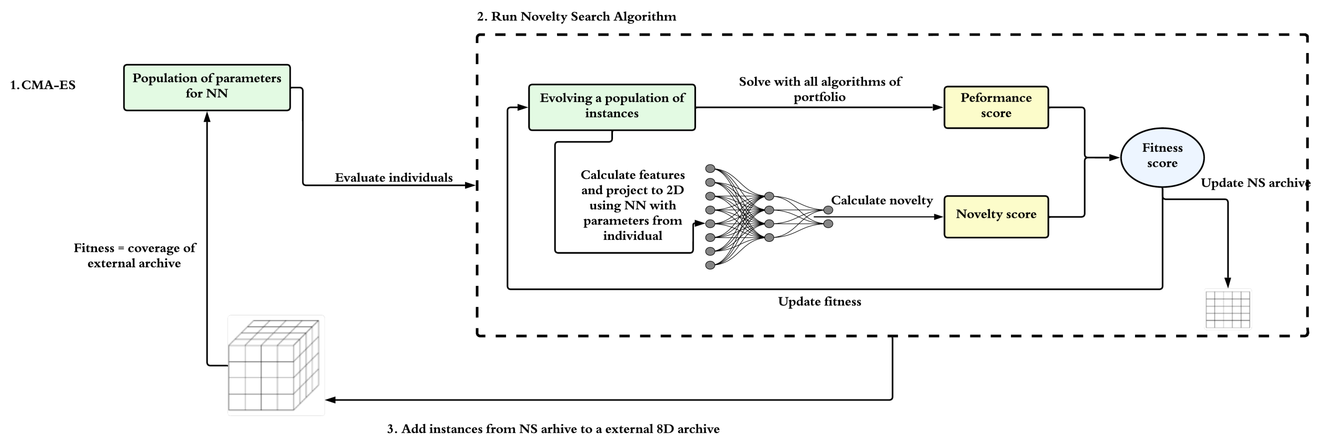
2. Run Novelty Search Algorithm
1. CMA-ES
Population of parameters for NN
Evaluate individuals
Evolving a population of instances
Solve with all algorithms of portfolio
Peformance score
Fitness score
Update NS archive
Calculate features and project to 2D using NN with parameters from individual
Calculate novelty
Novelty score
Fitness = coverage of external archive
Update fitness
3. Add instances from NS arhive to a external 8D archive

Figure 1: Meta-Learning of Descriptors: (1) CMA-ES is used to evolve the weights of an NN with fixed architecture that projects a feature-vector to 2D. (2) For each evolved NN (weight vector), a Novelty Search (NS) algorithm is run to evolve knapsack instances that are discriminatory w.r.t. a solver portfolio and diverse in the 2D projection defined by the corresponding NN. (3) The output of the NS algorithm is a set of instances which are projected into a fixed 8D hypercube. The coverage of the hypercube is assigned as the fitness of the corresponding NN weight vector. (4) CMA-ES repeats the above procedure for a fixed number of generations.

ABSTRACT

The ability to generate example instances from a domain is important in order to benchmark algorithms and to generate data that covers an instance-space in order to train machine-learning models for algorithm selection. Quality-Diversity (QD) algorithms have recently been shown to be effective in generating diverse and discriminatory instances with respect to a portfolio of solvers in various combinatorial optimisation domains. However these methods all rely on defining a *descriptor* which defines the space in which the algorithm searches for diversity: this is usually done manually defining a vector of features relevant to the domain. As this is a limiting factor in the use of QD methods, we propose a *meta-QD* algorithm which uses an evolutionary algorithm to search for a non-linear 2D projection of an original feature-space such that applying novelty-search method in this space to generate instances improves the coverage of the instance-space. We demonstrate the effectiveness of the approach by generating instances from the Knapsack domain, showing the meta-QD approach both generates instances in regions of an instance-space not covered by other methods, and also produces significantly more instances.

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654028

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**; *Discrete space search*; Neural networks; *Instance-based learning*.

KEYWORDS

Instance generation, instance-space analysis, knapsack problem, novelty search, evolutionary computation, neural-network

ACM Reference Format:
Alejandro Marrero, Eduardo Segredo, Coromoto León, and Emma Hart. 2024. Learning Descriptors for Novelty-Search Based Instance Generation via Meta-evolution. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3638529.3654028

1 INTRODUCTION

For any optimisation domain it is well-known that a portfolio of solvers will have complementary performance over a space of instances. To understand the strengths and weaknesses of different solvers over this space, it is crucial to have access to a large set of instances that ideally provide uniform coverage of an instance-space, i.e. the space of all possible instances. For many domains however, both the quantity and quality of available instances is poor [6]. On the one hand this limits benchmarking practices which enable a fair and unbiased comparison of solvers, and on the other, has an impact on machine-learning approaches such as algorithm-selection where large sets of unbiased training data are required to learn accurate models. To address this, a variety of methods have been proposed to generate new instances that are either *diverse* (cover a large region of the instance-space), *discriminatory* (elicit different performance from different solvers) or have both properties. For example, [13] use a full factorial design of experiments method, to generate problem instances for every combination of parameters in the knapsack domain. In [1], the automated parameter configuration tool iRace [17] is used to generate discriminatory instances for Constraint Programming and Mixed Integer Programming problems. On the other hand, Evolutionary algorithms (EAs) have been used to both generate new instances that fill gaps in an existing instance-space [31] (focusing on improving coverage) and to generate discriminatory instances [2, 26] in the bin-packing domain.

