Towards Automated Generation of Benchmark Instances with Diverse Solver Performance

Tianchen Wu
University of St Andrews, United Kingdom

Ian Miguel
University of St Andrews, United Kingdom

Nguyen Dang
University of St Andrews, United Kingdom

Abstract

Access to high-quality benchmark instances is essential for developing new algorithms and constraint models, as well as for comparing existing approaches. However, hand-crafted instances are often time-consuming to produce and may offer limited coverage of the relevant instance space. Consequently, automated benchmark instance generation has been widely studied across optimisation and related fields. In this work, we study AutoIG, a previously proposed constraint-based automated instance generation framework that combines automated algorithm configuration, constraint modelling, and constraint solving to generate instances of a specified difficulty for a target solver. A key limitation of AutoIG is that it does not explicitly encourage diversity among the generated instances. This can lead to benchmark sets that satisfy the desired difficulty requirements but provide limited variation in solver behaviour. As a preliminary step towards addressing this limitation, we investigate three simple mechanisms for promoting diversity in the performance space. That is, we aim to generate instances that not only meet a target difficulty level, but also induce diverse performance from the target solver. Experiments on two problem classes and two solvers show promising potential, while also revealing several open challenges, including how diversity should be defined and measured, and how diversity-promoting mechanisms affect the search process during instance generation.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Automated instance generation, benchmarking, instance diversity, constraint programming

1 Introduction

Empirical evaluation is central in solver development, and the choice of benchmarking instances is an important factor of consideration. Benchmark datasets are typically chosen by researchers to ensure comparability between studies and maintain standardisation. Many benchmarks exist for different combinatorial problem communities, whether it be through curating crowd-sourced data [29, 8], semi-automated generation processes [28], or domain specific fully automated instance generators [21] [16] [4]. However, many datasets become benchmarks due to availability or inclusion in competitions, rather than formal, analytical rigour. Consequently, the limited understanding of bias within these datasets risks poor generalisation of developed algorithms, especially to real-world problems [9] [3].

It is desirable that a benchmarking set contain a large quantity of diverse instances. This diversity can either be in *performance* space (e.g. time taken to solve) or in instance *feature* space (e.g. number of decision variables). For this study, we focus on performance space, i.e. the time it takes for a solver to either solve an instance (with proof of optimality for optimisation problems) or to prove unsatisfiability.

We will explore generating diverse instances within the performance space using AutoIG [6], a domain-independent constraint-based framework for generating problem instances. AutoIG

leverages constraint modelling to let users describe the components of a problem instance, and the constraints among those components, as a constraint model. This makes it broadly useful for generating benchmark instances that reflect problem-specific structure, while helping users explore solver performance across instances of a specific level of difficulty or instances that highlight performance differences between solvers. However, a central limitation of AutoIG is the lack of explicit mechanisms to ensure diversity of the instances generated.

This study uses AutoIG to generate instances with a specific level of difficulty for a target solver. Such instances are called *graded instances*, and are defined as solved within a targeted range of solving time. We introduce three simple mechanisms to promote diversity in the solver performance space, two of which make use of a global view of all graded instances generated so far, while the remaining one focuses on local information to promote instance subspaces where diverse solver performance potentially lie in. We empirically evaluate those mechanisms on two problem classes (the multi-agent collaborative construction problem [12] and the lot sizing problem [30]) and two solvers (chuffed [5] and OR-Tools [20]).

2 Related Work

The generation and selection of benchmark instances has long been studied in combinatorial optimisation and constraint solving. Benchmark libraries and instance generators have been developed across different problem domains, including scheduling [14], timetabling [7], nurse rostering [4], bin packing [19], knapsack [21], SAT [2], and AI planning [1]. These works highlight the importance of constructing instance sets that are not only sufficiently challenging, but also representative of the broader instance space.

A closely related line of work is Instance Space Analysis (ISA) [24, 26], which studies algorithm performance across feature-based representations of an instance space. ISA has been used both to analyse existing benchmarks and to generate new instances that fill under-represented regions of the space, often using evolutionary algorithms [25, 27]. This line of work demonstrates that diversity and coverage are important properties of benchmark sets, and that benchmark generation should go beyond simply producing hard instances.

