DIGNEA: A tool to generate diverse and discriminatory instances for optimisation domains

Alejandro Marrero, Eduardo Segredo, Coromoto León
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna
San Cristóbal de La Laguna, Spain
`(amarrerd|esegredo|cleon)@ull.edu.es`

Emma Hart
School of Computing, Edinburgh Napier University
Edinburgh, United Kingdom
`e.hart@napier.ac.uk`

Abstract

This paper presents DIGNEA, a novel tool designed to generate diverse and discriminatory instances for various optimisation domains. DIGNEA utilizes an evolutionary algorithm-based framework that incorporates novelty search techniques to ensure the generated instances are not only varied but also useful for assessing different solvers. The tool, written in C++, is available as a repository and a Docker image, making it easily adaptable to different domains and solver types. Recently, a Python version has been released to facilitate the adoption by the research community. An application to the Knapsack Problem is showcased, demonstrating its effectiveness in generating meaningful test instances.

Diverse Instance Generator with Novelty Search and Evolutionary Algorithms – DIGNEA is a software that was first published in [1].

1 Introduction

The performance of optimisation algorithms is highly dependent on the nature of problem instances. As such, designing effective optimisation algorithms requires diverse and discriminatory test instances that highlight the strengths and weaknesses of different solvers. Traditional approaches to instance generation often fail to provide sufficiently diverse and discriminatory problem instances since their main focus is to increase the hardness of the instances [2, 3, 4], understanding hardness as the computational time required for the state-of-the-art exact solver to obtain the optimal solution for each instance. Back in the 70s Rice highlighted the necessity of diverse instance datasets to enable better algorithm evaluation and selection in the definition of The Algorithm Selection Problem (ASP) [5]. The ASP has garnered considerable attention over recent years [6] with the rise of Machine Learning approaches to predict either the performance of a given algorithm or the label of the best solver using large datasets of instances from the optimisation domain. However, the process of gathering sufficient instances that both cover the feature-space of instances and are discriminatory with respect to the solvers in the portfolio is considerably challenging.

DIGNEA [1] is a novel software tool designed to address this challenge by leveraging Quality Diversity (QD) [7] algorithms within an Evolutionary Algorithm (EA) framework to the problem of generating instances that are diverse with respect to a feature, instance or performance space and also discriminatory to a set of solvers of the users choice. This approach allows researchers to better analyse algorithmic behaviour and improve algorithm selection strategies. The software is available in both C++ and Python versions. The C++ version is provided as a repository and a Docker image, making it easily adaptable for various domains and solver types, whereas the Python library (known as DIGNEApy) is accessible at the Python Package Index (PyPi),[1] where it can be easily installed using the standard package manager 'pip'. The tool is designed to facilitate research in algorithm selection, instance space analysis, and performance benchmarking.

2 DIGNEA: Architecture and Functionalities

DIGNEA is structured around a modular framework that allows users to define problems, domains and solvers easily. Although there exist more classes in the framework, from the user perspective the key classes are the following:

- Domain: Establishes the instance generation space for a given optimisation problem. A domain defines all the characteristics of the combinatorial optimisation problem, such as KP or TSP, to which the instances will be generated, i.e., the number of items in the instance, the bounds for each item, what features describe an instance and in that case how to extract them, etc. Note that it is necessary to define a domain for every optimisation problem the user is interested in.
- Problem: While the domain specifies the characteristics of the instances for an optimisation problem, the Problem class in DIGNEA describes how to evaluate an instance for a specific domain, i.e., given a solution, how to calculate the constraints, fitness and/or objectives.
- Solver: Defines algorithms applicable to optimisation problems, from generic heuristics and meta-heuristics, to domain-specific algorithms.

Combined, both versions of DIGNEA include several domains and solvers alongside lots of functionalities. Some of them are KP, TSP and BP domains to generate instances, several solvers for each domain such as Parallel-EA, deterministic heuristics and exact methods. Additionally, DIGNEA provides lots of flexibility in terms of how the instances can be described, by a set of computed features (domain-dependent), a performance descriptor (portfolio-dependent) or even the full definition of the instance. While DIGNEA's extensibility is ensured through the use of design patterns and modern features of both C++ and Python programming languages, enabling easy adaptation to various domains, parallelization and high-performance computing capabilities is another of the goals of the software using libraries like OpenMP, MPI and Numpy.

