Measures of diversity and space-filling designs for categorical data

Cedric Malherbe [1] **Emilio Domínguez-Sánchez** **Merwan Barlier** **Igor Colin** [2] **Haitham Bou-Ammar** [3 4]
Tom Diethe [1]

Abstract

Selecting a small subset of items that represent the diversity of a larger population lies at the heart of many data analysis and machine learning applications. However, when it comes to items described by categorical features, the lack of natural ordering and the combinatorial nature of the search space pose significant challenges to the current selection techniques and make existing methods ill-suited. In this paper, we propose to make a step in that direction by proposing novel methods to select subsets of diverse categorical data based on the advances in combinatorial optimization. First, we start to cast the subset selection problem through the lens of the optimization of three diversity metrics. We then provide novel bounds for this problem and present exact solvers that unfortunately come with a high computational cost. To overcome this bottleneck, we go on and show how to employ tools from linear programming and submodular optimization by introducing two computationally plausible methods that still present approximation guarantees about the diversity metrics. Finally, a numerical assessment is provided to illustrate the potential of the designs with respect to state-of-the-art methods.

1. Introduction

The diversity of data is essential in machine learning. Indeed, whenever it comes to applications where we wish to display a limited number of items to a user (Lin & Bilmes, 2011; Malherbe & Scaman, 2022) or from collecting training examples for learning algorithms (Durakovic, 2017), it is crucial to have a set of items as diverse as possible to either efficiently represents the diversity of the data or to have a rich training set that captures all the information spread out across the space of possibilities (Pronzato & Müller, 2012). When facing such problems, one can generally employ two routes by either randomly subsampling the space or by using space-filling methods (Mead, 1990; Pronzato, 2008; Joseph, 2016) such as quasi-Monte Carlo methods (Johnson et al., 1990; Niederreiter, 1992; Sobol, 1967; Faure, 1982). Nonetheless, it has long been known that random sampling provides suboptimal results for these tasks (Mead, 1990) and space-filling methods, on the other hand, rely on the strong assumption that the data live in a continuous space. Moreover, in many application domains, we often encounter categorical data such as hyperparameters in deep architecture crafting, words in natural language, atoms in molecules or job categories in demographic data that do not satisfy the continuous assumption and thus create a gap between existing techniques and practical settings. Additionally, although the diversity problem is rather fundamental, almost nothing is known from a theoretical perspective as soon as the continuity assumption does not hold. In this paper, we propose to address both these issues by providing new theoretical insights and novel subset selection methods that are guaranteed to preserve diversity for categorical data. The main difficulty of designing such methods stems from the fact that, in categorical spaces, one can only rely on the minimal information that two items are different from one another without any possible ordering, which drastically limits the possibility to adapt existing techniques that rely on ordering the data space. In this paper, we propose to tackle these challenges and show how to employ tools from both the combinatorial and greedy optimization literature to develop methods to generate diverse designs. More precisely, our contribution can be summarized as follows:

1. we formulate the problem of optimal categorical designs by using three notions of diversity in categorical spaces: the average covering, packing, and covering radii (Section 2);
2. we provide theoretical results for the construction of optimal categorical designs concerning diversity metrics (Proposition 3.1, 3.2 and 3.3) and novel exact formulation of the problems using linear programming (Proposition 3.4), which allows to generate exact designs for small spaces due to their computational cost;

[1]DS&AI, BioPharmaceuticals R&D, AstraZeneca, UK [2]LTCI, Télécom Paris [3]University College London [4]Huawei Noah's Ark Lab. Correspondence to: <cedric.malherbe@astrazeneca.com, haitham.bou-ammar@huawei.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

3. to overcome the computational bottleneck, we propose two novel approximation algorithms based on greedy schemes (GRIPPR and GAC, Section 4) to create designs that are less computationally intensive than exact solvers but still present approximation guarantees with regard to the diversity metrics (Theorems 4.3 and 4.7);

4. we provide numerical evidence showing that the methods introduced outperform standard baselines for various downstream tasks such as diversity sampling, parameter tuning and active learning (Section 5).

Related continuous works. Due to the generic nature of the problem and the capacity of these methods to outperform random selection in a wide variety of tasks (Fang et al., 2005; Santner et al., 2003), there is an abundant literature dedicated to the construction of diverse designs for continuous data (see, e.g., (Pronzato & Müller, 2012; Joseph, 2016) for specific reviews on the topic). Perhaps, the most famous methods include Latin Hypercube Sampling (McKay et al., 2000), Halton (Halton & Smith, 1964), Hammersley (Hammersley, 1960), Sobol (Sobol, 1967), Faure (Faure, 1982), Korobov and Neiderreiter (Niederreiter, 1992) sequences, minimax and maxmin designs (Johnson et al., 1990). However, most of these methods rely on specific constructions that either divide each continuous unit of the hypercube along each dimension (Kenny et al., 2000) or involve continuous sequences (Chi et al., 2005) which makes them impracticable for non-continuous data, and it is not clear whether discretizing those designs preserve their guarantees. Nonetheless, among these approaches, the closest works related to ours is the series of work of (Pronzato, 2017; Gómez et al., 2021) where they propose methods to construct designs that are optimal with regard to various statistical measures similar to the one considered in the paper. For instance, they show that they can construct optimal designs using Delaunay triangulation (Pronzato & Müller, 2012) or Voronoï tessellation which restricts their use to continuous spaces. Our contribution follows this line of works by showing that one can still construct optimal designs for data that are non-continuous by using adequate tools from both the linear programming and submodular optimization literatures. As a byproduct, we also show that their construction methods differ and the associated performance guarantees are discrete by nature. Finally, note also that some other approaches focus on information maximization using Kullback-Leibler (Jourdan & Franco, 2009), Entropy (Jones & Johnson, 2009) or Chaining (Contal et al., 2015) and create designs for an underlying system f over which we have a prior knowledge (linear, quadratic, gaussian process). Here, we follow the line of optimal designs where we do not have specific underlying systems but focus on diversity criteria, hence a different goal (covering a space instead of maximizing the knowledge of a system).

Related discrete works. On top of the continuous methods presented above, there is also a body of combinatorial literature related to our work. First, the metric k-center problem (Hochbaum & Shmoys, 1985; Lim et al., 2005), which is a theoretical combinatorial optimization problem, consists of finding a subset of k nodes in a graph for which the largest distance of a node in the graph to the subset is minimal. It has been known to be an NP-hard problem (Hochbaum, 1984) and that iteratively constructing the set of k nodes by adding the node the farthest away from the current centers (which is proved to be NP-hard in (Lanctot et al., 2003)) provides an approximation algorithm (Gonzalez, 1985). In particular, the GIPPR algorithm we propose builds upon this scheme by proposing an exact method for categorical data that relies on linear programming. Moreover, it is worth mentioning that determinantal point processes (DPPs) that were originally proposed to model repulsions of particle distributions in physics (Macchi, 1975) also propose probabilistic models that can be used to select a small diverse subset out of a large collection of categorical items (see, e.g., k-DPP in Section 5 of (Kulesza et al., 2012)). With regards to this literature, our approach is orthogonal in the sense that we approach this problem from an optimality perspective and provide results for novel criteria while DPPs provide probabilistic models. Finally, it is also worth mentioning that greedy optimization schemes similar to the one we use in the GAC algorithm have been used in a wide variety of similar tasks such as diversity in the news feed and document summarization (Malherbe & Scaman, 2022; Lin & Bilmes, 2011).

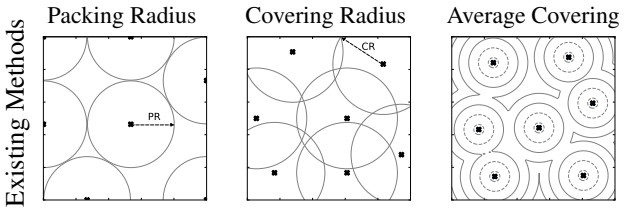
2. Categorical designs: problem statement and diversity measures

2.1. Problem statement and notations

Setup. In this paper, we focus on the problem of constructing space-filling designs for data described by a collection of categorical variables. More precisely, we consider the problem of constructing designs in the boolean hypercube, denoted here by $\mathcal{X} = \{0,1\}^d$ for any dimension $d \geq 2$. Given a set of size $n \geq 1$ as input, we aim at constructing a subset $D_n = (x_1, \dots, x_n)$ of n points in $\mathcal{X}$, called a design, that preserves the diversity of the space $\mathcal{X}$ suitably. Since the notion of diversity does not admit a unique definition, we will focus on designs solutions to the following problems:

$$\begin{array}{ll} \text{find} & D_n^* := (x_1, \dots x_n) \in \mathcal{X}^n \\ \text{such that} & \ell(D_n^*) = \min_{D_n \in \mathcal{X}^n} \ell(D_n) \end{array} \tag{1}$$

where $\ell : \mathcal{X}^n \to \mathbb{R}$ is a fixed measure of the diversity of a design D_n. To be more precise, we focus on creating designs optimizing three different diversity measures which are further defined below: the covering radius, the packing radius, and the average covering (i.e., $\ell \in \{\texttt{CR}(\cdot), -\texttt{PR}(\cdot), \texttt{AC}(\cdot)\}$).



This Paper			
	[0,0,0,0,0,0,0,0]	[0,1,1,1,0,0,1,0]	[0,1,1,1,0,0,1,0]
	[0,0,1,1,1,1,1,1]	[1,1,0,0,1,0,0,1]	[1,1,0,0,1,0,0,1]
	[1,0,0,1,0,1,0,1]	[0,0,1,0,1,0,0,0]	[0,0,1,0,1,0,0,0]
	[1,1,1,1,1,0,0,1]	[1,1,0,0,0,0,1,1]	[1,1,0,0,0,0,1,1]
	[1,1,1,1,0,1,1,0]	[0,1,0,1,0,1,0,1]	[0,1,0,1,0,1,0,1]

Figure 1. **Top:** Designs of $n = 7$ points that respectively optimize the Packing Radius, Covering Radius, and Average Covering in the continuous space $\mathcal{X} = [0, 1]^2$. The PR and CR radii are plotted in grey as well as the level sets of the function $x \mapsto d(x, D_n)$. **Bottom:** Designs of $n = 5$ points provided in this paper which optimize the same diversity metrics when $\mathcal{X}_8 = \{0, 1\}^8$ is a categorical space.

As an example, three designs satisfying these optimality conditions are displayed in Figure 1. Lastly, we point out that the methods presented in the paper can also be extended to non-binary variables (see Appendix D).

Notations. In the remainder of the paper, we denote by $\mathbb{I}\{\cdot\}$ the standard indicator function taking values in $\{0, 1\}$. For any $x = (x_1, \dots, x_d) \in \mathcal{X}$ and $x' = (x'_1, \dots, x'_d) \in \mathcal{X}$, we denote the vectorized XOR operator as follows $x \oplus x' = (x_1 \oplus x'_1, \dots, x_d \oplus x'_d)$ where $x_i \oplus x'_i = \mathbb{I}\{x_i \neq x'_i\}$ for all $i \in \{1, \dots, d\}$. With a slight abuse of notation, we denote by $\|x\|_1 = \sum_{i=1}^d x_i$ the standard ℓ_1-norm. For any finite set V, its number of elements is denoted by $|V|$, and its power set by 2^V. For any set function $F : 2^V \to \mathbb{R}$, we denote by $\arg\max_{x \in A} F(x) = \{x \in 2^V : F(x) \geq F(x') \ \forall x' \in 2^V \backslash x\}$ the set of points for which the function is maximal. A set function $F : 2^V \to \mathbb{R}$ is said to be submodular if for every $A \subseteq B \subseteq V$ and $e \in V/B$ it holds that $F(A \cup \{e\}) - F(A) \geq F(B \cup \{e\}) - F(B)$. Conversely, F is said to be supermodular if $-F$ is submodular. We denote by $X \sim \mathcal{U}(V)$ the standard uniform distribution over the set V (i.e. $\mathbb{P}(X = e) = 1/|V|$ for all $e \in V$). For any random variable X, we denote by $\mathbb{E}[X]$ its expectation. The Hamming ball of radius $r \geq 0$ around any point $x \in \mathcal{X}$ is denoted by $B_H(x, r) = \{x' \in \mathcal{X} : d_H(x, x') \leq r\}$ where $d_H(\cdot, \cdot)$ denotes the Hamming distance defined below. We denote by $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ the root and floor functions. Finally, the distance of a point $x \in \mathcal{X}$ to any subset $V \subseteq \mathcal{X}$ is defined by $d(x, V) = \min_{e \in V} d_H(x, e)$.