Several automated instance generation approaches have also focused on producing instances with particular algorithmic properties, such as hardness or discriminating performance between solvers [11, 17]. AutoIG belongs to this line of work, but differs from ISA-style approaches in its use of constraint modelling to describe valid instance spaces and constraint solving to generate instances from them.

3 Automated Instance Generation using AutoIG

In this section, we give a brief overview of how AutoIG supports generating graded instances, i.e., instances with a specific level of difficulty for a target solver. AutoIG takes as input a target solver and two constraint models. The first constraint model specifies the combinatorial problem that we want to generate instances for. The second one is a *generator* model, which describe components of the problem instances and any constraints among those components that reflect the desired problem structure. Graded instances, whose solving times fall within a desired range by the target solver and whose components satisfy the desired problem structure described in the generator model, are found using a combination of IRACE [13], an automated algorithm configurator, and constraint solving. The role of IRACE is to *tune* the generator model's parameters so that we can get as many graded instances from the generator as possible. Concretely, at each iteration of the instance generation process, IRACE

is used for sampling a number of *generator configurations* – instances of the generator model. Each generator configuration covers a subspace of the problem instance space under study, and a problem instance covered by a generator configuration can be produced by solving this generator configuration using any constraint solver. AutoIG then evaluates the quality of those generated instances by running the target solver on the instances to see whether the solving time is within the desired graded range. After this evaluation step, each configuration receives a score, which is given back to IRACE to help it decide which configurations are more likely to generate graded instances (and therefore will be used to guide the sampling in the next iterations). In the rest of this section, we first describe the irace's tuning process of generator configurations, then describe the process by which these configurations are scored.

3.1 irace's Tuning Process

IRACE [13] is an automated algorithm configuration tool for optimising configurations of a parameterised algorithm. At the core of IRACE is the idea of *racing* [15], which uses ranked statistical tests (e.g. the Friedman test) to discard underperforming configurations early. Each race is split into multiple steps (called *instances* in IRACE, though we use *steps* to avoid ambiguity), where each surviving configuration is evaluated and ranked against others in the same step. IRACE uses these rankings to identify and retain the best-performing, or *elite*, configurations during races. Between races, the sampling distributions are updated using the elites' values and new configurations are generated from these updated distributions. Races are repeated until an evaluation budget is consumed, after which the best configurations are returned.

By default, IRACE uses an elitist racing mode to prevent elite configurations from being discarded due to an unlucky series of evaluations. To achieve this, elite configurations are not discarded before non-elite configurations have been evaluated on the same steps. Each race starts with a number of new steps where all configurations are evaluated. Subsequently, a shuffled set of previous steps is appended, preserving prior scores of elite configurations while the new configurations are evaluated. Only when the new configurations are statistically better are the existing elites discarded. As such, elitist racing mode should lead to faster convergence. However, the elitist racing mode assumes that scores from previous evaluations remain unchanged. This assumption fails as we begin to evaluate instances based on diversity. Configurations that initially expand coverage may no longer do so as areas of the performance space become saturated. Therefore, the historical scores become obsolete and may cause IRACE to retain configurations longer than desired.

3.2 Scoring the Generator Configurations

Generator configurations are scored in a two-step process. First, the generator configuration is solved using a constraint solver (the MINION solver [10] is used by AutoIG) to get a problem instance. If no problem instance is produced for any reason (e.g., timeout or the generator configuration is unsatisfiable), the configurator receives a high penalty score. Otherwise, the generated problem instance is given to the target solver. If the solving time meets the user-defined graded range, the instance is marked as graded and the generator configuration receives a reward score.

The original AutoIG framework assigns identical reward scores and treats all graded instances equally. This scoring scheme does not actively encourage diversity in the instances generated. In the next section, we address this limitation by proposing three scoring mechanisms to promote diversity during irace's tuning process.

4 Diversity Scoring During Instance Generation

To encourage diversity in the performance space during the instance generation process, we introduce three scoring methods to AutoIG. Two of these are *global* methods, which measure the impact of new instances on diversity in reference to all previously generated instances. The third method is *local*, which only tracks the history of instances generated by each configuration. We implement this dual approach as the origin of diversity within the generation process remains unknown. It may be configuration-specific, where a single configuration defines a complex instance space that inherently generates diverse instances. Alternatively, if it is driven by the broader instance space, achieving diversity requires exploring and retaining varied configurations. Having established the rationale for this dual approach, the remainder of this section details the two global methods, followed by the local one.