More recently, approaches from the class of EAs known as Quality-Diversity (QD) [28] algorithms have shown to be promising in being able to generate large sets of diverse and discriminatory instances in one run of an algorithm in travelling salesman, bin-packing and knapsack domains [4, 18, 20]. The intuition behind QD methods is to force exploration of a search-space to discover high-quality solutions, maintaining stepping-stones in the population that can lead to high performing solutions. The two most common QD algorithms, i.e. MAP-Elites [24] and Novelty-Search (NS) [14], rely on the definition of a multi-dimensional space in which to search for diversity, generally referred to as a *descriptor*. This is commonly defined by hand, for example using a feature-vector derived from the domain. However, the selection of appropriate features is challenging, and furthermore, high-dimensional feature-spaces can be very time-consuming to search. To address this issue when using QD to generate instances, [18] used the popular dimensionality-reduction method PCA [8] to reduce a feature-space to 2D, and then applied NS in the projected space, showing that new instances were found that were not discovered when searching for novelty in the feature-space directly. However, the choice of PCA as a dimensionality-reduction method is somewhat limiting given that it only finds linear relationships among variables.

While there are other 'off-the-shelf' dimensionality-reduction methods including non-linear approaches such as UMAP [22] that could be used instead of PCA, in this article we propose a method to *learn* a projection of features into a 2D space such that when an NS algorithm is applied in this new space, it produces a set of instances that maximise coverage of an instance-space. Specifically, the proposed *meta*-EA searches over a space of possible descriptors, where a descriptor is defined as the 2D coordinates output given by a neural network (NN) that takes a high-dimensional feature-vector as input.

The well-known continuous algorithm CMA-ES [12] is used to evolve the weight vectors of the NN that outputs the 2D descriptor vector, while the NS algorithm proposed by [20] is used to evolve diverse and discriminatory knapsack instances in the 2D space. Comparison of the results in terms of the number and diversity of instances discovered demonstrates: (1) the novel meta-EA finds a order of magnitude more discriminatory instances than previously proposed methods using hand-defined descriptors or PCA; (2) previous work using PCA-based descriptors is significantly outperformed in terms of the coverage metric; (3) instances that cover regions of the instance-space not covered by other NS methods from the literature are found (4) the method finds significantly more instances that are discriminatory for two of the heuristic solvers (Def and MPW) than previous approaches which struggled with these solvers.

2 RELATED WORK

As noted in the Introduction, there is a long history of EAs being used to generate instances for combinatorial optimisation domains [2, 26, 32]. Typically these methods focus on filling gaps in an instance-space, or finding instances that are discriminatory with respect to a portfolio of solvers. Recent work has shown that algorithms from the QD literature that were originally developed in the context of Evolutionary Robotics can be used to evolve instances that are both diverse and discriminatory in the Travelling Salesman Problem (TSP) and Knapsack (KP) domains [4, 18, 20]. However, as already pointed out, QD methods have some drawbacks, particularly with respect to the need to define an appropriate descriptor that sets the space in which diversity is measured. Recognising that searching for novelty in a high-dimensional space defined by a feature-vector is computationally expensive, [18] proposed it might be easier to search for novelty in a low-dimensional space, by using PCA to project to 2D with promising results in the KP domain. However, PCA provides a linear projection that might fail to capture relevant patterns in the data.

Looking beyond the field of instance-generation, the wider QD literature has embraced the idea of automatic generation of descriptors, particularly in the robotics domain. For example, in robotics, [11] propose a method of learning low-dimensional descriptors from raw sensory data using an encoder. The descriptors are used with a MAP-Elites algorithm where they are demonstrated to outperform hand-coded descriptors on three tasks. [3, 4] proposed a meta-QD algorithm that also learns descriptors for MAP-Elites by evolving an NN to create a low-dimensional encoding. This is evaluated on a set of standard benchmark functions from the continuous optimisation domain and to learn behaviours for a hexapod robot. Meyerson [23] proposed a method for learning behaviour descriptors for an NS algorithm used in a maze-navigation task that outperforms hand-coded descriptors. Their framework requires an underlying descriptor that provides a set of base features and learns a weighting over the features of the base descriptor for each new task.

Although not specifically related to learning descriptors, recent work which uses EAs to learn low-dimensional embeddings of data is also worthy of mention. For example, [30] used Genetic Programming to learn a 2D projection of instance-data in which the distance between pairs of instances in the projected space correlated with the distance between pairs of instances in the performance space. Lensen *et al.* [15] proposed a GP approach to manifold learning of machine-learning datasets. A multi-tree GP method is used to learn a 2D projection of data using a fitness function that attempts to maintain the same ordering between neighbours of an instance in both the original and low-dimensional spaces. The quality of a learned embedding is estimated via a proxy measure calculated post-evolution — applying a classifier to the newly projected data and measuring classification accuracy. In more recent work, the same authors propose further extensions that (1) optimise the embedding learned by GP to match a pre-computed UMAP embedding, and (2) optimise UMAP's own cost-function directly [29]. Most recently, they adapt their approach to consider how local structure within an embedding can be better reflected, proposing a modified fitness function that seeks to measure how well local topology is preserved by the evolved mapping [16].