3 Illustrative Example: The Knapsack Problem

To demonstrate DIGNEA's functionality,[2] the tool was applied to generate instances for the 0/1 KP with N = 50 items. A portfolio of deterministic heuristics was used as solvers, and instances were evolved by considering both features and performance-based descriptors. Figure 1 shows an example of source code to run the experiment using a feature-based descriptor and the instances generated by the method. The collection `solution_set` (Figure 1) contains the resulting instances with all the required information to perform exhaustive data analysis and solver performance benchmarking.

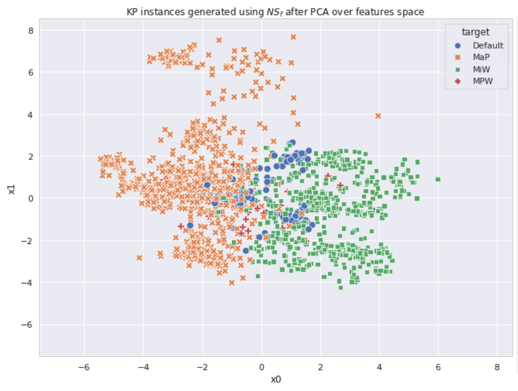
4 Impact and conclusions

DIGNEA significantly improves instance generation for optimisation research. It streamlines the ASP process by automating instance generation and solver performance evaluation in a single step, reducing computational overhead and human error. The tool's modularity and portability make it applicable across multiple domains. DIGNEA has been successfully deployed on several HPC systems, such as Archer2 and TeideHPC, demonstrating its scalability and efficiency demonstrating its scalability and efficiency, and has been successfully used in several research publications [8, 9, 10, 11]. Future developments will expand DIGNEA's applicability to additional domains, enhancing its utility for researchers and practitioners. The source code for both versions is available in GitHub.[3]

[1] PyPi: `https://pypi.org/`

[2] A tutorial is available in the DIGNEA docs.

[3] DIGNEA Org. at GitHub: `https://github.com/DIGNEA`



```
kp_domain = KnapsackDomain(50)
archive = Archive(threshold=3)
eig = EAGenerator(
    pop_size=128,
    generations=1000,
    domain=kp_domain,
    portfolio=portfolio,
    novelty_approach=NS(archive, k=15),
    solution_set=Archive(threshold=1),
    repetitions=1,
    descriptor='features',
    replacement=generational)
solution_set = eig(verbose=True)
```

Figure 1: Source code to generate KP instances with $N = 50$ items in DIGNEA and the resulting KP instances distributed over the features space. Colours reflect the target algorithm for which an instance was produced. Instances were generated using a features-based descriptor.

Acknowledgment

This work is part of the project PID2023-152614OB-I00 funded by MICIU/AEI/10.13039/501100011033 and by "ERDF/EU".

References

[1] A. Marrero, E. Segredo, C. León, and E. Hart, "DIGNEA: A tool to generate diverse and discriminatory instance suites for optimisation domains," *SoftwareX*, vol. 22, p. 101355, 2023.

[2] D. Pisinger, "Where are the hard knapsack problems?" *Computers and Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[3] K. Smith-Miles, J. Christiansen, and M. A. Muñoz, "Revisiting where are the hard knapsack problems? via Instance Space Analysis," *Computers & Operations Research*, vol. 128, p. 105184, 2021.

[4] K. Michalak, "Generating hard inventory routing problem instances using evolutionary algorithms," 2021.

[5] J. R. Rice, "The Algorithm Selection Problem," *Advances in Computers*, vol. 15, no. C, pp. 65–118, 1976.

[6] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[7] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[8] A. Marrero, E. Segredo, C. León, and E. Hart, "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*. Cham: Springer International Publishing, 2022, pp. 223–236.

[9] A. Marrero, E. Segredo, E. Hart, J. Bossek, and A. Neumann, "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*, ser. GECCO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 312–320.

[10] A. Marrero, E. Segredo, C. León, and E. Hart, "Learning Descriptors for Novelty-Search Based Instance Generation via Meta-evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 206–213.

[11] A. Marrero, E. Segredo, C. León, and E. Hart, "Synthesising Diverse and Discriminatory Sets of Instances Using Novelty Search in Combinatorial Domains," *Evolutionary Computation*, pp. 1–36, 08 2024.