4.1 Global Change in Normalised Entropy

Our first metric measures the impact of newly generated instances on the uniformity of the distribution via Shannon entropy [23]. To achieve this, we discretise the graded time range into B bins and calculate the frequency of instances in each bin, represented by $\mathbf{c} = (c_1, c_2, \ldots, c_B)$. From this, the normalised Shannon entropy is defined as:

$$H_{norm} = -\frac{1}{\log(B)} \sum_{j=1}^{B} p_j \log(p_j)$$

where n_m is the total number of instances and $p_j = \frac{c_j}{n_m}$. The difference between entropy values before and after adding a new instance serves as the evaluation score. Specifically, instances that decrease uniformity are penalised, while those that increase it are rewarded.

4.2 Maximising Distance to Closest Neighbours

A limitation of the first metric is that it ignores the distance between instances and focuses on the uniformity of their frequencies. For example, in a five-bin distribution with one instance in each of the first two bins, the entropy metric is indifferent to whether an instance is added to the adjacent third bin or the distant fifth bin. However, intuitively, the latter is preferable due to its distance from existing instances. To capture this, our second metric uses distance as a proxy for diversity, aiming to maximise the separation between instances.

Because IRACE requires a single numeric score that is comparable when evaluating configurations in parallel, our second metric encodes instance separation into a single optimisable score. We first discretise the graded time range into B bins, each represented by a single digit in base B. When a new graded instance is generated, we calculate the absolute temporal distance to its k closest neighbours, mapping each distance to its respective bin digit. By sorting these digits in ascending order and concatenating them, we create a single numeric score that can be lexicographically compared. By optimising this score, the process is forced to find instances that maximise the distance to its nearest neighbours.

4.3 Local Normalised Entropy

While the first two metrics encourage diversity in the broader configuration space, the third metric focuses on finding configurations with strong diversity. For this approach, we again employ Shannon entropy. However, rather than evaluating the global history of

graded instances, we apply it on only instances local to an individual configuration. In addition, instead of measuring the change in entropy caused by a new instance, we use the absolute entropy value as the score for configurations. This prioritises configurations that can independently generate a uniform distribution of instances.

5 Experimental Setup

For the empirical evaluation, we select two problem classes, multi-agent collaborative construction problem (MACC) [12] and Lot-Sizing [30], alongside two solvers: chuffed [5] and OR-Tools [20]. All experiments are conducted on a high-performance computing cluster with AMD EPYC 9224 processors. While each experiment uses up to 24 CPU cores for parallel evaluation, each solver invocation is limited to a single core via the MiniZinc toolchain [18] and memory capped at 8 GB via the RUNSOLVER tool [22]. Every experiment is allocated an evaluation budget of 2000 runs, and the graded time range is set between 10 seconds and 20 minutes.

For the global entropy method, we divide the time range into 120 bins, obtaining approximately 10 seconds per bin. While the choice of 10 second intervals is arbitrary, this choice balances the trade-off between overly fine-grained binning that fails to distinguish between heavily skewed distributions and non-skewed distributions and excessively coarse-grained binning that obscures meaningful variations within the bins. For the global lexicographic metric, we discretise the time differences into ten bins for simplicity during concatenation as we remain in base ten. Finally, for this metric, we evaluate the distance to the nearest ten neighbours so that enough differences exist to make a comparison while staying within integer encoding limits.

The evaluation is split into two phases. The first set of experiments determines whether the elitist racing mode should remain enabled. To test this, we generate graded instances for each solver-problem combination using the global lexicographic measure, with elitist racing both enabled and disabled.

Informed by the first set of experiments, a second set of experiments compare the three proposed scoring methods using the elitist racing mode. For each solver-problem combination, we generate graded instances using each of the three metrics, alongside a default baseline without any diversity metric.