2.2. Diversity measures

To specify the three notions of diversity considered in the paper, we start with the following notion of distance that naturally fits categorical spaces.

Definition 2.1. (Hamming Distance). The Hamming distance between two points $x = (x_1, \dots, x_d) \in \mathcal{X}$ and $x' = (x'_1, \dots, x'_d) \in \mathcal{X}$ is defined as the number of coordinates in which x and x' differ. Formally, it is defined as follows: $d_H(x, x') = \sum_{i=1}^d \mathbb{I}\{x_i \neq x'_i\}$.

It is well known that the Hamming distance is positive define and satisfies the triangular inequality. It plays the role of natural distance on categorical spaces since here we can only compare if two items are different. Equipped with this distance, we can now introduce the notions of diversity.

Definition 2.2. (Packing Radius). Let $D_n = (x_1, \dots, x_n) \in \mathcal{X}^n$ be any design of $n \geq 2$ points of the categorical space. Then, we define the packing radius of the design D_n as follows:

$$\texttt{PR}(D_n) := \min_{x_i \neq x_j \in D_n} \frac{d_H(x_i, x_j)}{2}.$$

Having a packing radius as large as possible ensures that the points of the design are as different as possible. It is thus a desirable characteristic. For instance, it can be motivated by setting up a series of n elements as diverse as possible so that it minimizes the chance to take one element for another one. Designs maximizing this criterion are also sometimes called maximin designs (Johnson et al., 1990).

Definition 2.3. (Covering Radius). Let $D_n = (x_1, \dots, x_n)$ be any design of $n \geq 1$ points of the categorical space. Then, we define the covering radius of the design D_n over $\mathcal{X}$ as follows:

$$\texttt{CR}(D_n) := \max_{x \in \mathcal{X}} d_H(x, D_n) = \max_{x \in \mathcal{X}} \min_{i=1\dots n} d_H(x, x_i).$$

This criterion computes the distance of the point of the domain which is the farthest away from the design. Consequently, having a small covering radius ensures that *any* point of the full space has a similar point in the design. Hence, having a CR as small as possible is desirable. It is sometimes called dispersion in the theory of quasi-Monte Carlo methods (Niederreiter, 1992) or minimax-distance (Johnson et al., 1990).

Definition 2.4. (Average Covering). The average covering of a design $D_n = (x_1, \dots, x_n)$ of $n \geq 1$ points of the boolean hypercube is defined as follows:

$$\texttt{AC}(D_n) := \mathbb{E}\left[d_H(X, D_n)\right] = \mathbb{E}\left[\min_{i=1\dots n} d_H(X, x_i)\right].$$

where $X \sim \mathcal{U}(\mathcal{X})$ is uniformly distributed over the space.

This criterion is perhaps the most intuitive and computes the average distance of a point taken at random in the space. Having an average covering as small as possible ensures that, on average, all the points of the space are effectively represented by the design. Our goal in this paper is to create categorical designs that are optimal according to the above criteria.

3. Optimal categorical designs: some properties and exact construction

In this section, we present the main theoretical properties of optimal designs solving (1) and present two simple and exact methods to construct such designs.

3.1. Properties of optimal designs

We present here three important results that explain the generic behavior of optimal designs. We point out that additional results as well as the proofs of the results can be found in Appendix B. To our knowledge, these results are new to the literature.

Proposition 3.1. *Consider any diversity measure $\ell \in \{\texttt{CR}(\cdot), \texttt{PR}(\cdot), \texttt{AC}(\cdot)\}$ and let D_n^* be an optimal design of any size $n \geq 2$ solving (1) with regards to the diversity metric. Then, the function $n \mapsto \ell(D_n^*)$ is non-increasing.*

Informally, this result states that any of the optimal diversity metrics can only decrease or stagnate the more points n are added to the design. Of course, it is also important to note that to have a perfect covering (e.g., $\ell(D_n) = 0$ for $\texttt{CR}$), we necessarily have to include all the points of the space (i.e., $n = 2^d$). Thus, to have a finer understating of the decreasing rates of the diversity metrics towards 0 we can expect with regard to the size of the design n, we cast the next result.

Proposition 3.2. (Bounds on optimal diversity metrics). *Consider any diversity measure $\ell \in \{\texttt{CR}(\cdot), \texttt{PR}(\cdot)\}$, let D_n^* be any optimal design of any size $n \geq 4$ solving (1) with regards to the diversity metric and consider for simplicity that d is odd. Then, the following inequalities hold:*

$$\left\lfloor \frac{d}{2} \right\rfloor - \lceil \ln_2(n) \rceil \leq \ell(D_n^*) \leq \left\lfloor \frac{d}{2} \right\rfloor - \frac{\ln_2(n-1)}{2}.$$

Hence, we deduce for instance that the decreasing rate of the optimal covering radius is of order $\texttt{CR}(D_n^*) = \lfloor d/2 \rfloor - \Theta(\ln_2(n))$. As a consequence, since we have a logarithmic decreasing rate in terms of the size n, this result highlights the fact that the marginal impact of adding a novel point to the design will decrease as the size n grows. The curves displayed in Figure 2 in Section 5 highlight this phenomenon in practice. Additionally, it is also interesting to compare this result to the case of continuous designs (Gómez et al., 2021) where it is known that $\texttt{CR}(D_n^*) = \Theta(n^{-1/d})$ when $\mathcal{X} = [0,1]^d$. Here, the dimension d impacts the initial value $\lfloor d/2 \rfloor$ in categorical spaces and not the decreasing rate as opposed to continuous spaces where it directly impacts the rate of decay. Note also that a similar upper bound for the average covering can be found in B. The next result states that optimal designs are invariant under permutation of the index and stable by addition of any point.

Proposition 3.3. (Invariance of design measures). *Consider any diversity measure $\ell \in \{\texttt{CR}(\cdot), \texttt{PR}(\cdot), \texttt{AC}(\cdot)\}$ and let D_n^* be any optimal design solving (1). Then, for any permutation $\pi : \{1, \dots, d\} \to \{1, \dots, d\}$ of the dimensions of the design and any point $z \in \mathcal{X}$, we have that: $\ell(\pi(D_n^*) \oplus z) = \ell(D_n^*)$ where $\pi(D_n^*) = (\pi(x_1), \dots, \pi(x_n))$ with $\pi(x_i) = (x_i^{\pi(1)}, \dots, x_i^{\pi(d)})$.*

Thus, we deduce that categorical designs are—as opposed to continuous designs—also invariant by addition. Moreover, since taking any permutation and adding a point to an optimal design also provides an optimal design, we deduce that there is a non-unicity of optimal designs which limits the opportunity to find a closed form. On the other hand, this result states that once we find a suitable design D_n, one can easily create other designs with the same properties by simply applying permutations (as we use to generate designs in practical settings in Section 5).

3.2. Exact computation of optimal designs

Exhaustive-search. Perhaps, the most straightforward way to create an optimal design with regard to a given measure is to perform an exhaustive search over all possible designs of the desired size. More formally, given a size $n \geq 1$ and a metric ℓ, an exhaustive search can be described as follows: (1) compute $\ell(D_n)$ for any design $D_n \in \mathcal{X}^n$ of size n; (2) return $D_n^* \in \arg\max_{D_n \in \mathcal{X}^n} \ell(D_n)$. By enumerating all possible solutions, we will of course identify an optimal design. However, the main drawback of this method is its numerical complexity. Indeed, since there are $\binom{|\mathcal{X}|}{n}$ possible designs and the complexity of computing a diversity metric is at least of order $n^2 d$ (see Appendix A), the overall complexity of an exhaustive search is at least of order $\Omega(\binom{|\mathcal{X}|}{n} n^2 d)$, which makes it almost impracticable even for small designs. The main computational bottleneck stems from the combinatorial factor $\binom{|\mathcal{X}|}{n}$. We show how to improve it below.

Linear programming. Interestingly, one possible way to improve the numerical complexity of identifying an optimal design in practice is to notice that those problems can be put in standard linear forms. Indeed, since optimal designs can be formulated as a maximization problem as in (1), one can employ tools from linear programming to speed up the computation. More precisely, the next result shows that one can find an optimal design by solving a linear program.

Proposition 3.4. (LP-exact solvers). *Consider any diversity measure $\ell \in \{\texttt{CR}(\cdot), \texttt{PR}(\cdot), \texttt{AC}(\cdot)\}$ and any design size $n \geq 2$. Then, identifying an optimal design D_n^* solving (1) can be done by solving a standard integer program of the form:*

$$\begin{array}{ll} \textit{maximize} & \mathbf{c}^T \mathbf{x} \\ \textit{subject to} & A\mathbf{x} \leq b \end{array}$$

where $\mathtt{x} \in \mathbb{Z}^{n_v}$, $c \in \mathbb{R}^{n_v}$, $A \in \mathbb{R}^{n_c \times n_v}$ *and* $b \in \mathbb{R}^{n_c}$ *are fully detailed in Appendix B due to their size and with a number of constraints* n_c *and variables* n_v *of order* $O(dn^2)$ *for the packing radius and* $O(n|\mathcal{X}|)$ *for the covering radius and average covering.*

In practice, it is thus possible to employ standard integer linear programming solvers (Cplex, 2009; Achterberg, 2009) which rely on relaxing the integrity constraints to drastically prune the search space and speed up the computations. Of course, using a linear programming formulation does not improve the theoretical complexity in a worst-case scenario. However, in practice, it only allows to find optimal designs of small sizes (i.e. $n < 6$ as shown in Appendix D). To the best of our knowledge, we are not aware of any other algorithms that can create exact optimal designs with a better ptractical complexity. In the next section, we will provide two approximate algorithms that have a much lower computational cost and allow to create designs for larger sizes.

4. Approximate algorithms for the construction of categorical designs

In this section, we introduce two algorithms (GIPPR and GAC) which allow to construct categorical designs that still present approximation guarantees with regard to optimal design solving (1) while drastically reducing the computational cost of exact solvers.

4.1. The GIPPR algorithm with approximation guarantees for packing and covering radii

Algorithm statement. The GIPPR algorithm (Algorithm 1) creates a design D_n over the combinatorial input space by iteratively adding points $x_{t+1} \in \mathcal{X}$ to the design D_t at any time step $t \geq 1$ (lines 6 & 7). It only takes as input the dimensionality d of the space and the size n of the desired design. It starts with a design D_1 filled by a random point (line 1). Then, at each iteration $t \geq 1$, it solves an integer program where the values of the constraints depend on the points D_t gathered so far. To have a better understanding of GIPPR, the next proposition sheds some light on the linear program.

Proposition 4.1. *Let* $D_t = (x_1, \ldots, x_t)$ *be the set of points generated by GIPPR at time* $t \geq 1$ *with* $x_i = (x_i^1, \ldots, x_i^d)$ *for all* $i \in \{1, \ldots, t\}$. *Then, we have the following equivalence:*

$$\arg\max_{\substack{[1,d] \\ A\mathtt{x} \leq b}} \mathbf{c}^T \mathtt{x} = \arg\max_{x \in \mathcal{X}} d(x, D_t)$$

where $\mathtt{x} \in \{0,1\}^d \times \{0, \ldots, d\}$, $\mathbf{c} = (0, \ldots, 0, 1) \in \mathbb{R}^{d+1}$, $b = (\|x_1\|_1, \ldots, \|x_t\|_1) \in \mathbb{R}^{t \times 1}$ *and* $A \in \mathbb{R}^{t \times (d+1)}$ *with* $a_{i,j} = (2x_i^j - 1)$ *for all* $(i,j) \in \{1, \ldots, t\} \times \{1, \ldots, d\}$ *else 1.*

Algorithm 1 Greedy Integer Programming Packing Radius (GIPPR)

Require: Dimensionality $d \geq 1$ of the categorical space, size $n \geq 2$ of the design

1: Set randomly the first design point $D_1 \leftarrow \{x_1\}$ where $x_1 \sim \mathcal{U}(\mathcal{X})$
2: Set $\mathbf{c} = (0, 0, \ldots, 0, 1) \in \mathbb{R}^{d+1}$
3: **for** $t = 1, ..., n-1$ **do**
4: Set $b = (\|x_1\|_1, \ldots, \|x_t\|_1) \in \mathbb{R}^t$ and $A \in \mathbb{R}^{t \times d+1}$ as described in Proposition 4.1
5: Solve the following integer program with $\mathtt{x} \in \{0,1\}^d \times \{0, \ldots, d\}$ and store any of the results:

$$(\mathtt{x}_1^*, \ldots, \mathtt{x}_{d+1}^*) \in \arg\max_{A\mathtt{x} \leq b} \mathbf{c}^T \mathtt{x}$$

6: Set the novel design point: $x_{t+1} \leftarrow (\mathtt{x}_1^*, \ldots, \mathtt{x}_d^*)$
7: Add the point to the design: $D_{t+1} \leftarrow D_t \cup \{x_{t+1}\}$
8: **end for**
9: **Return** the design D_n

Thus, since the above equality is a set equality, we deduce that GIPPR selects in fact any point $x_{t+1} \in \arg\max_{x \in \mathcal{X}} d(x, D_t)$ which the farthest away from the current design D_t in lines 5 & 6. It is noteworthy that this generic mechanism of selecting the furthest away point can alternatively be found under the name coffee-house design in the continuous and graph domains (Müller, 2001; 2007). Here, the key differences are the following: (1) while there is not an exact way to solve the furthest away problem in the continuous domain, here we can solve it exactly using linear programming tools and (2) we do not rely on any discretization of the space here and only requires solving an integer program with $d+1$ variables and n constraints.