Our proposed approach is mainly inspired by work in robotics that automatically learns descriptors for MAP-Elites using a meta-learning approach [4]. We adapt this method to work with NS in the context of instance-generation.

3 METHODS

This section provides specific details on each of the steps described above. The innovation of this work is the use of a *meta* framework to learn a projection of a feature-vector into a new space in which NS is able to find diverse, discriminatory instances. We accomplish this using an evolutionary strategy ($CMA-ES$) [12] which evolves the parameters of a NN model that outputs the coordinates of an instance in the new space in which novelty can be measured. The term *parameters* refers to the weights and bias terms of the layers of the NN [10]. The intuition is that rather than searching for novelty in a fixed space defined by an off-the-shelf method (as per [18]), we use the meta-algorithm to find the best space possible to apply NS in. The method overcomes some of the main disadvantages that PCA models have, such as presented by Marrero *et al.* [18]; e.g., (1) the data has to be standardised before applying PCA and (2) PCA assumes that the data hides a linear pattern between the variables.

We chose to use an NN to learn a new projection as there is ample evidence in the literate that the weights of an NN can be easily learned by an EA [33] and similar approaches have been utilised in the robotics literature [3]. Other choices such as Genetic Programming [16] or UMAP [22] could have been made — we return to this in Section 6. We use a fixed structure NN and use $CMA-ES$ to learn the weight vector W. The NN is constructed from three layers of eight, four and two units respectively. The first layer is defined by the number of features in the original vector that we wish to reduce, and the output layer by the need to project to 2D. Only one middle layer is set for simplicity. However, the NN could be defined with any number of hidden layers and various units per layer. The input layer receives an 8D feature descriptor which is passed to the four-neuron layer, the only hidden layer of the model, and then the output layer provides the 2D representation of the original 8D descriptor. The activation functions are fixed to Rectified Linear Unit (ReLU) [9] for the input and middle layers of the NN, and none for the output layer to avoid restricting the possible output values of NN [10]. NN is a fully connected architecture and the number of parameters for each layer can be calculated as $(n + 1) \times m$ where n is the number of input units and m is the number of output units. Therefore, the number of parameters to evolve W is the sum of the parameters for each layer of NN. Thus, $W = (8 + 1) \times 4 + (4 + 1) \times 2 = 46$. Consequently, each individual in the $CMA-ES$ population is a list of $W = 46$ floating point numbers which encodes the parameters of NN; i.e., 8 weights for the first layer and a bias term repeated 4 times followed by 4 weights plus a bias term repeated 2 times.

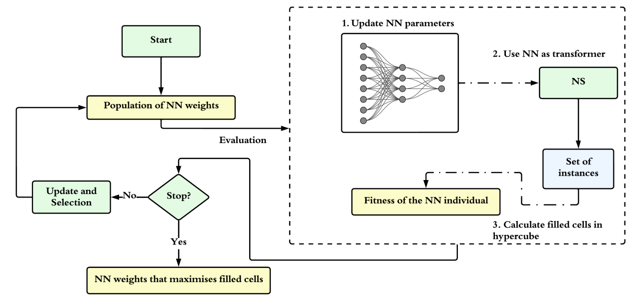

Figure 2: ***CMAES*** **algorithm for evolving a population of** ***NN*** **parameters.**

$CMA-ES$ starts by creating an initial population λ individuals sampling a multivariate normal distribution with an initial centroid defined by a vector of $|W|$ values equal to 0.05 and $\sigma = 1.0$, following common practices in the field [12]. Then, at each generation $g < G$ and for each individual i, $CMA-ES$ runs NS (see Figure 3) to generate instances for different solvers in a portfolio. The NS algorithm calculates the features of each instance and projects them to a 2D vector using NN with the parameters from individual i which is used to calculate novelty. The NS variant which uses an NN to reduce the descriptors is termed as NS_{NN} in the rest of the paper.

It is important to note that the evaluation procedure of the $CMA-ES$ algorithm is the most computationally expensive part of this work. For each individual i, a complete run of NS_{NN} is required to generate and collect sets of diverse and discriminatory instances for each of the four solvers in the portfolio. After that, the fitness of the individual in the $CMA-ES$ population is calculated as the

Table 1: Features ranges of 8D feature space hypercube. C is the maximum capacity of the knapsack, minimum weight/profit (min_w, min_p), maximum weight/profit (max_w, max_p), average item efficiency (also known as correlation), mean and standard deviation of values between profits and weights (mean, std).