6 Experimental Results

First, we compare the empirical results of enabling versus disabling elitist racing mode, concluding that it should remain enabled. Next, we analyse the outcomes of the second phase of experiments, visualised in histograms. Finally, we quantitatively assess the diversity of the generated instance sets and discuss the open challenges associated with any approach towards quantification.

6.1 Disabling Elitist Racing Mode

The distribution and volume of generated instances for both modes can be seen in figures 1 and 2. While Figure 2 demonstrates non-elitist racing resulting in significantly broader coverage over the time range, covering the 200-400 seconds range, this behaviour was not replicated across the other three experiments. The other experiments align more closely with the pattern seen in Figure 1, where disabling elitist racing mode achieves similar coverage to elitist racing mode, but consistently generates fewer instances per bin.

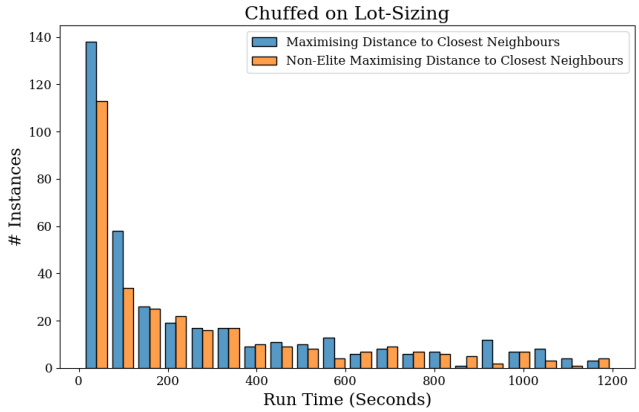

Figure 1 Distribution of chuffed solving times for the lot-sizing problem, comparing elitist racing enabled and disabled. Solving times are grouped into 20 bins of approximately one minute each.The y-axis shows the number of generated instances in each bin.

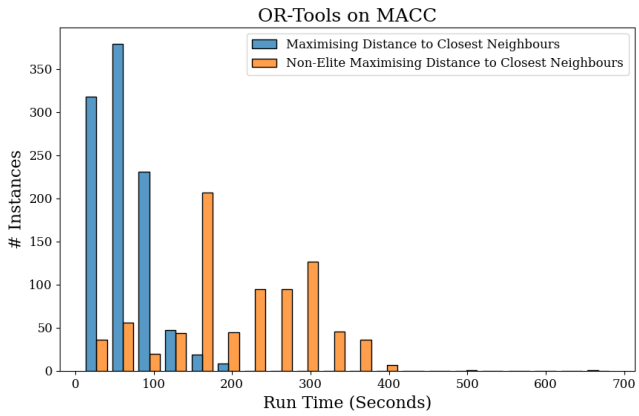

Figure 2 Distribution of OR-Tools solving times for the MACC problem, comparing elitist racing enabled and disabled. Solving times are grouped into 20 bins of approximately one minute each.The y-axis shows the number of generated instances in each bin.

Notably, across the three similar experiments, maximising the distance to nearest neighbours is able to generate instances that spanned the graded range, while for the outlier experiment it failed to do so. This suggests that combining non-elitist racing with a diversity metric may be beneficial when a problem is particularly trivial for a solver. However further experimentation is needed to fully explore this phenomenon.

Without further data to justify a switch to non-elitist racing mode, we leave elitist racing mode enabled for the next phase of experiments.

6.2 Comparing Three Diversity Scoring Metrics

We move on to evaluate the effectiveness of the three diversity scoring metrics proposed in Section 4 (with elitist racing mode enabled for all experiments).

For all experiments, the solving time distributions are heavily skewed towards computationally easy instances. This is particularly pronounced for the MACC problem (see Figures

	Base	Maximising Distance to Closest Neighbours	Global Entropy	Local Entropy
Chuffed MACC	773	649	1035	1066
Chuffed Lot-Sizing	528	380	501	470
OR-Tools MACC	1125	1004	949	871
OR-Tools Lot-Sizing	452	394	504	573