Theoretical analysis. Since GIPPR sequentially adds the points which are the furthest away from the collected points, it is not obvious how it relates to the optimization of the diversity metrics of Section 2. However, the next result shows that the algorithm implements a greedy scheme with regard to the packing radius. To our knowledge, it is the first time these results are stated in the literature.

Lemma 4.2. (Greediness of GIPPR). *Let* $D_t = (x_1, \ldots, x_t)$ *be the sequence of points generated by GIPPR at any time* $t \geq 1$ *and let* x_{t+1} *denotes the next point of the design defined in line 6. Then, we have the following property:* $x_{t+1} \in \arg\max_{x \in \mathcal{X}} d_H(x, D_t) = \arg\max_{x \in \mathcal{X}} \mathtt{PR}(D_t \cup \{x\})$.

Thus, we deduce that by solving the integer program of Proposition 4.1, GIPPR selects at each time step a point $x_{t+1} \in \arg\max_{x \in \mathcal{X}} \mathtt{PR}(D_t \cup \{x\})$ maximizing the packing radius. The generic strategies of taking the locally optimal choice at each stage are often referred to as greedy

or myopic in the literature (Bednorz & Sciyo.com, 2008; Wilson et al., 2018), hence the name GIPPR. From this perspective, it could be tempting to exploit results from the submodular optimization literature (Nemhauser et al., 1978) to explain the convergence of GIPPR. Unfortunately, the $\texttt{PR}$ does not satisfy the submodular property (see Appendix C.1). Nonetheless, it is still possible to show that GIPPR still achieves an approximation guarantee with regard to optimal diversity metrics by using the covering arguments of (Gonzalez, 1985), as shown below.

Theorem 4.3. (Approximation guarantees). *Let $D_n = (x_1, \dots, x_n)$ denotes the design provided by GIPPR (Algorithm 2) over the categorical space $\mathcal{X}$ and denote by $\texttt{PR}_n^*$ and $\texttt{CR}_n^*$ the values of the optimal packing and covering radii of any size $n \geq 2$. Then, we have the following guarantees:*

$$\texttt{CR}_n^* \leq \texttt{CR}(D_n) \leq 2 \cdot \texttt{CR}_n^*$$

and

$$\frac{1}{2} \cdot \texttt{PR}_n^* \leq \texttt{PR}(D_n) \leq \texttt{PR}_n^*.$$

Thus, the design provided by GIPPR will be at least half as good in terms of packing radius as the optimal design solving (1). More surprisingly, it also shows that the greedy policy on the packing radius, also presents similar guarantees for the covering radius. Here, it is important to note that we have a trade-off between accuracy and numerical complexity when compared to the exact solvers of Section 3. Indeed, at the price of slightly worst theoretical guarantees (ratio 1/2), we have an algorithm that allows to create designs for larger sizes n thanks to a drastically better numerical complexity (see Appendix D for numerical examples).

Remark 4.4. **(Numerical complexity).** At each iteration, we solve one linear program with $d+1$ variables and n constraints. Thus, the total complexity of the algorithm is of order $O(n \times LP(d+1, n))$ while the complexity of the exact solver using linear programming is of order $O(LP(dn^2, dn^2))$. To have a better grasp on the numerical improvement, when solving line 5 with a brute-force solution (Appendix A), the complexity of GIPPR is of order $O(|\mathcal{X}|n^2 d)$ while the complexity of exact solvers of Section 3 is of order of $O(\binom{|\mathcal{X}|}{n} n^2 d)$.

Remark 4.5. **(Non-binary variables).** We point out that GIPPR can be extended to the case of non-binary input spaces $\mathcal{X} = \{0, \dots, n_1\} \times \cdots \times \{0, \dots, n_d\}$ by simply encoding each non-binary variables with several binary variables, as detailed in Appendix D.2.

4.2. The GAC algorithm with approximation guarantees for the average covering

Algorithm statement. Similarly to the previous algorithm, GAC (Algorithm 2) creates a design D_n over the categorical space $\mathcal{X}$ by iteratively adding a point x_{t+1} to the previously constructed design D_t at each time step (line 4). It implements a greedy optimization scheme (Nemhauser et al., 1978) over the average covering. It only takes as input the dimensionality d of the input space and the size of the desired design n. The mechanism behind GAC is simple. At each iteration, it adds any point $x_{t+1} \in \arg\min_{x \in \mathcal{X}} \mathbb{E}_X[d_H(X, D_u \cup \{x\})] = \texttt{AC}(D_t \cup \{x\})$ greedily maximizing the average covering. In this sense, it is similar to GIPPR, but here we greedily maximize the average covering.

Algorithm 2 Greedy Average Covering (GAC)

Require: Dimensionality $d \geq 1$ of the categorical space, number $n \geq 2$ of design points

1: Set the first design point at random $D_1 \leftarrow \{x_1\}$ where $x_1 \sim \mathcal{U}(\mathcal{X})$
2: **for** $t = 1, ..., n-1$ **do**
3: Get any point that greedily minimizes the average covering:

$$x_{t+1} \in \arg\min_{x \in \mathcal{X}} \mathbb{E}_{X \sim \mathcal{U}(\mathcal{X})}[d_H(X, D_t \cup \{x\})]$$

4: Add the point to the design: $D_{t+1} \leftarrow D_t \cup \{x_{t+1}\}$
5: **end for**
6: **Return** the design D_n

Theoretical analysis. To investigate the properties of GAC, we first cast a generic result that highlights an important property of the average covering measure. To the best of our knowledge, these results are new to the literature.

Proposition 4.6. (Average covering is supermodular). *Consider the set function $D \mapsto \texttt{AC}(D)$ where D is any non-empty design of $\mathcal{X}$. Then, the function is non-increasing and supermodular:*

$$|\texttt{AC}(D_{t+i} \cup \{x\}) - \texttt{AC}(D_{t+i})| \leq |\texttt{AC}(D_t \cup \{x\}) - \texttt{AC}(D_t)|$$

for all $x \in \mathcal{X}, D_t \subseteq D_{t+i} \in \mathcal{X}^{t+i}, i, t \geq 0$.

Hence, the average covering satisfies the diminishing return property in the sense that the marginal gain $|\texttt{AC}(D_t \cup \{x\}) - \texttt{AC}(D_t)|$ of adding an element $x \in \mathcal{X}$ will decrease as the design D_t gets larger. In other words, the more we add points to the current design, the more marginal will be the improvements. A crucial consequence of this proposition is that we can directly invoke arguments from the submodular optimization literature (Nemhauser et al., 1978) to obtain the following convergence result.

Theorem 4.7. (Approximation guarantees). *Let $D_n = (x_1, \dots, x_n) \in \mathcal{X}^n$ denotes a design of size $n \geq 1$ returned by GAC and denote by $\texttt{AC}_n^* = \min_{D_n \in \mathcal{X}^n} \texttt{AC}(D_n)$ the best average covering that can be achieved and $\texttt{AC}_1^* = \texttt{AC}(D_1) = d/2$. Then, we have the following approximation*

guarantee:

$$\frac{\mathtt{AC}_1^* - \mathtt{AC}_n^*}{2} \leq \mathtt{AC}_1^* - \mathtt{AC}(D_n) \leq \mathtt{AC}_1^* - \mathtt{AC}_n^*.$$

Thus, the spread between the initial covering with one point and the covering provided by GAC is at least half as good as the optimal one. Similarly to GIPPR, the main message here is that we observe a trade-off between accuracy and numerical complexity with regard to exact solvers. We can now create designs with larger values of n thanks to the gain on the numerical complexity described below.

Remark 4.8. **(Numerical complexity).** At each time step, we simply need to identify a point $x_{t+1} \in \arg\min_{x \in \mathcal{X}} \mathtt{AC}(D_t \cup \{x\})$. By using a naive implementation provided in Appendix C using $\mathtt{AC}(D_n) = (1/|\mathcal{X}|) \sum_{x \in \mathcal{X}} d(x, D_n)$, the complexity of the algorithm is at most of order $O(n|\mathcal{X}|^2)$, which has to be compared with the complexity of order $O(\binom{|\mathcal{X}|}{n} nd2^d)$ of the exact solver. Again, with a lower computational cost, one can achieve an approximation of the true result.

Remark 4.9. **(Practical implementation).** Since the practical optimization of submodular functions is a dense subject of research by itself, we point out that there are several ways to provide more efficient implementations of GAC depending on the available computational hardware if necessary. In particular, due to 1) the expectation in the average covering, it is possible to use a faster stochastic approximations (Proposition A.2), 2) the incremental nature of the function, it is possible to use the iterative update $d_H(x, D_{t+1}) = \min(d_H(x, x_{t+1}), d_H(x, D_t))$ to drastically reduce the computation as shown in (Gómez et al., 2021) and 3) the submodularity of the average covering allows to use both lazy (Minoux, 2005) and stochastic evaluations (Mirzasoleiman et al., 2015). Details regarding efficient implementations can be found in Section D.5.

5. Numerical experiments

In this section, we empirically demonstrate the advantages of GIPPR and GAC compared to the baselines 1) random designs, 2) discretized continuous design (d-Halton (Halton & Smith, 1964)) and 3) k-DPP designs (Kulesza et al., 2012). The full details of the experiments as well as additional results for non-binary spaces can be found in Appendix D as well as the runing times of all the algorithms.

Diversity measures. Finally, in many applications such as selecting a subset of items from a catalog or news feed (Malherbe & Scaman, 2022), the end goal is to build a subset as diverse as possible. Thus, to measure the overall capacity of the methods to provide diverse subsets, we directly recorded the values of the diversity metrics $\mathtt{PR}, \mathtt{CR}$ and $\mathtt{AC}$ for different sizes n in Figure 2. Again, Random designs are always outperformed by any other design. More importantly, although GAC and GIPPR are approximate methods, they still respectively consistently outperform other methods over the $\mathtt{PR}$ and $\mathtt{AC}$ measures. For $\mathtt{PR}$, either GAC or GIPPR achieve the best results, making them the overall best methods for diversity metrics. Finally, is also interesting to note that for most methods we indeed observe a logarithmic decay over the measures as suggested by the theoretical results of Section 3. Lastly, we can see that GIPPR (resp. GAC) almost provides empirically better results for PR (resp. AC). However, since they are approximate solvers (and it is generally hard to predict which method will give the best results on a downstream task) we generally advice that you should go with GAC if you want to have a good "average" and if you want to have a better "worst-case" you should go to GIPR.

Numerical integration. Next, similarly to continuous designs that outperform random methods for numerical integration (Morokoff & Caflisch, 1995), we investigated the capacity of our designs to perform integration in discrete spaces. We computed the empirical mean $\widehat{F}_n(D_n) = \sum_{x \in D_n} f(x)/n$ using the diverse subsets D_n over two functions f commonly encountered in the combinatorial litera-

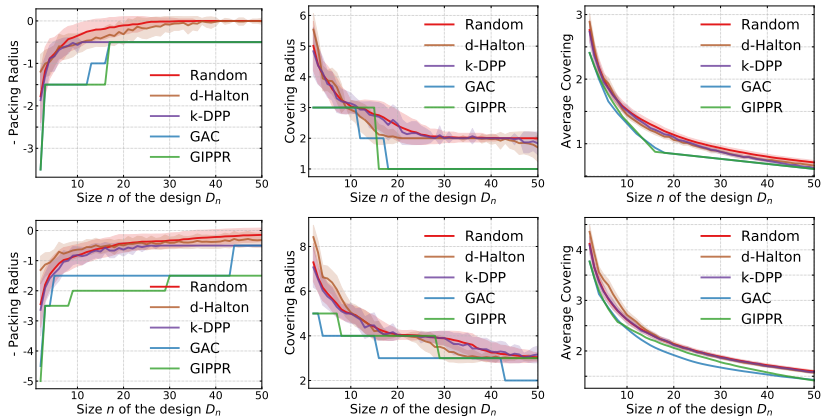


Figure 2. The graph displays the evolution of the diversity measures $\mathtt{PR}(D_n)$, $\mathtt{CR}(D_n)$ and $\mathtt{AC}(D_n)$ for different design sizes $n \in \{2, \dots, 50\}$. The bold line represents the average value and, when available, the transparent area represents the standard deviation computed over 100 runs. The top line considers the case when $d = 7$ and the bottom line considers the case when $d = 8$. For each of the plots, lower is better.