Feature	Minimum	Maximum
C	711	30000
max_p	890	1000
max_w	860	1000
min_p	1.0	200
min_w	1.0	230
avg_eff	0.10	12.0
$mean$	400	610
std	240	330

space coverage by the set of instances over a predefined 8D feature space hypercube H. The use of a *fixed* reference space to calculate coverage (i.e. the fitness of each individual represented by W) is essential as the metrics obtained by the NS algorithm are defined in the projected space which is unique to each individual. The hypercube H is defined by the interval $[min_j, max_j]$ for each feature j of the 8D feature descriptors extracted from available datasets [18, 20]. Table 1 shows the intervals for each feature. For each dimension j in H, we create a set of B evenly spaced bins calculated over the interval $[min_j, max_j]$. Then, for every instance in the set, we map each jth feature value from its 8D descriptor into the corresponding bin of the jth dimension of H. A bin is considered to be covered if the NS_{NN} can generate at least one instance with a feature value located in such a bin. The fitness f of the individual is calculated as the sum of bins filled in H. After the evaluation, the $CMA-ES$ proceeds to update and create the next-generation population until reaching the maximum number of generations to perform G. The new population maintains the $\mu = \lambda/2$ best individuals from the parent population. Then, the algorithm returns the best individual found; i.e., the parameters for the NN which maximises the space coverage in H. Figure 2 shows the flow of the $CMA-ES$ algorithm. Moreover, Algorithm 1 presents the pseudo-code of $CMA-ES$, NS_{NN} and the procedure to calculate f, the space coverage in H.

3.1 Instance Representation and Novelty Descriptors

We apply the approach to generating instances for the zero-one KP domain, a commonly studied [5, 25] 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 satisfying a constraint that the weight remains under the knapsack capacity C. Each instance is described by an array of real numbers of size $2 \times N$, where N is the dimension (number of items) of the instance of the KP we want to create, with the weights and profits of the items stored at the even and odd positions of the array, respectively. We use NS_{NN} to generate instances with $N = 50$ items and a capacity C as 80% of the sum of

Algorithm 1: Meta-evolution with CMA-ES

```
Best ← ∅, $f_{best}$ ← 0, P ← create initial population()
for $g = 0$ to $G$ do
    O ← generate offspring population (P)
    for $i = 0$ to $\lambda$ do
        solution_set ← ∅
        for alg in portfolio do
            solution_set ← solution_set ∪ run $NS_{NN}(O_i)$
        end
        $f_i$ ← calculate space coverage(solution_set, H)
        if $f_i > f_{best}$ then
            Best ← $O_i$, $f_{best}$ ← $f_i$
        end
    end
    P ← update and select new population (O)
end
return Return Best
Procedure $NS_{NN}$(N, k, MaxEvals, portfolio, NN):
    initialise and evaluate(population, N, portfolio)
    archive = ∅
    for $i = 0$ to MaxEvals do
        offs = select and reproduce(population)
        offs = evaluate(offs, portfolio, archive, k, NN)
        population = update(population, offs)
        archive = update_archive(population, archive)
        solution_set = update_ss(population, solution_set)
    end
    return solution_set
end procedure
Procedure Evaluate(offs, portfolio, archive, k, NN):
    for instance in offs do
        for algorithm in portfolio do
            apply algorithm to solve instance R times;
            calculate mean profit of algorithm
        end
        descriptors ← reduce dimension(offs, NN)
        calculate the novelty score(descriptors, archive, k)
        calculate the performance score(offs)
        calculate fitness(offs)
    end
    return offs
end procedure
Procedure Space Coverage(solution_set, H):
    Cov = {∅ × 8};
    for instance i in solution_set do
        for $f_j$ in the 8D descriptor of i do
            $Cov_j = map(f_j, H)$;
        end
    end
    $f \leftarrow \sum_{j=1}^{8} |Cov_j|$;
    return f
end procedure
```

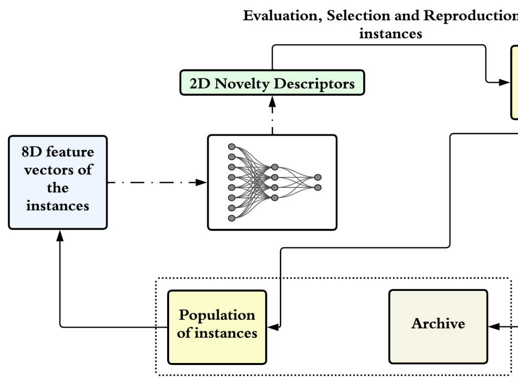
Evaluation, Selection and Reproduction of instances
2D Novelty Descriptors
Offspring population
8D feature vectors of the instances
Population of instances
Archive

Figure 3: Flow of NS_{NN} algorithm: the algorithm calculates the 8D feature vector and then uses NN to create the reduced 2D projected vector.

the weights of all items.[1] These parameters are chosen to reflect those used by *Marrero et al.* [18] to enable a direct comparison.

We search for an NN that optimises the projection of an n-dimensional feature-vector to two dimensions. The input to the NN is a set of 8 features, again taken directly from [18]: *capacity of the knapsack (C); maximum weight/profit (max_w, max_p); minimum weight/profit (min_w, min_p); average item efficiency (also known as correlation); mean distribution of values between profits and weights (mean); standard deviation of values between profits and weights (std).*

3.2 Algorithm Portfolio

To provide a fair comparison between the methods, we also use the same portfolio of solvers described in [18]. Although this portfolio may contain any number or kind of solvers (Marrero *et al.* [19] used a portfolio of EA solvers tuned with different configurations), the computational effort required to run each solver scales with the size of the portfolio and the time taken for each solver to run. Consequently, in these experiments, we use a portfolio of heuristic solvers [18, 27] to reduce computational effort. The portfolio includes: *Default* (Def) selects the first item available to be inserted into the knapsack; *Max Profit* (MaP) sorts the items by profit and selects those items with largest profit first; *Max Profit per Weight* (MPW) sorts the items according to efficiency (ratio between the profit and weight of each item) and selects those items with largest ratio first; and finally, *Min Weight* (MiW), which selects items with the lowest weight first [27].