Table 1 Total number of graded instances generated for each experiment

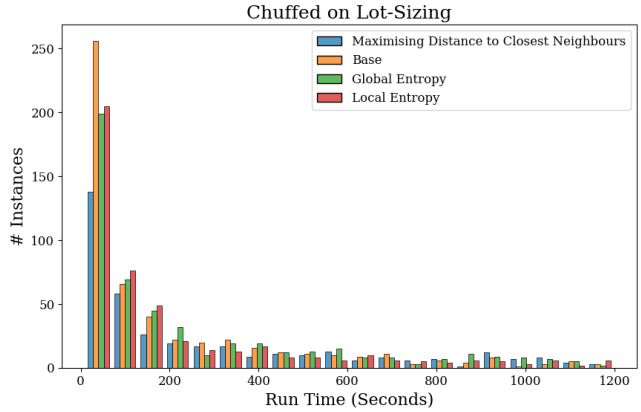

Figure 3 Distribution of chuffed solving times on the Lot-Sizing problem for three diversity metrics and the baseline (AutoIG without any diversity scoring), discretised into 20 approximately one-minute bins. The y-axis shows the number of generated instances in each bin.

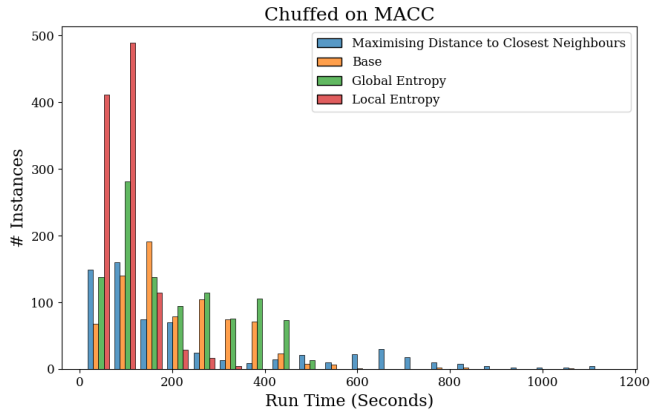

Figure 4 Distribution of chuffed solving times on the MACC problem for three diversity metrics and the baseline, discretised into 20 approximately one-minute bins.

4 and 6), where AutoIG fails to generate instances above 600 seconds using most metrics. For such computationally simple problems, utilising diversity metrics seems to encourage the generation of more difficult instances. Examples include applying the maximising distance to closest neighbours (or the global lexicographic) metric to chuffed or global entropy to OR-

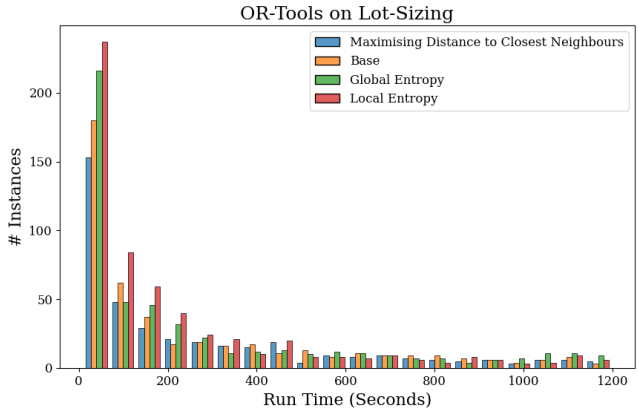

Figure 5 Distribution of OR-Tools solving times on the Lot-Sizing problem for three diversity metrics and the baseline, discretised into 20 approximately one-minute bins.

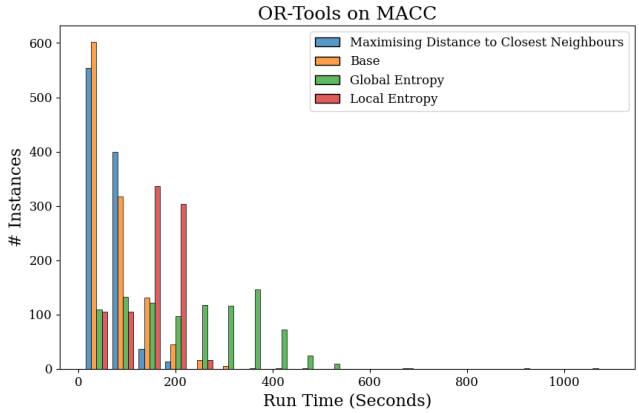

Figure 6 Distribution of OR-Tools solving times on the MACC problem for three diversity metrics and the baseline, discretised into 20 approximately one-minute bins.