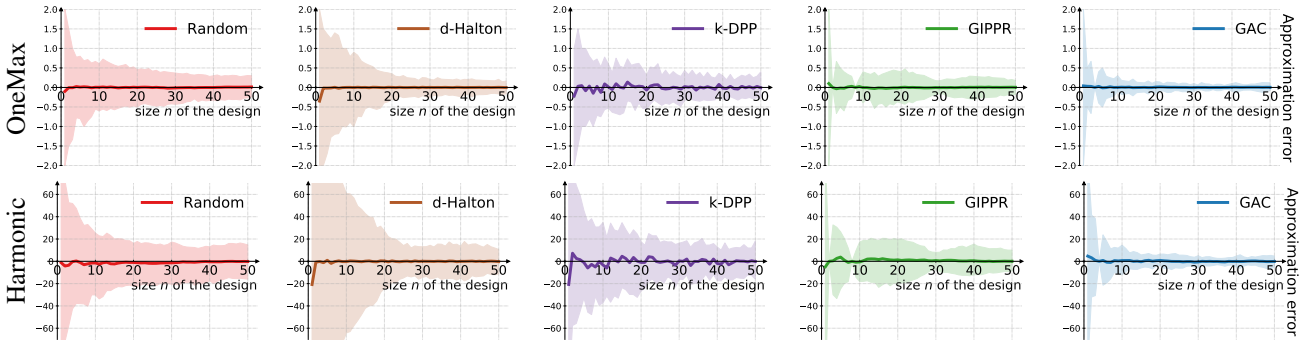


Figure 3. The graphs display the average approximation error $\widehat{F}_n(D_n) - \mathbb{E}_{X\sim\mathcal{U}(\mathcal{X})}[f(X)]$ in bold for various design sizes $n \in \{1,\dots,50\}$, and the transparent colors represent the 90% and 10% quantile of the error computed over 100 runs with $d = 10$ for OneMax (Top) and Harmonic (Bottom).

ture (Doerr et al., 2019): OneMax $f(x) = \sum_{i=1}^{d} \mathbb{I}\{x_i = 1\}$ which equally captures the impact of each dimension and the weighted Harmonic $f(x) = \sum_{i=1}^{d} i^2\mathbb{I}\{x_i = 1\}$. The difference between the approximate $\widehat{F}_n(D_n)$ and the ground truth mean $\sum_{x\in\mathcal{X}} f(x)/n$ are averaged and collected in Figure 3 over 100 runs. First, note that all the methods evenly cover the space in the sense that they are unbiased and centered around the true average for both functions. Second, observe that discretizing a continuous design (d-Halton), which reduces the variance in continuous spaces, can in fact generate estimates with larger tails than random designs for categories. Last, note that the variance of the estimates we propose is much lower than random methods for which it can be up to five times smaller. Thus, similarly to the continuous case, optimal designs also provide better estimates than random methods in the discrete case.

Data collection. First, we evaluated the capacity of the methods to build and collect efficient datasets for any regression problem. We considered the generic problem of selecting a set of n samples $D_n = (x_1,\dots,x_n) \in \mathcal{X}^n$ in order to collect a (costly) training set of labelled data $((x_1, f(x_1)),\dots,(x_n, f(x_n)))$ to then build an approximation of any function $f : \mathcal{X} \to \mathbb{R}$ described over categories. We used three real-world systems f taken from the categorical benchmark of (Doerr et al., 2019): (1) LAB from telecommunication which measures the autocorrelation of a sequence of bits (2) MIS from combinatorics which consists of estimating the size of the largest independent set in a graph and (3) Ising Model from physics which measures the configuration's energy of the Ising spin glass model with d spins directing up or down. All the details can be found in Appendix D. For each design D_n, we labeled the corresponding dataset with a budget of $n = 20$ data and $d = 10$ parameters. Then, we learned a gaussian process model $\hat{f}_{\theta_n}$ by maximizing the marginal log-likelihood over the labeled data and evaluated its generalization capacities by computing its mean square error (MS) and log-likelihood (LL) over the whole space $\mathcal{X}$ in Figure 4. First, observe that Random (baseline) designs are always outperformed by any other design. More importantly, GAC and GIPPR respectively consistently outperform other methods such as DPP and discretized continuous designs, making them the overall best methods for data collection and we deduce that when facing such problems it is better to use approximates of optimal designs than random methods.

	Random	d-Halton	k-DPP	GRIPPR	GAC
Ising-MS	6.33 (±0.9)	6.14 (±0.9)	6.34 (±0.8)	6.13 (±0.8)	**5.77 (±0.7)**
Ising-LL	3.88 (±0.9)	3.53 (±0.8)	3.75 (±0.7)	3.48 (±0.6)	**3.42 (±0.6)**
MIS-MS	63.4 (±12)	65.4 (±11)	65.0 (±13)	58.5 (±13)	**47.2 (±7.8)**
MIS-LL	295 (±98)	336 (±105)	314 (±121)	235 (±111)	**163 (±56)**
LAB-MS	3.63 (±0.3)	3.58 (±0.3)	3.60 (±0.3)	**3.47 (±0.3)**	3.58 (±0.2)
LAB-LL	1.12 (±0.3)	1.09 (±0.2)	1.10 (±0.3)	**1.03 (±0.2)**	1.08 (±0.2)

Figure 4. Results of the data collection experiments (± std). Best results are highlighted in bold. Lower is better.

Hyperparameter tuning. Then, we investigated whether evaluating diverse subsets can improve upon the standard random searches to select hyperparameters. To do so, we considered the problem of tuning (1) the architecture of a classification graph neural network over the standard Cora (Cora) and CiteSeer (Cite) datasets (Sen et al., 2008) involving nine discrete decisions such as the choice of activation function, presence/absence of skip-connections and presence/absence of drop-out at each layer and (2) the seven catgeorical hyperparameters of the SCIP linear solver (Achterberg, 2009) taken from the NeurIPS 2011 competition (Gasse et al., 2022) such as separating/maxcuts and branching/scorefac over the item placement (Item) and load balancing tasks (Load). All the details are provided in Appendix D. For each problem, we evaluated $n = 20$ different sets of parameters provided by each design and kept the best configuration. The average improvements (%) over the default configuration are collected in Figure 5. Again, similarly to the above experiment, we observe that the design we propose perform better than other methods on this task.

6. Discussion and Conclusion

In this paper, we introduced two novel methods to create experimental designs for discrete variables that rely on

	Random	d-Halton	k-DPP	GRIPPR	GAC
Cora	75.5 (±2.9)	74.9 (±4.2)	76.1 (±4.1)	77.2 (±3.3)	**78.2 (±4.2)**
Cite	82.0 (±6.2)	80.7 (±5.7)	83.0 (±4.9)	83.2 (±2.5)	**84.0 (±4.2)**
Item	23.4 (±2.3)	37.4 (±1.9)	26.3 (±8.3)	40.3 (±4.8)	**41.7 (±0.5)**
Load	22.0 (±2.7)	18.0 (±2.1)	17.2 (±15)	**25.0 (±4.6)**	19.6 (±2.4)
Anon	46.8 (±5.2)	42.2 (±7.3)	47.7 (±5.3)	**63.3 (±3.2)**	58.2 (±5.7)

Figure 5. Average improvements over the default configuration (%±std). Higher is better. Best results are highlighted in bold.

greedy and linear optimization. They present both strong approximation guarantees and are shown to outperform existing methods on empirical benchmarks. Our methods have been implemented up to dimension fifty using open-source solvers allowing us to cover a new variety of problems as shown in Appendix D. In the future, we plan to (1) investigate the use of deep-learning to solve more efficiently these specific LP problems (Khalil et al., 2017) to make these methods adapted to even larger dimensional systems, (2) considering a mix of the diversity measures, and (3) extending the approach to mixed continuous/discrete inputs.

Impact Statement

In our work, we proposed a novel methodology to generate a subset of items that are described by categorical features. These new solvers are mostly agnostic to the specific application and can be applied to a wide range of problems (from experimental physics to news feed creation). Therefore, the societal and ethical impacts of our contribution are heavily dependent on the nature of the problems solved with the algorithm. We start by noting that beneficial applications of our work are thick on the ground, ranging from the design of a more efficient diversity in display systems to the design of low correlation sequences for telecommunication, or the design of methods to efficiently gather training data. For instance, in the case where we wish to display a subset of some population like in dating apps, our methods allow us to have a better subset that represents the diversity of the whole population compared to random methods. In particular, since our algorithm can create designs that minimize the covering radius or average covering, we are sure that using these methods we have the designs that represent the whole population in the best possible way and nobody will be left behind. Thus, it is easy to see the beneficial impacts in terms of inclusion, and building less biased recommendation systems. Another by-product of our methodology is perhaps to reduce the training time of big systems by only using subsets of the data. Thus, it would help to reduce the overall computational cost of any energy glutton systems and thus be beneficial from this side. From the latter example we can perceive that, even though the collection of good labeled data makes them more accurate or efficient to accomplish their tasks, the positive or negative impact of our work fully depends on the nature of the tasks considered. So, providing novel diversity methods could lead to a more beneficial models, as it could allow the design of malicious systems. However, we hope that our contribution alone will not in itself encourage individuals to design new malicious models and we believe that in the long run the impact of this work will be positive.

References

Achterberg, T. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

Back, T. and Khuri, S. An evolutionary heuristic for the maximum independent set problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 531–535. IEEE, 1994.

Bednorz, W. and Sciyo.com. *Greedy Algorithms*. Sciyo.com, 2008. ISBN 9789537619275. URL https://books.google.de/books?id=l0QezQEACAAJ.

Calandriello, D., Derezinski, M., and Valko, M. Sampling from a k-dpp without looking at all items. *Advances in Neural Information Processing Systems*, 33:6889–6899, 2020.

Chi, H., Mascagni, M., and Warnock, T. On the optimal halton sequence. *Mathematics and computers in simulation*, 70(1):9–21, 2005.

Contal, E., Malherbe, C., and Vayatis, N. Optimization for gaussian processes via chaining. *arXiv preprint arXiv:1510.05576*, 2015.

Cplex, I. I. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., and Bäck, T. Benchmarking discrete optimization heuristics with iohprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1798–1806, 2019.

Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., and Bäck, T. Benchmarking discrete optimization heuristics with iohprofiler. *Applied Soft Computing*, 88:106027, 2020.

Durakovic, B. Design of experiments application, concepts, examples: State of the art. *Periodicals of Engineering and Natural Sciences*, 5(3), 2017.

Fang, K.-T., Li, R., and Sudjianto, A. *Design and modeling for computer experiments*. Chapman and Hall/CRC, 2005.

Faure, H. Discrépance de suites associées à un système de numération (en dimension s). *Acta arithmetica*, 41(4): 337–351, 1982.

Gasse, M., Bowly, S., Cappart, Q., Charfreitag, J., Charlin, L., Chételat, D., Chmiela, A., Dumouchelle, J., Gleixner, A., Kazachkov, A. M., et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 220–231. PMLR, 2022.

Gautier, G., Polito, G., Bardenet, R., and Valko, M. DPPy: DPP Sampling with Python. *Journal of Machine Learning Research - Machine Learning Open Source Software (JMLR-MLOSS)*, 2019. URL http://jmlr.org/papers/v20/19-179.html. Code at http://github.com/guilgautier/DPPy/.

Gómez, A. N., Pronzato, L., and Rendas, M.-J. Incremental space-filling design based on coverings and spacings: improving upon low discrepancy sequences. *Journal of Statistical Theory and Practice*, 15(4):1–30, 2021.

Gonzalez, T. F. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38: 293–306, 1985.

Halton, J. and Smith, G. Radical inverse quasi-random point sequence, algorithm 247. *Commun. ACM*, 7(12): 701, 1964.

Hammersley, J. M. Monte carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences*, 86(3):844–874, 1960.

Hochbaum, D. S. When are np-hard location problems easy? *Annals of Operations Research*, 1(3):201–214, 1984.