3.3 Novelty Search

Although the NS element of our proposed architecture has been described in detail in previous work and is implemented without modification, we provide a brief overview of NS for completeness. The reader is referred to [18, 20] for a detailed description of this part of the framework.

NS was first introduced by Lehman *et al.* [14] as an attempt to mitigate the problem of finding optimal solutions in deceptive landscapes, with a focus on the control problems. The core idea replaces the objective function in a standard evolutionary search process with a function that rewards novelty rather than a performance-based fitness value to force exploration of the search-space.

Briefly, a population defining new instances is evolved that are discriminatory with respect to a target algorithm selected from a portfolio. Uniform crossover and Uniform One mutation (only one gene of the chromosome is updated) are applied to create new child instances. The evaluation function (NS_{NN} evaluation in Algorithm 1) assigns a fitness value to a new instance based on a linearly weighted combination of *novelty* and *objective fitness*. The former measures the novelty of an instance with respect to a user-defined *descriptor* that denotes a set of characteristics of a solution, e.g. a set of features describing the instance. For each solution, novelty is calculated as the average distance between its descriptor and that of its k nearest neighbours. Neighbours are calculated w.r.t the current population *and* an archive of previously discovered solutions, which is updated at each generation of the algorithm and provides a historical record of places visited in the search-space. On the other hand, objective fitness is calculated as the difference in performance between the target solver and the next best solver in the portfolio, with the goal of maximising this value. After each iteration, the archive of past solutions is updated in two ways. Firstly, we randomly select a sample of individuals from the current population and insert it in the archive with a probability of 1% following common practice in the literature [34]. Then, any individual from the current generation that has a novelty score greater than a predefined threshold t_a is also included into the archive. The algorithm returns a *set* of solutions. Figure 3 graphically describes the process and a pseudo-code is provided in procedure NS_{NN} of Algorithm 1.

4 EXPERIMENTAL EVALUATION

We use the $CMA-ES$ implementation from the framework *DEAP* [7] to evolve a population of NN parameters. Table 2 provides the parameter setting of $CMA-ES$ and NN_{NN} for the experimental evaluation. It is important to note that (1) λ and G parameters of $CMA-ES$ are set to ensure computation is tractable; and (2) the remaining parameters are set to the default values provided by DEAP [7, 12]. NS parameters are set according to previous work of *Marrero et al.* [18]. At the end of the runs, the individual that scores the highest f is compared with the other NS approaches proposed by *Marrero et al.* [18]; i.e., an NS algorithm that uses the 8D descriptor to search for novelty ($NS_{f_{8D}}$), three NS_{PCA} methods which transform a high-dimensional descriptor into a projected 2D descriptor using PCA and finally, an ensemble of the three NS_{PCA} methods known as NS_{PCA_c}. The source code of the NS was made available by *Marrero et al.* [21] at Github[2] and used directly. Moreover, the instances generated from NS_{PCA} methods [18] and used for comparison were downloaded from the publicly available repository.[3]

To provide a fair comparison between methods, we then run NS_{NN} using the same computational budget of [18] to generate

[1]The description of an instance follows the general method of [20].

[2]https://github.com/dignea/dignea
[3]https://github.com/PAL-ULL/ns_pca_gecco23

Table 2: Parameter settings for $CMA-ES$ which evolves a population of NN parameters which evolves the diverse population of instances.

Method	Parameter	Value
$CMA-ES$	λ	16
	μ	8
	G	250
	B	20
	Repetitions	10
NS_{NN}	Knapsack items (N)	50
	Weight and profit upper bound	1000
	Weight and profit lower bound	1
	Population size	10
	Crossover rate	0.8
	Mutation rate	1 / ($N \times 2$)
	Crossover operator	Uniform
	Mutation operator	Uniform One
	Generations	1000
	Repetitions (R)	1
	Distance metric	Euclidean Distance
	Neighbourhood size (k)	3
	Threshold (t_a)	0.0001

diverse and discriminatory instances for each solver in the portfolio using the best individual found by $CMA-ES$. The instances from NS_{NN} and the other methods were gathered into a single dataset, which will be analysed in the following section.

5 RESULTS

We run the framework ten independent times and calculate the distribution of the best f found. The aggregate fitness values after 10 runs of the framework are $min = 96$, $max = 113$, $mean = 103.9$, $median = 104.5$ and $sd = 4.721$ bins filled. In order to provide a direct comparison with previous work that aggregated results over 10 runs of an NS algorithm with pre-defined novelty vectors, additionally we run NS_{NN} 10 times using the projection obtained from the best NN obtained from the framework (i.e. the NN that filled 113 bins). However, the computational effort required to learn this projection was significant: NS was run 160,000 times to produce the best individual found by the framework, i.e. $(16 \times 250 \times 4) \times 10 = 160,000$ runs of NS. The source code used and the instances generated are available through the following Github repository.