Tools. In the chuffed example, instances were found beyond the baseline range, populating nearly every minute-sized bin.

However, this increased coverage tends to be at the cost of the total volume of instances generated. As is demonstrated with the previous two mentioned experiments, generating 124 and 176 fewer instances in comparison to the baseline respectively (see Table 1). There are exceptions to this pattern. For example, applying the individual entropy metric to the Lot-Sizing problem with OR-Tools (Figure 5) maintains the same coverage as the baseline, but increases the number of instances generated.

A distinction will need to be made in future experiments based on whether the baseline is independently able to cover the graded time range. Perhaps a smaller range of time is necessary to explore how the metrics perform in a saturated time range.

6.3 Diversity Measures for Graded Instance Sets

While analysing histograms visually for diversity is tenable for a small number of experiments, some quantification of this diversity is required to make analysis scalable. The difficulty arises when trying to pin down exactly what to include in such a quantity. Intuitively, even coverage over the entire graded time range is the most diverse and should be rewarded. In addition, the total number of instances needs to be considered to avoid a metric winning by reducing skew through reducing the number of instances generated. Following this intuition, we develop two quantifications metrics which we describe below.

6.3.1 Scaled Normalised Shannon Entropy

Normalised Shannon entropy naturally arises as one of the methods when trying to measure evenly distributed coverage. However, Shannon entropy does not include a measure for the number of instances. Therefore, to create a metric where both are considered, we multiply the entropy score with the log-scaled volume of instances:

$$score = H_{norm} \log(1 + n_m)$$

6.3.2 Softened Lexicographic Coverage Score

A fundamental limitation of the entropy score is that it does not specifically look at the filling of bins, rather assuming that uniformity will naturally distribute instances between bins. To create a metric that prioritises coverage, we define a lexicographic coverage profile:

$$\mathbf{L}(m) = (L_1(m), L_2(m), L_3(m), \ldots)$$

where $L_k(m)$ represents the number of bins containing at least k generated instances. So $L_1(m)$ is the number of bins containing at least one generated instance, $L_2(m)$ is the number of bins containing at least two generated instances and so on. By comparing this score between instance sets lexicographically, we are able to prioritise the coverage of bins a layer at a time.

However, relying solely on the lexicographic coverage profile will be overly penalising to instance sets that may have missed one bin but otherwise provides good coverage and volume. To soften the penalty, we introduce a tunable variable $\rho \in (0, 1)$ that defines the diminishing returns of repeated bin coverage. This soft coverage metric is defined as:

$$D_{\text{soft}} = \frac{1}{B} \sum_{k=1}^{K} \rho^{k-1} L_k, \quad 0 < \rho < 1$$

6.3.3 Post-hoc Diversity Analysis

We now apply the post-hoc diversity metrics to the instance data from the experiments (Figure 7 and 8), with $\rho = 0.5$. The baseline method scores well on either metric, usually coming in second place. The global entropy method performs best between all the different configurations and across metrics, while the local entropy tends to perform the worst. The implication is that for the studied problem classes, diversity arises from the generator instance space, and not from individual generator instances. Improving the rate at which new valid generator configurations are discovered will be critical to improve upon the diversity of the results.

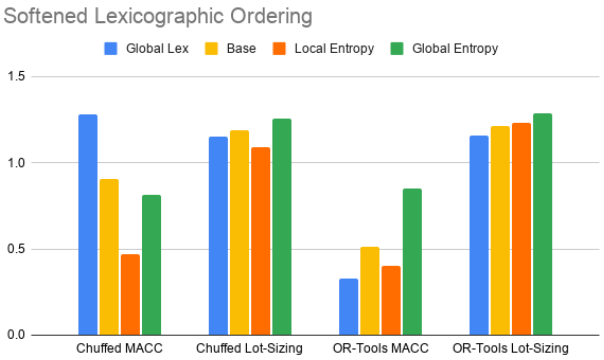

Figure 7 Post-hoc diversity analysis using the softened lexicographic ordering metric