Hochbaum, D. S. and Shmoys, D. B. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

Johnson, M. E., Moore, L. M., and Ylvisaker, D. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990.

Jones, B. and Johnson, R. T. Design and analysis for the gaussian process model. *Quality and Reliability Engineering International*, 25(5):515–524, 2009.

Joseph, V. R. Space-filling designs for computer experiments: A review. *Quality Engineering*, 28(1):28–35, 2016.

Jourdan, A. and Franco, J. A new criterion based on kullback-leibler information for space filling designs. *arXiv preprint arXiv:0904.2456*, 2009.

Kenny, Q. Y., Li, W., and Sudjianto, A. Algorithmic construction of optimal symmetric latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Kulesza, A., Taskar, B., et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Lanctot, J. K., Li, M., Ma, B., Wang, S., and Zhang, L. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003.

Lim, A., Rodrigues, B., Wang, F., and Xu, Z. k-center problems with minimum coverage. *Theoretical Computer Science*, 332(1-3):1–17, 2005.

Lin, H. and Bilmes, J. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pp. 510–520, 2011.

Macchi, O. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(1):83–122, 1975.

MacWilliams, F. J. and Sloane, N. J. A. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

Malherbe, C. and Scaman, K. Robustness in multi-objective submodular optimization: a quantile approach. In *International Conference on Machine Learning*, pp. 14871–14886. PMLR, 2022.

Malherbe, C., Grosnit, A., Tutunov, R., Ammar, H. B., and Wang, J. Optimistic tree searches for combinatorial black-box optimization. In *Advances in Neural Information Processing Systems*.

McKay, M. D., Beckman, R. J., and Conover, W. J. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

Mead, R. *The design of experiments: statistical principles for practical applications*. Cambridge university press, 1990.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pp. 234–243. Springer, 1978.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*, pp. 234–243. Springer, 2005.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Morokoff, W. J. and Caflisch, R. E. Quasi-monte carlo integration. *Journal of computational physics*, 122(2): 218–230, 1995.

Müller, W. G. Coffee-house designs. *Optimum design 2000*, pp. 241–248, 2001.

Müller, W. G. *Collecting spatial data: optimum design of experiments for random fields*. Springer Science & Business Media, 2007.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

Niederreiter, H. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.

Pronzato, L. Optimal experimental design and some related control problems. *Automatica*, 44(2):303–325, 2008.

Pronzato, L. Minimax and maximin space-filling designs: some properties and methods for construction. *Journal de la Société Française de Statistique*, 158(1):7–36, 2017.

Pronzato, L. and Müller, W. G. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22:681–701, 2012.

Santner, T. J., Williams, B. J., Notz, W. I., and Williams, B. J. *The design and analysis of computer experiments*, volume 1. Springer, 2003.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Sobol, I. M. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4): 784–802, 1967.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wan, X., Nguyen, V., Ha, H., Ru, B., Lu, C., and Osborne, M. A. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In *International Conference on Machine Learning*, pp. 10663–10674. PMLR, 2021.

Wilson, J., Hutter, F., and Deisenroth, M. Maximizing acquisition functions for bayesian optimization. *Advances in neural information processing systems*, 31, 2018.

Abstract

This appendix contains additional material for the paper "Measures of diversity and space-filling designs for categorical data". In Section 6, we discuss the potential negative social impact of our work. In Section A, we provide numerical methods to compute the diversity metrics. In Section B, we present additional results for Section B and provide the proofs of the results. In Section C, we provide the proofs of Section 4. Finally, Section D describes the setup of the numerical experiments and present the computational times associated with each method.

A. Computation and approximations of the diversity measures

In this section, we provide algorithms and methods to compute the three diversity measures described in Section 2: packing radius, covering radius, and average covering.

A.1. Packing Radius

Exhaustive-computation. The computation of the packing radius is straightforward in the sense that it does not involve any computation over the whole space $\mathcal{X}$. Algorithm 3 provides a direct way to compute the packing radius. Considering that it takes d operations to compute a Hamming distance between two points, the total complexity of the algorithm is thus of order $O(dn^2)$.

Algorithm 3 Packing radius computation

Require: Design $D_n = (x_1, \ldots, x_n)$

Set $\texttt{PR} \leftarrow d$
for $i = 1, ..., n-1$ **do**
  for $j = i+1, ..., n$ **do**
    dist $\leftarrow d_H(x_i, x_j)/2$
    if dist $<$ PR **then**
      PR $\leftarrow$dist
    end if
  end for
end for
Return the packing radius PR

A.2. Covering Radius

Exhaustive-computation. Since the computation of the covering radius involves finding the maximum distance of a point in the total space $\mathcal{X}$ to the design, its direct computation is more numerically intensive than the packing radius. Similarly, Algorithm 4 provides a straightforward way to compute the covering radius. Since it takes dn operations to compute $\min_{i=1\ldots n} d(x, x_i)$ for any given point $x \in \mathcal{X}$, its total complexity is of order $O(dn|\mathcal{X}|)$.

Algorithm 4 Covering radius computation

Require: Design $D_n = (x_1, \ldots, x_n)$

Set $\texttt{CR} \leftarrow 0$
for $x \in \mathcal{X}$ **do**
  dist $\leftarrow \min_{i=1\ldots n} d_H(x, x_i)$
  if dist $>$ CR **then**
    CR $\leftarrow$dist
  end if
end for
Return the covering radius CR

Linear programming. Similarly to the case of identifying an optimal design (Section 3), it is possible to employ tools from linear programming to efficiently compute the covering radius of a design. More precisely, the next result provides an exact

linear programming formulation which allows a faster computation.

Proposition A.1. (Linear programming covering radius). *Let $D_n = (x_1, \ldots, x_n)$ be any design of $n \geq 1$ points with $x_i = (x_i^1, \ldots, x_i^d) \in \mathcal{X}$ for all $1 \leq i \leq n$. Then, the covering radius of the design is the solution of the following linear program:*

$$\texttt{CR}(D_n) = \max_{A\mathbf{x} \leq b} \mathbf{c}^T \mathbf{x}$$

where $x \in \{0,1\}^d \times \{0, \ldots, d\}$, $\mathbf{c} = (0, \ldots, 0, 1) \in \mathbb{R}^{d+1}$, $b = (\|x_1\|_1, \ldots, \|x_n\|_1)^T$, $\mathbf{x} \in \{0,1\}^d \times \{0, \ldots, d\}$ and $A \in \mathbb{R}^{n \times d+1}$ with $a_{i,j} = (2x_i^j + 1)$ for all $(i,j) \in \{1, \ldots, n\} \times \{1, \ldots, d\}$ else 1.

Note that this problem has $d+1$ variables and n constraints. Thus, we can trade the complexity of order $O(dn|\mathcal{X}|)$ of the exact computation method by solving a linear program and resulting in a complexity of $O(LP(d+1, n))$.

A.3. Average Covering

Exhaustive-computation. Similarly to the covering radius, computing the average covering is more challenging in the sense that it involves the computation of the distance of any point of the domain to the design. Algorithm 5 provides a straightforward way to compute the average covering, with a total complexity of order $O(dn|\mathcal{X}|)$.

Algorithm 5 Average covering computation

Require: Design $D_n = (x_1, \ldots, x_n)$

1: Set $\texttt{AC} \leftarrow 0$
2: **for** $x \in \mathcal{X}$ **do**
3: $\quad \text{dist} \leftarrow \min_{i=1\ldots n} d_H(x, x_i)$
4: $\quad \texttt{AC} \leftarrow \texttt{AC} + \text{dist}/|\mathcal{X}|$
5: **end for**
6: **Return** the average covering AC

Approximate evaluation. However, it is important to note that contrarily to the covering radius which can be computed with linear programming tools, here we can compute a provably approximately correct estimation of the average covering using stochastic methods. In particular, the next proposition shows that one can easily compute the average covering using a stochastic approximation.

Proposition A.2. (Average Covering Approximation). *Let $D_n = (x_1, \ldots, x_n)$ be any design of $n \geq 1$ points. Consider the following estimate of the average covering of D_n as defined follows*

$$\widehat{AC}_M(D_n) = \frac{1}{M} \sum_{i=1}^{M} d(X_i, D_n)$$

where $X_1, \ldots, X_M \sim \mathcal{U}(\mathcal{X})$ denotes any series of $M \geq 1$ points independently and uniformly distributed over X. Then, for any $\delta \in (0,1)$, we have the following approximation error which holds with probability at least $1 - \delta$:

$$\texttt{AC}(D_n) - \widehat{\texttt{AC}}_M(D_n) \leq d \cdot \sqrt{\frac{\ln(2/\delta)}{2M}}.$$

This result provides a bound on the error of the stochastic estimate that only uses M points as a proxy for $\mathcal{X}$. More importantly, it highlights the fact that it is only necessary to take $M = O((d/\varepsilon)^2)$ points to have an approximation error of at most $\varepsilon > 0$, resulting in a total complexity of $O(nd^3/\varepsilon^2)$ to compute $\texttt{AC}(D_n)$ instead of $O(dn|\mathcal{X}|)$ provided by the exact computation of the average covering.

A.4. Proofs of the computation results

Proof of Proposition A.1. Fix any $D_n \in \mathcal{X}^n$ and recall that $\texttt{CR}(D_n) = \max_{x \in \mathcal{X}} \; d(x, D_n) = \max_{x \in \mathcal{X}} \min_{i=1\ldots n} d(x, x_i)$

by definition. Thus, by a straightforward rewriting, it can be seen that $\texttt{CR}(D_n)$ is a solution of the following problem:

$$\begin{array}{ll} \max & \mathtt{D} \\ \text{subject to} & \mathtt{D} \leq d(\mathtt{x}, x_i), \ \ \forall i \in \{1, \dots, n\} \\ & \mathtt{x} \in \{0,1\}^d \\ & \mathtt{D} \in \{0, \dots, d\}. \end{array}$$

Indeed, observe that, in the above problem, for any $\mathtt{x} \in \mathbb{Z}^d$, the first constraint is satisfied for all $i \in \{1, \dots, n\}$ and maximum whenever $\mathtt{D} = \min_{i=1\dots n} d(\mathtt{x}, x_i)$. Thus, by taking the maximum over any $\mathtt{x} \in \{0,1\}^d$, we obtain $\max_{\mathtt{x} \in \{0,1\}^d} d(\mathtt{x}, D_n) = \texttt{CR}(D_n)$. However, to turn the problem into a linear program, we need to linearize the first constant. Now observe that for any $\mathtt{x} = (\mathtt{x}_1, \dots, \mathtt{x}_\mathtt{d}) \in \{0,1\}^d$ and $x_i = (x_i^1, \dots, x_i^d)$, we have:

$$d(\mathtt{x}, x_i) = \sum_{j=1}^{d} \mathbb{I}\{\mathtt{x}_j \neq x_i^j\} = \sum_{j=1}^{d} \mathtt{x}_j(1 - x_i^j) + x_i^j(1 - \mathtt{x}_j) = \sum_{j=1}^{d} \mathtt{x}_j(1 - 2x_i^j) + x_i^j$$

which is linear in terms of each of the $\mathtt{x}_\mathtt{j}$. Hence, we have

$$\mathtt{D} \leq d(\mathtt{x}, x_i) \Leftrightarrow \mathtt{D} + \sum_{j=1}^{d} \mathtt{x}_j(2x_i^j - 1) \leq \|x_i\|$$

which is linear. Therefore, one can compute $\texttt{CR}(D_n)$ by solving the following linear problem written in standard form:

$$\begin{array}{ll} \max & \mathbf{c}^T \mathtt{x} \\ \text{subject to} & A\mathtt{x} \leq b \\ & \mathtt{x} \in \{0,1\}^d \times \{0, \dots, d\} \end{array}$$

where $\mathbf{c} = (0, \dots, 0, 1)^T \in \mathbb{R}^{d+1}$, $b = (\|x_1\|, \dots, \|x_n\|)^T$, $\mathtt{x} = (\mathtt{x}_1, \dots, \mathtt{x}_d, \mathtt{D})^T$ using the previous notations and $A \in \mathbb{R}^{n \times d+1}$ with $a_{i,j} = 2x_i^j + 1$ if $j \leq d$ and $a_{i,d+1} = 1$ for all $i = 1, \dots, n$ further described below:

$$A = \begin{pmatrix} (2x_1^1 - 1) & \cdots & (2x_1^d - 1) & 1 \\ (2x_2^1 - 1) & \cdots & (2x_2^d - 1) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ (2x_n^1 - 1) & \cdots & (2x_n^d - 1) & 1 \end{pmatrix}.$$

Proof of Proposition A.2. First, since the random variables $X_1, \dots, X_M$ are independent, it follows that $d(X_1, D_n), \dots, d(X_M, D_n)$ are also independent and identically distributed. As a direct consequence, it follows that

$$\mathbb{E}\left[\widehat{AC}_M(D_n)\right] = \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} d(X_i, D_n)\right] = \mathbb{E}[d(X_1, D_n)] = \texttt{AC}(D_n).$$

Moreover, since $0 \leq d(X_1, D_n)/M \leq d/M$ by definition, a straightforward application of Hoeffding's inequality gives that for all $t \geq 0$:

$$\mathbb{P}\left(\texttt{AC}(D_n) - \widehat{AC}_M(D_n) \geq t\right) \leq 2\exp\left(-\frac{2Mt^2}{d^2}\right).$$

Now, observing that $2e^{-2Mt^2/d^2} \leq \delta$ when $t = d\sqrt{\frac{\ln(2/\delta)}{2M}}$ gives the result.