To compare the different NS methods, we calculate the space coverage of the set of instances generated over H. Table 3 presents not only the number of cells filled by each method but also the number of instances generated per method and solver in the portfolio after 10 runs of the method. Note that the maximum fitness, i.e. the number of bins filled, that an individual can score is 160 since there are 20 bins for each of the 8 dimensions of H. Of interest here is that, although $NS_{f_{8D}}$ seems to provide better coverage of H, NS_{NN} produced more than 13 times instances after 10 runs considering all the solvers in the portfolio. At the same time, it can be observed that NS_{NN} is particularly useful in finding instances for solvers Def and MPW, for which the remaining methods struggle on. In the case

Table 3: Aggregated number of bins filled and instances generated by NS variant after 10 runs. Moreover, we include the bins filled and instances generated by the best individual found (*Best*) by the framework.

Method	Bins filled	Def	MPW	MaP	MiW	Combined
Best	113	907	817	830	0	2554
NS_{NN}	115	3312	8288	6456	3311	21367
$NS_{f_{8D}}$	125	131	31	774	687	1623
$NS_{PCA_{8D}}$	87	140	21	900	740	1801
$NS_{PCA_{6D}}$	94	39	20	960	900	1919
$NS_{PCA_{4D}}$	72	270	10	640	680	1600
NS_{PCA_c}	109	449	51	2500	2320	5320

of MPW, differences between NS_{NN} and the rest of methods are even more significant. Finally, the results from the *Best* individual found by the framework are also included in Table 3. Note that the results of *Best* are calculated considering only one run per solver in the portfolio of the NS.

These findings suggest that there is an overlap between NS_{NN} and the other methods, particularly $NS_{f_{8D}}$. To provide more insight into this finding, we calculate the overlap between NS_{NN}, $NS_{f_{8D}}$ and NS_{PCA_c} for each dimension in H. Table 4 shows the overlap among methods. Is it important to note that the maximum overlap for each dimension is $B = 20$. The results indicate that the instances provided by NS_{NN} provide less overlap with $NS_{f_{8D}}$ and NS_{PCA} in comparison to $NS_{f_{8D}}$ and NS_{PCA} among themselves. Also, note how the larger overlap between NS_{NN} and other approaches occurs in the features that are restricted in the parameter setting; i.e., the maximum and minimum profits/weights of the instances to be generated. In particular, NS_{NN} and $NS_{f_{8D}}$ filled the same 18 out of 20 bins considering the maximum profit dimension.

Table 4: Overlap of filled bins in H between the different NS methods. The NS part of the names is omitted to facilitate the reading.

Method	NN / PCA_c	NN/ $8D$	$8D$ / PCA_c
C	7	7	18
max_p	15	18	16
max_w	14	13	13
min_p	10	10	9
min_w	7	10	7
avg_eff	10	12	12
$mean$	13	14	13
std	17	17	17
Total Overlap	93	101	105

Finally, we provide a qualitative analysis of the results. The dataset of instances was used to obtain a 2D projection of the data using the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [22] algorithm for easy visualisation. The projection was created considering the 8D feature descriptor of every instance as input to UMAP, regardless of the method used to

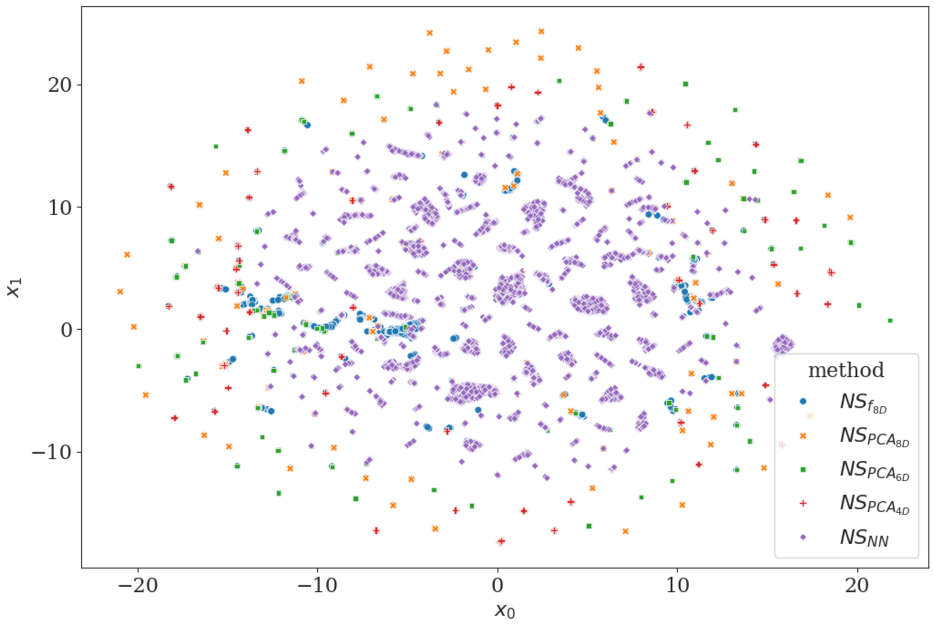

Figure 4: Instances generated by different NS algorithms represented in the same space. Purple diamonds represent the instances from NS_{NN}; yellow crosses represent the instances from NS_{PCA} projecting 8D to 2D; green squares for NS_{PCA} projecting 6D to 2D; red pluses for NS_{PCA} projecting 4D to 2D