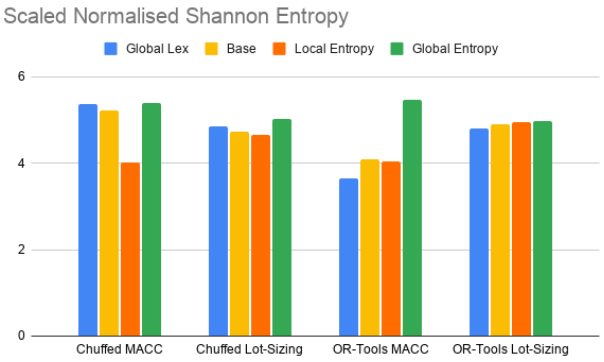

Figure 8 Post-hoc diversity analysis using the scaled normalised Shannon entropy metric

It is notable that according to the histogram in Figure 4, the global lexicographic method should demonstrate much greater diversity for the MACC problem using chuffed, yet only the softened lexicographic ordering metric demonstrates this clearly. This is due its emphasis on coverage first, which is useful for distinguishing between skewed and non-skewed datasets.

It should also be acknowledged that these metrics do not consider distance between instances: a distribution filling adjacent bins is scored similarly to one filling distant bins. Further work is needed to develop a distance aware metric.

7 Conclusion and Future Work

In this study we introduce three diversity metrics, two global and one local, to the AutoIG framework to encourage diverse instance generation. We first explore the effects of disabling elitist racing mode in combination with the global lexicographic metric. Having seen inconsistent results, we set the racing mode back to its default and leave further investigation as future work. We then empirically evaluated the three methods on a combination of the MACC and Lot-Sizing problems with the chuffed and OR-Tools solvers. While some of the metrics were visually better than the baseline, the best method was inconsistent across different experiments. Further experiments using more problem classes are required to decide on the efficacy of the methods.

In addition, two post-hoc diversity analysis metrics were created in order to find a scalable

quantifier for diversity. Applying these metrics to the results, it seems that global entropy dominates the other methods, with the baseline coming in second. Further improvement can be done by introducing distance awareness into the metrics to avoid clustered distributions.

References

1 Özgür Akgün, Nguyen Dang, Joan Espasa, Ian Miguel, András Z Salamon, and Christopher Stone. Exploring instance generation for automated planning. In *ModRef 2020-The 19th workshop on Constraint Modelling and Reformulation*, 2020.

2 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Community structure in industrial SAT instances. *Journal of Artificial Intelligence Research*, 66:443–472, October 2019. doi:10.1613/jair.1.11741.

3 Sam Bayless, Dave A. D. Tompkins, and Holger H. Hoos. Evaluating instance generators by configuration. In Panos M. Pardalos, Mauricio G.C. Resende, Chrysafis Vogiatzis, and Jose L. Walteros, editors, *Learning and Intelligent Optimization*, pages 47–61, Cham, 2014. Springer International Publishing.

4 Edmund K. Burke and Tim Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014. URL: https://www.sciencedirect.com/science/article/pii/S0377221714000605, doi:10.1016/j.ejor.2014.01.039.

5 Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. The Chuffed CP solver. https://github.com/chuffed/chuffed.

6 Nguyen Dang, Özgür Akgün, Joan Espasa, Ian Miguel, and Peter Nightingale. A framework for generating informative benchmark instances. *LIPIcs, Volume 235, CP 2022*, 235:18:1–18:18, 2022. arXiv:2205.14753 [cs]. doi:10.4230/LIPIcs.CP.2022.18.

7 Andreas Drexl and Frank Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102(1):193–214, 1997. URL: https://www.sciencedirect.com/science/article/pii/S0377221796002093, doi:10.1016/S0377-2217(96)00209-3.

8 Nils Froleyks, Marijn Heule, Ashlin Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artificial Intelligence*, 301:103572, 2021. doi:10.1016/j.artint.2021.103572.

9 Zijie Geng, Xijun Li, Jie Wang, Xiao Li, Yongdong Zhang, and Feng Wu. A deep instance generative framework for milp solvers under limited data availability. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

10 Ian P Gent and Chris Jefferson. Minion: A fast, scalable, constraint solver[1]. In *ECAI 2006: 17th European Conference on Artificial Intelligence, August 29-September 1, 2006, Riva del Garda, Italy: including Prestigious Applications of Intelligent Systems (PAIS 2006): proceedings*, volume 141, page 98. Ios Press, 2006.