B. Supplementary material for Section 3

In this section, we provide additional results for Section 3 as well as the proofs of the results.

B.1. Additional results

First, we start to case a simple result about the non-increasing property of the diversity measures that will be used in the main results of the paper.

Lemma B.1. (Non-increasing property). *For any diversity measure $\ell \in \{\texttt{CR}(\cdot), \texttt{PR}(\cdot), \texttt{AC}(\cdot)\}$, we have the following non-increasing property:*

$$\ell(D_n \cup \{z\}) \leq \ell(D_n), \ \ \forall z \in \mathcal{X}$$

where $D_n \in \mathcal{X}^n$ denotes any design of n points in $\mathcal{X}$.

Interestingly, this property also points out that this non-decreasing property motivates the use of incremental algorithms that sequentially add points to a given design to decrease their diversity measure (as used in GIPPR and GAC). However, a more straightforward consequence is that it will be used to prove Proposition 3.1. In the continuation of this result, we provide several results that will be used to prove Proposition 3.2.

Lemma B.2. *Consider any design $D_n \in \mathcal{X}$ of any size $n \geq 2$. Then, we have:*

$$\texttt{AC}(D_n) \leq \texttt{CR}(D_n).$$

Proposition B.3. *For any dimension $d \geq 2$ and any design size $n \geq 1$, we have the following inequalities:*

$$\texttt{PR}^*_{n+1} \leq \texttt{CR}^*_n \ \text{ and } \ \texttt{AC}^*_n \leq \texttt{CR}^*_n$$

where $\texttt{CR}^*_n = \min_{D_n \in \mathcal{X}^n} \texttt{PR}(D_n)$, $\texttt{AC}^*_n = \min_{D_n \in \mathcal{X}^n} \texttt{AC}(D_n)$ and $\texttt{PR}^*_{n+1} = \max_{D_{n+1} \in \mathcal{X}^{n+1}} \texttt{PR}(D_n)$ denotes the optimal metrics.

Proposition B.4. (Bounds on optimal covering radius). *Consider any dimension $d \geq 2$ of the categorical space and let D^*_n be an optimal design solving (1) where $\ell = \texttt{CR}$ of any size $n \geq 4$. Then, the following inequalities hold:*

$$\frac{d}{2} - \ln_2(n) \leq \texttt{CR}(D^*_n) \leq \left\lfloor \frac{d}{2} - \frac{\lfloor \ln_2(n/2) \rfloor}{2} \right\rfloor$$

and $\texttt{CR}^*_2 = \texttt{CR}(D^*_2) = \lfloor d/2 \rfloor$.

Proposition B.5. (Bounds on optimal packing radius). *Consider any dimension $d \geq 2$ of the categorical space and let D^*_n be an optimal design solving (1) of any size $n \geq 5$. Then, the following inequalities hold:*

$$\left\lfloor \frac{d}{2} - \ln_2(n) \right\rfloor \leq \texttt{PR}^*_n \leq \left\lfloor \frac{d}{2} - \frac{\lfloor \ln_2((n-1)/2) \rfloor}{2} \right\rfloor.$$

where $\texttt{PR}^*_2 = d/2$.

Proposition B.6. (Upper bound on optimal average covering). *Consider any dimension $d \geq 2$ of the categorical space and let D^*_n be an optimal design solving (1) of any size $n \geq 5$. Then, the following inequalities hold:*

$$\texttt{AC}^*_n \leq \left\lfloor \frac{d}{2} - \frac{\lfloor \ln_2(n/2) \rfloor}{2} \right\rfloor.$$

where $\texttt{AC}^*_1 = d/2$.

B.2. Proofs of the results of Section 3

Here, we provide the proofs of the results.

Proof of Lemma B.1. For simplicity, we will directly prove the result for any nested designs $D \subseteq D' \subseteq \mathcal{X}$. Now, observe that:

$$\begin{aligned}
\texttt{CR}(D) &= \max_{x \in \mathcal{X}} \min_{y \in D} d_H(x, y) && (2)\\
&= \max_{x \in \mathcal{X}} \min_{y \in D \cup (D' \setminus D)} d_H(x, y) && (3)\\
&\geq \max_{x \in \mathcal{X}} \min_{y \in D'} d_H(x, y) && (4)\\
&= \texttt{CR}(D'). && (5)
\end{aligned}$$

For the average covering, we have:

$$\texttt{AC}(D) = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} \min_{y\in D} d_H(x,y) = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} \min_{y\in D\cup(D'\setminus D)} d_H(x,y) \geq \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} \min_{y\in D'} d_H(x,y) = \texttt{AC}(D').$$

Finally, since $D \subseteq D'$,

$$\texttt{PR}(D) = \min_{\substack{x,y\in D\\ x\neq y}} d_H(x,y) \geq \min_{\substack{x,y\in D'\\ x\neq y}} d_H(x,y) = \texttt{PR}(D').$$

Proof of Proposition B.2. For the average covering, we have

$$\texttt{AC}(D_n) = \frac{1}{|\mathcal{X}|}\sum_{i=1}^{n} d(x, D_n) \leq \frac{1}{n}\sum_{i=1}^{n} \max_{x\in\mathcal{X}} d(x, D_n) = \texttt{CR}(D_n).$$

Proof of Proposition B.3. First, we prove that $\texttt{PR}^*_{n+1} \leq \texttt{CR}^*_n$. Consider any $n \geq 1$ and let $x_1^{\texttt{CR}}, \ldots, x_n^{\texttt{CR}}$ be any design with minimum covering radius and $x_1^{\texttt{PR}}, \ldots, x_{n+1}^{\texttt{PR}}$ be any design achieving optimal Packing Radius. Then, since the search space $\mathcal{X} \subseteq \bigcup_{i=1}^n B(x_i^{\texttt{CR}}, \texttt{CR}^*_n)$ is included in a union of n balls by definition of the covering radius and we have $n+1$ points in the packing radius design, it necessarily follows that one of the ball $B(x_{i^*}^{\texttt{CR}}, \texttt{CR})$ contains two points $x_{i_1}^{\texttt{PR}}$ and $x_{i_2}^{\texttt{PR}}$ of the PR design. Then, by definition of the Packing Radius, it follows that:

$$\texttt{PR}^*_{n+1} = \min_{1\leq i\neq j\leq n+1} \frac{d_H(x_i, x_j)}{2} \leq \frac{d_H(x_{i_1}^{\texttt{PR}}, x_{i_2}^{\texttt{PR}})}{2} \leq \frac{(d_H(x_{i_1}^{\texttt{PR}}, x_{i^*}^{\texttt{CR}}) + d_H(x_{i^*}^{\texttt{CR}}, x_{i_2}^{\texttt{PR}}))}{2} \leq \texttt{CR}^*_n$$

which proves the first part of the result. We now prove that $\texttt{AC}^*_n \leq \texttt{CR}^*_n$. Let $D^*_n \in \arg\min_{D_n\in\mathcal{X}^n} \texttt{CR}(D_n)$. Then, we have that:

$$\texttt{AC}^*_n = \min_{D_n\in\mathcal{X}^n} \texttt{AC}(D_n) \leq \texttt{AC}(D^*_n) \leq \texttt{CR}(D^*_n) = \texttt{CR}^*_n$$

which concludes the proof.

Proof of Proposition B.4. We start to prove the lower bound. First, note that since $\texttt{CR}^*_1 = d$ and $\texttt{CR}^*_2 = \lfloor d/2 \rfloor$, the result trivially holds whenever $n < \min\{n \in \mathbb{N} : \texttt{CR}^*_n < d/2\}$ and $n > 2^{d/2}$. Now, consider any $\min\{n \in \mathbb{N} : \texttt{CR}^*_n < d/2\} \leq n < 2^d$ and let $D_n = (x_1, \ldots, x_n) \in \mathcal{X}^n$ be any design of n points minimizing the covering radius. Since, $\texttt{CR}^*_n = \max_{x\in\mathcal{X}} d_H(x, D_n)$, it necessarily follows that $\mathcal{X} \subseteq \bigcup_{i=1}^n B_H(x_i, \texttt{CR}^*_n)$ by definition. Thus, since both these sets are finite, we have that:

$$\begin{aligned}
2^d &= |\mathcal{X}| && (6)\\
&\leq \left|\bigcup_{i=1}^n B_H(x_i, \texttt{CR}^*_n)\right| && (7)\\
&\leq \sum_{i=1}^n |B_H(x_i, \texttt{CR}^*_n)| && (8)\\
&= n|B_H(x_1, \texttt{CR}^*_n)| && (9)\\
&= n\sum_{i=0}^{\texttt{CR}^*_n} \binom{d}{i} && (10)\\
&\leq n2^{dH\left(\frac{\texttt{CR}^*_n}{d}\right)} && (11)\\
&\leq n2^{d\frac{(1+2\texttt{CR}^*_n/d)}{2}} && (12)\\
&= n2^{\frac{d}{2}+\texttt{CR}^*_n} && (13)
\end{aligned}$$

where we used the fact that $\sum_{i=0}^k \binom{d}{i} \leq 2^{dH(k/n)}$ for any $0 < k < n/2$ where $H(x) = -x\log(x) - (1-x)\log(1-x)$ denote the binary cross entropy (see (MacWilliams & Sloane, 1977) Chapter 10) and we used the fact that $H(x) \leq (1+2x)/2$ for any $x \leq 1/2$ on lines (6) and (7). Finally, taking the logarithm on both sides gives that $d - \ln_2(n) \leq \frac{d}{2} + \texttt{CR}^*_n$ which proves the lower bound.

Upper bound for the covering radius. To prove the upper bound, we use the fact that $\min_{D_n \in \mathcal{X}^n} \texttt{CR}(D_n) \leq \texttt{CR}(D_n^{UB})$ for a design D_n^{UB} that is inspired by the construction provided in (Malherbe et al.). Fix any $n \geq 4$ and set $L = \lfloor \ln_2(n/2) \rfloor$ which has been chosen so that $2 \times 2^L \leq n$. Now, consider the design:

$$D_n^{UB} = D_{2^L}^0 \cup D_{2^L}^1$$

where

$$D_{2^L}^0 = \{(x_1, \dots, x_L, 0, \dots, 0) : (x_1, \dots, x_L) \in \{0,1\}^L\}$$

and

$$D_{2^L}^1 = \{(x_1, \dots, x_L, 1, \dots, 1) : (x_1, \dots, x_L) \in \{0,1\}^L\}$$

Observe that D^0 and D_1 respectively contains all the possible combinations of L elements filled with 0 (resp. 1) for the remaining elements. Moreover, note that $|D_n^{UB}| = |D_{2^L}^0| + |D_{2^L}^1| \leq 2 \times 2^L \leq n$ by definition of L. Now, for any $x = (x_1, \dots, x_d) \in \mathcal{X}$, observe that there necessarily $x^0 \in D_{2^L}^0$ and $x^1 \in D_{2^L}^1$ that share the same first L elements as x, i.e. $x_{[1,\dots,L]} = x^0_{[1,\dots,L]} = x^1_{[1,\dots,L]}$ which implies that

$$\begin{aligned}
d_H(x, D_n^{UB}) &\leq \min\left(d_H(x, x^0), d_H(x, x^1)\right) \\
&= \min\left(\sum_{i=1}^{L} \mathbb{I}\{x_i^0 \neq x_i\} + \sum_{i=L+1}^{d} \mathbb{I}\{x_i^0 \neq x_i\}, \sum_{i=1}^{L} \mathbb{I}\{x_i^1 \neq x_i\} + \sum_{i=L+1}^{d} \mathbb{I}\{x_i^1 \neq x_i\}\right) \\
&= \min\left(\sum_{i=L+1}^{d} x_i, \sum_{i=L+1}^{d} (1 - x_i)\right) \\
&= \min\left(\sum_{i=L+1}^{d} x_i, (d-L) - \sum_{i=L+1}^{d} x_i\right) \\
&= (d-L) \min\left(\sum_{i=L+1}^{d} x_i/(d-L), 1 - \sum_{i=L+1}^{d} x_i/(d-L)\right) \\
&\leq \frac{d-L}{2}
\end{aligned}$$

where we used on the last inequality the fact that $\sum_{i=L+1}^{d} x_i/(d-L) \in [0,1]$ and that $\min(x, 1-x) \leq 1/2$ for all $x \in [0,1]$. Since the previous results hold for any $x \in \mathcal{X}$, and the covering radius only takes integer values, we deduce that

$$\min_{D_n \in \mathcal{X}^n} \texttt{CR}(D_n) \leq \texttt{CR}(D_n^{UB}) \leq \left\lfloor \frac{d-L}{2} \right\rfloor = \left\lfloor \frac{d}{2} - \frac{\lfloor \ln_2(n/2) \rfloor}{2} \right\rfloor$$

which proves the result.