generate it. The input was scaled before applying the UMAP. Results are shown in Figure 4. The figure shows that $NS_{f_{8D}}$ are clustered in a few regions of the space, and sometimes overlapping instances from NS_{PCA} variants, while NS_{NN} provides better coverage of the centre of the space with a large number of clustered instances. Finally, NS_{PCA} is also able to locate instances surrounding the $NS_{f_{8D}}$ and NS_{NN} ones.

6 CONCLUSIONS

Instance-generation methods play a critical role in facilitating benchmarking and providing data to train algorithm-selectors. However, as described in Sections 1 and 2, current methods all exhibit one or more weaknesses: they generate diverse *or* discriminatory instances; they require the need to manually define features or in the case of QD approaches, to define the dimensions of an archive defining the dimensions of diversity; methods used to reduce dimensionality only find linear relationships between variables.

To address this, we proposed a new *meta*-EA that uses evolution to learn a projection of a multi-dimensional feature space into a 2D space in which NS can be applied to find diverse and discriminatory instances: optimising the coverage of an external reference space drives the meta-algorithm to find spaces in which NS locates instances that maximise this metric. The results demonstrate significantly better performance in comparison to previous methods that use PCA to define the search-space in terms of the coverage metric. Although searching in the 8D space produces slightly higher coverage than NS_{NN}, it is clear that NS_{NN} produces an order of magnitude more instances. This has considerable benefit if the purpose of instance-generation is to generate data to train models or benchmark algorithms in terms of their strength and weaknesses.

Furthermore, NS_{NN} outperformed all other methods in its ability to find instances that were won by two solvers, Def and MPW. The other methods failed to generate large sets of instances for these particular solvers, specifically in the case of MPW. Table 4 indicates that although there is overlap between pairs of methods in terms of the bins covered, this never reaches 100%, showing that an *ensemble* of methods can be useful to maximise coverage of the space, with each method contributing something unique. NS_{NN} instances have less overlap with $NS_{f_{8D}}$ and NS_{PCA} in comparison to the overlap arising among those two methods. The most significant overlap between NS_{NN} and the other methods occurred in features with restricted parameter settings, such as the maximum and minimum profits/weights. Finally, qualitative analysis of the results obtained by visualising with UMAP revealed that $NS_{f_{8D}}$ instances were clustered in certain areas, occasionally overlapping with NS_{PCA} variants, whereas NS_{NN} cover more of the central space, with several clusters of instances.

It is clear that any kind of meta framework requires considerable computational effort, as the search algorithm used to generate instances in the 'inner' loop of the framework needs to be run completely in order to evaluate every individual in the outer-loop, which searches for the definition of the space. However, although we described results from running NS 10 times in the *best space* found by the framework in order to exactly mimic the set up used in previous work, Table 3 also shows that there is little to be gained in terms of coverage by running additional runs of NS in the best space found as this only increases coverage by two bins. Despite the additional expense, the vast increase in instances produced appears worth the trade-off in computational time.