11 Marc Goerigk and Stephen J. Maher. Generating hard instances for robust combinatorial optimization. *European Journal of Operational Research*, 280(1):34–45, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0377221719306009, doi:10.1016/j.ejor.2019.07.036.

12 Edward Lam, Peter J. Stuckey, Sven Koenig, and T. K. Satish Kumar. Exact approaches to the multi-agent collective construction problem. In *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-La-Neuve, Belgium, September 7–11, 2020, Proceedings*, page 743–758, Berlin, Heidelberg, 2020. Springer-Verlag. doi:10.1007/978-3-030-58475-7_43.

13 Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi:10.1016/j.orp.2016.09.002.

14 Carlos March, Christian Pérez, and Miguel A. Salido. A novel instance generator for benchmarking the job shop scheduling problem. In Hamido Fujita, Richard Cimler, Andres Hernandez-Matamoros, and Moonis Ali, editors, *Advances and Trends in Artificial Intelligence. Theory and Applications*, pages 413–424, Singapore, 2024. Springer Nature Singapore.

15 Oden Maron and Andrew W. Moore. The racing algorithm: Model selection for lazy learners. *Artif. Intell. Rev.*, 11(1–5):193–225, February 1997. doi:10.1023/A:1006556606079.

16 Alejandro Marrero, Eduardo Segredo, Coromoto León, and Emma Hart. A novelty-search approach to filling an instance-space with diverse and discriminatory instances for the knapsack problem. In Günter Rudolph, Anna V. Kononova, Hernán Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tušar, editors, *Parallel Problem Solving from Nature – PPSN XVII*, page 223–236, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-14714-2_16.

17 Alejandro Marrero, Eduardo Segredo, Coromoto León, and Emma Hart. Synthesising diverse and discriminatory sets of instances using novelty search in combinatorial domains. *Evolutionary Computation*, 33(1):55–90, 03 2025. arXiv:https://direct.mit.edu/evco/article-pdf/33/1/55/2464257/evco_a_00350.pdf, doi:10.1162/evco_a_00350.

18 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Eneko Osaba, Esther Villar-Rodriguez, and Sebastián V. Romero. Benchmark dataset and instance generator for real-world three-dimensional bin packing problems. *Data in Brief*, 49:109309, 2023. URL: https://www.sciencedirect.com/science/article/pii/S2352340923004286, doi:10.1016/j.dib.2023.109309.

20 Laurent Perron and Frédéric Didier. CP-SAT. URL: https://developers.google.com/optimization/cp/cp_solver/.

21 David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005. URL: https://www.sciencedirect.com/science/article/pii/S030505480400036X, doi:10.1016/j.cor.2004.03.002.

22 Olivier Roussel. Controlling a solver execution: the runsolver tool. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011. HAL ID: hal-00868234. doi:10.3233/SAT190083.

23 C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

24 Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014. URL: https://www.sciencedirect.com/science/article/pii/S0305054813003389, doi:10.1016/j.cor.2013.11.015.

25 Kate Smith-Miles, Jeffrey Christiansen, and Mario Andrés Muñoz. Revisiting where are the hard knapsack problems? via instance space analysis. *Computers & Operations Research*, 128:105184, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0305054820303014, doi:10.1016/j.cor.2020.105184.

26 Kate Smith-Miles and Mario Andrés Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. *ACM Computing Surveys*, 55(12):1–31, December 2023. doi:10.1145/3572895.

27 Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, February 2011. doi:10.1007/s10472-011-9230-5.

28 Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Proceedings of the International Symposium on Combinatorial Search*, 10(1):151–158, 2021. doi:10.1609/socs.v10i1.18510.

29 Peter J. Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the minizinc challenge. *Constraints*, 15(3):307–316, 2010. doi:10.1007/s10601-010-9093-0.

30 Hafiz Ullah and Sultana Parveen. A literature review on inventory lot sizing problems. *Global Journal of Researches In Engineering*, 10:21–36, 01 2010.