Algorithm 6 Lower Bound Algorithm

Require: Number $n \geq 2$ of points, parameter $D \geq 1$ such that $n|B(0,D)| \leq 2^d$

1: $\mathcal{X}_0 \leftarrow \mathcal{X}$
2: $\hat{D}_0 \leftarrow \emptyset$
3: **for** $t = 0, ..., n-1$ **do**
4: Pick any point $x_{t+1} \in \mathcal{X}_t$
5: Add the point to the design: $\hat{D}_{t+1} \leftarrow \hat{D}_t \cup \{x_{t+1}\}$
6: Remove the points from the current point $\mathcal{X}_{t+1} \leftarrow \mathcal{X}_t / B(x_{t+1}, D)$
7: **end for**
8: **Return** design $\hat{D}_n$

Proofs of Proposition B.5. We start to prove the lower bound. We use the fact that $\max_{D_n} \texttt{PR}(D_n) \geq \texttt{PR}(D_n)$ for any design D_n of size n. We investigate a specific class of designs $\hat{D}_n^D$ provided by Algorithm 6. For this algorithm, since $d(x, x') \geq D$ for all $x \neq x' \in \hat{D}_n^D$, it follows that $\texttt{PR}(\hat{D}_n^D) \geq D/2$. Moreover, since at each time step we remove at

most $|B(0,D)|$ points (line 6) and the initial size of the space is $|\mathcal{X}| = 2^d$, the algorithm does indeed terminates whenever $n|B(0,D)| \leq 2^d$. Thus, with a budget of n points, we can get a design with a packing radius satisfying:

$$\max_{D_n \in \mathcal{X}^n} \texttt{PR}(D_n) \geq \max_{D \geq 0} \texttt{PR}(\hat{D}_n^D) \tag{14}$$

$$\geq \frac{1}{2} \max_{D \geq 0} \{|B(0,D)| \leq 2^d/n\} \tag{15}$$

$$\geq \frac{1}{2} \max_{D \geq 0} \{\sum_{i=0}^{D} \binom{d}{i} \leq 2^d/n\} \tag{16}$$

$$\geq \frac{1}{2} \max_{D \geq 0} \{2^{dH(D/d)} \leq 2^d/n\} \tag{17}$$

$$\geq \frac{1}{2} \max_{D \geq 0} \{2^{\frac{d(1+D/d)}{2}} \leq 2^d/n\} \tag{18}$$

$$= \frac{1}{2} \max_{D \geq 0} \{\frac{d}{2} + \frac{D}{2} \leq d - \log_2(n)\} \tag{19}$$

$$= \frac{1}{2} \max_{D \geq 0} \{D \leq d - 2\log_2(n)\} \tag{20}$$

$$= \frac{\lfloor d - 2\log_2(n) \rfloor}{2} \tag{21}$$

$$\geq \lfloor \frac{d}{2} - \log_2(n) \rfloor \tag{22}$$

where we used the fact that $\sum_{i=0}^{D} \binom{d}{i} \leq 2^{dH(D/d)}$ where $H(x)$ denotes the binary gross entropy on line (26) and the fact that $H(x) \leq (1+x)/2$ for any $x \leq 1/2$ which proves the result. For the upper bound, we directly use Proposition B.3 and B.4.

Proof of Proposition B.6. Similarly to the previous result, the result is obtained by combining Propositions B.3 and B.4.

Proof of Proposition 3.1. Consider any $n \geq 2$. We first prove the result with the covering radius. Let D_n^* and D_{n+1}^* be an optimal covering solving (1) for the covering radius and consider any point $e \in \mathcal{X}/D_n^*$. Then, using the fact that $\texttt{CR}$ is non-increasing (Lemma B.1), we have:

$$\min_{D_n \in \mathcal{X}^n} \texttt{CR}(D_n) = \texttt{CR}(D_n^*) \geq \texttt{CR}(D_n^* \cup \{e\}) \geq \min_{D_{n+1} \in \mathcal{X}^{n+1}} \texttt{CR}(D_{n+1}) = \texttt{CR}(D_{n+1}^*)$$

which gives the non-increasing property for the covering radius. Now reproducing the exact same steps with the average covering, we obtain:

$$\texttt{AC}(D_n^*) \geq \texttt{AC}(D_n \cup \{e\}) \geq \min_{D_{n+1} \in \mathcal{X}^{n+1}} \texttt{AC}(D_{n+1}^*).$$

Finally, for the packing radius, considering any $e \in D_{n+1}$, we have:

$$\texttt{PR}(D_{n+1}^*) \leq \texttt{PR}(D_{n+1}/e) \leq \max_{D_n \in \mathcal{X}^n} \texttt{PR}(D_n) = \texttt{PR}(D_n^*)$$

which concludes the proof.

Proof of Proposition 3.2. To prove the result, we use the worst lower and upper bound of Propositions B.4 and B.5. Thus, we have:

$$\Big\lfloor \frac{d}{2} - \ln_2(n) \Big\rfloor \leq \ell(D_n^*) \leq \Big\lfloor \frac{d}{2} - \frac{\lfloor \ln_2((n-1)/2) \rfloor}{2} \Big\rfloor.$$

for both the covering and packing radius. For the lower bound, we directly have:

$$\Big\lfloor \frac{d}{2} - \ln_2(n) \Big\rfloor \geq \Big\lfloor \Big\lfloor \frac{d}{2} \Big\rfloor - \ln_2(n) \Big\rfloor = \Big\lfloor \frac{d}{2} \Big\rfloor - \lceil \ln_2(n) \rceil.$$

Finally, when d is odd, we have the following for the upper bound:

$$\left\lfloor \frac{d}{2} - \frac{\lfloor \ln_2(n-1/2) \rfloor}{2} \right\rfloor = \left\lfloor \left\lfloor \frac{d}{2} \right\rfloor + \frac{1}{2} - \frac{\lfloor \ln_2((n-1)/2) \rfloor}{2} \right\rfloor \tag{23}$$

$$= \left\lfloor \left\lfloor \frac{d}{2} \right\rfloor - \frac{\lfloor \ln_2(n-1) \rfloor}{2} \right\rfloor \tag{24}$$

$$= \left\lfloor \frac{d}{2} \right\rfloor - \left\lceil \frac{\lfloor \ln_2(n-1) \rfloor}{2} \right\rceil \tag{25}$$

$$\leq \left\lfloor \frac{d}{2} \right\rfloor - \frac{\ln_2(n-1)}{2} \tag{26}$$

which concludes the proof.

Proof of Proposition 3.3. The statement holds because for any permutation $\pi \colon \{1, \dots, n\} \to \{1, \dots, n\}$ and any point $z \in \mathcal{X}$, the functions $\pi \colon \mathcal{X} \to \mathcal{X}$ defined by $\pi(x)_i = x_{\pi(i)}$ and $x \mapsto x \oplus z$ are isometries, and the composition of two isometries is an isometry. We will prove the result for any design $D_n \in \mathcal{X}^n$. For the first function, this follows from the fact that the hamming distance is symmetrical on the coordinates:

$$d_H(\pi(x), \pi(y)) = \sum_{i=1,\dots,d} \mathrm{I}\{x_{\pi(i)} \oplus y_{\pi(i)}\} = \sum_{i=\pi(1),\dots,\pi(d)} \mathrm{I}\{x_i \oplus y_i\} = \sum_{i=1,\dots,d} \mathrm{I}\{x_i \oplus y_i\} = d_H(x, y).$$

For the second, it follows from the fact that the XOR operation is commutative and is its own inverse.

$$d_H(x \oplus z, y \oplus z) = \sum_{i=1,\dots,d} \mathrm{I}\{x_i \oplus z_i \oplus y_i \oplus z_i\}$$

$$= \sum_{i=1,\dots,d} \mathrm{I}\{x_i \oplus y_i \oplus z_i \oplus z_i\} = \sum_{i=1,\dots,d} \mathrm{I}\{x_i \oplus y_i\} = d_H(x, y).$$

Now, since we know that the function $f : x \to \pi(x) \oplus z$ is an isometry on $\mathcal{X}$ for any π and $z \in \mathcal{X}$ which implies that (1) $d_H(f(x), f(x')) = d_H(x, x')$ for any $(x, x') \in \mathcal{X}^2$, (2) $f(\mathcal{X}) = \mathcal{X}$ and (3) $d_H(f(x), f(x')) = 0$ if and only if $x = x'$, we obtain that:

$$\mathtt{CR}(f(D_n)) = \max_{x \in \mathcal{X}} \min_{x' \in D_n} d_H(x, f(x')) \tag{27}$$

$$= \max_{x \in f(\mathcal{X})} \min_{x' \in D_n} d_H(x, f(x')) \tag{28}$$

$$= \max_{x \in \mathcal{X}} \min_{y \in D_n} d_H(f(x), f(x')) \tag{29}$$

$$= \max_{x \in \mathcal{X}} \min_{y \in D_n} d_H(x, x') \tag{30}$$

$$= \mathtt{CR}(D_n). \tag{31}$$

Similarly, the same holds for the packing radius:

$$\mathtt{PR}(f(D)) = \min_{x \neq x' \in f(D_n)^2} d_H(x, x') \tag{32}$$

$$= \min_{\substack{x, x' \in D_n \\ f(x) \neq f(x')}} d_H(f(x), f(x')) \tag{33}$$

$$= \min_{x \neq x' \in D} d_H(f(x), f(x')) \tag{34}$$

$$= \min_{x \neq x' \in D} d_H(x, x') \tag{35}$$

$$= \mathtt{PR}(D_n). \tag{36}$$

Finally, the case of the average radius is analogous to the case of the covering radius.

Proof of Proposition 3.4. We proceed case by case and show the decomposition for (1) the packing radius, (2) the covering radius, and (3) the average covering.

Packing radius as an integer linear program. Consider any number of points $n \geq 1$ with the points of the design $x_1, \ldots, x_n$ and observe first that the optimal packing radius is solution to the following problem:

$$\begin{array}{ll} \max & r \\ \text{subject to} & r \leq d(x_i, x_j), \ \ \forall (i < j) \in \{1, \ldots, n\}^2 \\ & r \in \{0, \ldots, d\} \end{array}$$

However, the distance $d(x_i, x_j)$ is naturally non-linear with regards to the variables:

$$d(x_i, x_j) = \sum_{k=1}^{d} x_i^k \oplus x_j^k = \sum_{k=1}^{d} x_i^k + x_j^k - 2x_i^k x_j^k.$$

Here, the main trick is to introduce some auxiliary variables $b_{i,k}^k \in \{0, 1\}$ to linearize the XOR operator. In particular, $b_{i,j}^k = x_i^k \oplus x_j^k$ can be written as a solution of the following linear inequalities:

$$\begin{array}{ll} b_{i,j}^k & \leq x_i^k + x_j^k \\ b_{i,j}^k & \geq x_i^k - x_j^k \\ b_{i,j}^k & \geq x_j^k - x_i^k \\ b_{i,j}^k & \leq 2 - x_j^k - x_i^k. \end{array}$$

Thus, by plugging the previous inequalities into the initial formulation, we get that:

$$\begin{array}{lllll} \max & r & & & \\ \text{subject to} & r & \leq \sum_{k=1}^{d} b_{i,j}^k, & & \forall (i < j) \in \{1, \ldots, n\}^2 \\ & b_{i,j}^k & \leq x_i^k + x_j^k & & \forall (i < j) \in \{1, \ldots, n\}^2, k \in \{1, \ldots, d\} \\ & b_{i,j}^k & \geq x_i^k - x_j^k & & \\ & b_{i,j}^k & \geq x_j^k - x_i^k & & \\ & b_{i,j}^k & \leq 2 - x_j^k - x_i^k. & & \end{array}$$

which is a linear program in standard form with $1 + nd + dn(n+1)/2$ variables and $4n(n+1)/2 + dn(n+1)/2$ constraints. It thus results in a program with $O(dn^2)$ variables and $O(dn^2)$ constraints.