An NN was an obvious choice of method to perform dimensionality reduction, in that it creates a non-linear mapping, and its weights can easily optimised via an EA. Future work could be directed towards tuning the hyper-parameters of the NN used, including its architecture. Other future work could consider other approaches to learning a low-dimensional projection: for example, the NN could be replaced with a more complex autoencoder to find a suitable latent space. Alternatively, genetic-programming (with an appropriate choice of functions) could be used to learn a function that maps a high-dimensional feature vector to a 2D space.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Özgür Akgün, Nguyen Dang, Ian Miguel, András Z Salamon, Patrick Spracklen, and Christopher Stone. 2020. Discriminating instance generation from abstract specifications: A case study with CP and MIP. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21–24, 2020, Proceedings 17*. Springer, 41–51.
[2] Mohamad Alissa, Kevin Sim, and Emma Hart. 2019. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 198–206.
[3] David M Bossens, Jean-Baptiste Mouret, and Danesh Tarapore. 2020. Learning behaviour-performance maps with meta-evolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 49–57.
[4] David M Bossens and Danesh Tarapore. 2022. Quality-Diversity Meta-Evolution: Customizing Behavior Spaces to a Meta-Objective. *IEEE Transactions on Evolutionary Computation* 26, 5 (2022), 1171–1181.
[5] Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. 2022. Knapsack Problems — An Overview of Recent Advances. Part I: Single Knapsack Problems. *Comput. Oper. Res.* 143, C (jul 2022), 13 pages. https://doi.org/10.1016/j.cor.2021.105692
[6] Nguyen Dang, Özgür Akgün, Joan Espasa, Ian Miguel, and Peter Nightingale. 2022. A framework for generating informative benchmark instances. *arXiv preprint arXiv:2205.14753* (2022).
[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.
[8] Karl Pearson F.R.S. 1901. LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 1* 2 (1901), 559–572.
[9] Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biol. Cybern.* 20, 3–4 (sep 1975), 121–136. https://doi.org/10.1007/BF00342633
[10] Aurelien Geron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media, Inc.
[11] Luca Grillotti and Antoine Cully. 2022. Unsupervised Behavior Discovery With Quality-Diversity Optimization. *IEEE Transactions on Evolutionary Computation* 26, 6 (2022), 1539–1552.
[12] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11, 1 (2003), 1–18.
[13] Jorik Jooken, Pieter Leyman, and Patrick De Causmaecker. 2022. A new class of hard problem instances for the 0–1 knapsack problem. *European Journal of Operational Research* 301, 3 (2022), 841–854.
[14] Joel Lehman and Kenneth O. Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation* 19, 2 (2011), 189–222. https://doi.org/10.1162/EVCO_a_00025
[15] Andrew Lensen, Bing Xue, and Mengjie Zhang. 2019. Can genetic programming do manifold learning too?. In *European Conference on Genetic Programming*. Springer, 114–130.
[16] Andrew Lensen, Bing Xue, and Mengjie Zhang. 2021. Genetic Programming for Manifold Learning: Preserving Local Topology. *IEEE Transactions on Evolutionary Computation* (2021).
[17] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.
[18] Alejandro Marrero, Eduardo Segredo, Emma Hart, Jakob Bossek, and Aneta Neumann. 2023. 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* (Lisbon, Portugal) *(GECCO '23)*. Association for Computing Machinery, New York, NY, USA, 312–320. https://doi.org/10.1145/3583131.3590504
[19] Alejandro Marrero, Eduardo Segredo, and Coromoto Leon. 2021. A Parallel Genetic Algorithm to Speed up the Resolution of the Algorithm Selection Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Lille, France) *(GECCO '21)*. Association for Computing Machinery, New York, NY, USA, 1978–1981. https://doi.org/10.1145/3449726.3463160
[20] Alejandro Marrero, Eduardo Segredo, Coromoto León, and Emma Hart. 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: 17th International Conference, PPSN 2022, Dortmund, Germany, September 10–14, 2022, Proceedings, Part I* (Dortmund, Germany). Springer-Verlag, Berlin, Heidelberg, 223–236. https://doi.org/10.1007/978-3-031-14714-2_16
[21] Alejandro Marrero, Eduardo Segredo, Coromoto León, and Emma Hart. 2023. DIGNEA: A tool to generate diverse and discriminatory instance suites for optimisation domains. *SoftwareX* 22 (2023), 101355. https://doi.org/10.1016/j.softx.2023.101355
[22] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426 [stat.ML]
[23] Elliot Meyerson, Joel Lehman, and Risto Miikkulainen. 2016. Learning Behavior Characterizations for Novelty Search. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) *(GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 149–156. https://doi.org/10.1145/2908812.2908929
[24] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).
[25] David Pisinger. 2005. Where are the hard knapsack problems? *Computers and Operations Research* 32, 9 (2005), 2271–2284. https://doi.org/10.1016/j.cor.2004.03.002
[26] Luis Fernando Plata-González, Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, Hugo Terashima-Marín, and Carlos A Coello Coello. 2019. Evolutionary-based tailoring of synthetic instances for the Knapsack problem. *Soft Computing* 23, 23 (2019), 12711–12728.
[27] Luis Fernando Plata-González, Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, Hugo Terashima-Marín, and Carlos A. Coello Coello. 2019. Evolutionary-based tailoring of synthetic instances for the Knapsack problem. *Soft Computing* 23, 23 (2019), 12711–12728. https://doi.org/10.1007/s00500-019-03822-w
[28] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. 2016. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* 3 (2016), 40.
[29] Finn Schofield and Andrew Lensen. 2021. Using Genetic Programming to Find Functional Mappings for UMAP Embeddings. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 704–711.
[30] Kevin Sim and Emma Hart. 2022. Evolutionary Approaches to Improving the Layouts of Instance-Spaces. In *Parallel Problem Solving from Nature – PPSN XVII: 17th International Conference, PPSN 2022, Dortmund, Germany, September 10–14, 2022, Proceedings, Part I* (Dortmund, Germany). Springer-Verlag, Berlin, Heidelberg, 207–219. https://doi.org/10.1007/978-3-031-14714-2_15
[31] Kate Smith-Miles, Jeffrey Christiansen, and Mario Andrés Muñoz. 2021. Revisiting where are the hard knapsack problems? via instance space analysis. *Computers & Operations Research* 128 (2021), 105184.
[32] Kate Smith-Miles, Jano Van Hemert, and Xin Yu Lim. 2010. Understanding TSP difficulty by learning from evolved instances. In *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers 4*. Springer, 266–280.
[33] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35.
[34] Paul A. Szerlip, Gregory Morse, Justin K. Pugh, and Kenneth O. Stanley. 2015. Unsupervised Feature Learning through Divergent Discriminative Feature Accumulation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, Austin, Texas, 2979–2985. https://doi.org/10.1609/aaai.v29i1.9601