Covering radius as a linear program. First, observe that the design minimizing the covering radius is a solution of the following problem: $\min_{D_n \in \mathcal{X}^n} \max_{x \in \mathcal{X}} \ d(x, D_n)$. Now, the first step is to rewrite the problem by linearizing the maximum term as follows:

$$\begin{array}{lll} \min & D & \\ \text{subject to} & D \geq d_x & \forall x \in \mathcal{X} \\ & D \leq d_x + (1 - b_x)d, & \forall x \in \mathcal{X} \\ & 1 = \sum_{x \in \mathcal{X}} b_x, & \end{array}$$

where $b_x \in \{0, 1\}$ are $|\mathcal{X}|$ binary variables and $d_x = d(x, D_n)$ for any $x \in \mathcal{X}$. Now, the second trick is to use the following decomposition to linearize the term d_x by noticing that it is a solution of the following problem:

$$\begin{array}{lll} d_x & \leq d(x, x_i) & \forall i \in \{1, \ldots, n\} \\ d_x & \geq d(x, x_i) - (1 - b_i^x)d & \forall i \in \{1, \ldots, n\} \\ 1 & = \sum_{i=1}^{n} b_i^x & \end{array}$$

by introducing the n binary variables $b_i^x \in \{0, 1\}^n$. Finally, denoting $x = (x^1, \ldots, x^d)$ for any $x \in \mathcal{X}$ and $x_i = (x_i^1, \ldots, x_i^d)$, and observing that $d(x, x_i) = \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k)$ is linear with regard to x_i^k for any $1 \leq k \leq d$, by

putting the previous results altogether we obtain that the optimal covering radius is a solution of following linear program:

$$\begin{array}{lll}
\min & D & \\
\text{subject to} & D \geq d_x, & \forall x \in \mathcal{X} \\
& D \leq d_x + (1 - b_x)d, & \forall x \in \mathcal{X} \\
& 1 = \sum_{x \in \mathcal{X}} b_x, & \\
& d_x \leq \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k), & \forall x \in \mathcal{X},\ i \in \{1, \ldots, n\} \\
& d_x \geq \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k) - (1 - b_i^x) \times d, & \forall x \in \mathcal{X},\ i \in \{1, \ldots, n\} \\
& 1 = \sum_{i=1}^{n} b_i^x & \forall x \in \mathcal{X}
\end{array}$$

which is a linear program with $1 + |\mathcal{X}|(n + 2)$ variables and $1 + |\mathcal{X}|(2n + 3)$ constraints.

Average covering as a linear program. Consider any number of points $n \geq 1$ with the points of the design $x_1, \ldots, x_n$ and observe first that the optimal design D_n for the average covering is a solution of the following problem:

$$\begin{array}{ll}
\min & \displaystyle\sum_{x \in \mathcal{X}} d(x, D_n) \\
\text{subject to} & D_n \in \mathcal{X}^n.
\end{array}$$

Now, the trick is to linearize the term $d_x = d(x, D_n) = \min_{i=1\ldots n} d(x, x_i)$ for any $x \in \mathcal{X}$. Observe that d_x is a solution of the following program:

$$\begin{array}{rll}
d_x & \leq d(x, x_i) & \forall i \in \{1, \ldots, n\} \\
d_x & \geq d(x, x_i) - (1 - b_i^x) \times d & \forall i \in \{1, \ldots, n\} \\
1 & = \sum_{i=1}^{n} b_i^x &
\end{array}$$

by introducing the n binary variables $b_i^x \in \{0, 1\}^n$. Now, denoting $x = (x^1, \ldots, x^d)$ for any $x \in \mathcal{X}$ and $x_i = (x_i^1, \ldots, x_i^d)$, we have that $d(x, x_i) = \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k)$ which is linear with regard to x_i^k for any $1 \leq k \leq d$. Thus, plugging this distance into the previous program we have that the average covering is a solution of the following problem:

$$\begin{array}{lll}
\min & \displaystyle\sum_{x \in \mathcal{X}} d_x & \\
\text{subject to} & d_x \leq \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k), & \forall x \in \mathcal{X}\ \ i \in \{1, \ldots, n\} \\
& d_x \geq \sum_{k=1}^{d} x^k + x_i^k(1 - 2x^k) - (1 - b_i^x) \times d & \forall x \in \mathcal{X},\ \ i \in \{1, \ldots, n\} \\
& 1 = \sum_{i=1}^{d} b_i^x & \forall x \in \mathcal{X}
\end{array}$$

which is an integer linear program with $(n + 1)|\mathcal{X}| + nd$ variables ($n|\mathcal{X}|$ for the b_i, $|\mathcal{X}|$ for the d_x and nd for the $x_i \in D_n$) and $|\mathcal{X}|(2n + 1)$ constraints.

C. Supplementary material for Section 4

In this section, we provide additional results for Section 4 and provide the proofs of the results.

C.1. Additional results

The next result is a crucial property to show that the GIPPR algorithm presents some guarantees.

Lemma C.1. *Let $D_t = (x_1, \ldots, x_t)$ be a collection of $t \geq 1$ points generated by the GIPPR algorithm for any time step $t \geq 1$. Then, we have the following:*

$$\mathit{PR}(D_{t+1}) = \frac{d(x_{t+1}, D_t)}{2} = \frac{\max_{x \in \mathcal{X}}\ d_H(x, D_t)}{2} = \frac{\mathit{CR}(D_t)}{2}.$$

Moreover, to illustrate the fact that the packing radius is not submodular, we simply consider a counterexample:

$$D = \begin{matrix}[0,0,0,0,0] \\ [1,1,1,1,1], \\ [0,0,0,1,1]\end{matrix}\ D' = \begin{matrix}[0,0,0,0,0] \\ [1,1,1,1,1] \\ [0,0,0,1,1] \\ [1,1,1,0,0]\end{matrix}, \text{ and } z = [1, 1, 0, 0, 0].$$

Here, we have

$$|\texttt{PR}(D \cup z) - \texttt{PR}(D)| = 2 - 2 = 0 < 1 = 2 - 1 = |\texttt{PR}(D' \cup z) - \texttt{PR}(D')|$$

which contradicts the submodular assumption.

C.2. Proofs of the results

Proof of Lemma C.1. We show the result by induction. When $t = 1$, observe that $x_2 \in \arg\max_{x\in\mathcal{X}} d(x, x_1)$ by definition. Thus, we have $\texttt{PR}(D_2) = d(x_2, x_1)/2 = d(x_2, D_1)/2 = \max_{x\in\mathcal{X}} d(x, D_1)/2 = \texttt{CR}(D_1)/2$ by construction. Now assume that the result holds for any $t \geq 2$ (i.e., $\texttt{PR}(D_t) = \texttt{CR}(D_{t-1})/2$) and denote by $x_{t+1} \in \arg\max_{x\in\mathcal{X}} d(x, D_t)$ the next point of the GRIPPR algorithm. Then, we have

$$\begin{aligned}
\texttt{PR}(D_{t+1}) &= \min_{1\leq i<j\leq t+1} d(x_i, x_j)/2 && (37)\\
&= \min\left(\min_{1\leq i\leq t} d(x_i, x_{t+1})/2, \min_{1\leq i<j\leq t} d(x_i, x_j)/2\right) && (38)\\
&= \min\left(d(x_{t+1}, D_t)/2, \texttt{PR}(D_t)\right) && (39)\\
&= \min\left(\max_{x\in\mathcal{X}} d(x, D_t), \max_{x\in\mathcal{X}} d(x, D_{t-1})\right)/2 && (40)\\
&= \max_{x\in\mathcal{X}} d(x, D_t)/2 && (41)\\
&= d(x_{t+1}, D_t)/2 && (42)\\
&= \texttt{CR}(D_t)/2 && (43)
\end{aligned}$$

which proves the result by induction.

Proof of lemma 4.2. First, obverse that for any $e \in \mathcal{X}$ and any $t \geq 2$, we have:

$$\begin{aligned}
\texttt{PR}(D_t \cup \{e\}) &= \min(\min_{i=1\ldots t} d_H(e, x_i)/2, \min_{i<j\leq t} d_H(x_i, x_j)/2) && (44)\\
&= \min(d(e, D_t)/2, \texttt{PR}(D_t)) && (45)\\
&= \min(d(e, D_t)/2, \max_{x\in\mathcal{X}} d(x, D_{t-1})/2) && (46)\\
&\leq \min\left(\max_{x\in\mathcal{X}} d(x, D_t), \max_{x\in\mathcal{X}} d(x, D_{t-1})\right)/2 && (47)\\
&= \max_{x\in\mathcal{X}} d(x, D_t)/2 && (48)
\end{aligned}$$

where we used on the last line that $d(e, D_t) \leq \max_{x\in\mathcal{X}} d(x, D_t)$ for all $e \in \mathcal{X}$. Now, observing that the previous inequality turns into an equality whenever $e \in \arg\max_{x\in\mathcal{X}} d(x, D_t)$ proves that

$$\arg\max_{x\in\mathcal{X}} d(x, D_t) \subseteq \arg\max_{x\in\mathcal{X}} \texttt{PR}(D_t \cup \{x\}).$$

To prove that it is an equivalence, pick any $e \notin \arg\max_{x\in\mathcal{X}} d(x, D_t)$ and note that it implies that:

$$d(e, D_t) < \max_{x\in\mathcal{X}} d(x, D_t) \leq \max_{x\in\mathcal{X}} d(x, D_{t-1})$$

which translates into the fact that

$$\begin{aligned}
\texttt{PR}(D_t \cup \{e\}) &= \min(d(e, D_t), \max_{x\in\mathcal{X}} d(x, D_{t-1}))/2 && (49)\\
&= d(e, D_t)/2 && (50)\\
&< \max_{x\in\mathcal{X}} d(x, D_t)/2 && (51)\\
&= \max_{x\in\mathcal{X}} \texttt{PR}(D_t \cup \{x\}) && (52)
\end{aligned}$$

and we deduce that $e \notin \arg\max_{x\in\mathcal{X}} d(x, D_t)\} \Rightarrow e \notin \arg\max_{x\in\mathcal{X}} \texttt{PR}(D_t \cup \{e\})$ which proves the second part of the equivalence.

Proof of Theorem 4.3. Denote by $D_t = (x_1, \dots, x_t)$ be the design of $t \geq 1$ points generated by the GIPPR algorithm after $t \geq 1$ steps. Now, by virtue of Lemma C.1, we know that $\texttt{PR}(D_{n+1}) = d(x_{n+1}, D_n)/2 = \texttt{CR}(D_n)/2$ where $D_{n+1} = (x_1, \dots, x_{n+1})$ denotes a design generated by the GIPPR with size $n+1$. Now consider a design $D_n^* = (x_1^*, \dots, x_n^*)$ of size $n \geq 1$ with optimal covering radius, i.e., $D_n^* \in \arg\min_{D_n \in \mathcal{X}^n} \texttt{CR}(D_n)$. Then, similarly to the proof of Proposition B.3, since there are $n+1$ points in D_{n+1}, we necessarily have one ball $B(x_i^*, \texttt{CR}(D_n^*))$ centered around a point $x_i^* \in D_n^*$ of the optimal design of radius $\texttt{CR}_n$ which contains two points $x_i, x_j \in D_{n+1} \times D_{n+1}$, implying that: $\texttt{PR}(D_{n+1}) \leq d_H(x_i, x_j)/2 \leq \texttt{CR}(D_n^*)$. Therefore, we deduce from Lemma 4.2 that:

$$\frac{\texttt{CR}(D_n)}{2} = \texttt{PR}(D_{n+1}) \leq \texttt{CR}(D_n^*) = \texttt{CR}_n^*,$$

which proves that $\texttt{CR}(D_n) \leq 2\texttt{CR}_n^*$. Similarly, when taking a design $D_{n+1}^* = (x_1^*, \dots, x_{n+1}^*)$ with optimal parking radius (i.e., $D_{n+1}^* \in \arg\max_{D_{n+1} \in \mathcal{X}^{n+1}} \texttt{PR}(D_{n+1})$) using the same covering argument, we deduce from Lemma 4.2 that

$$\texttt{PR}_{n+1}^* = \texttt{PR}(D_{n+1}^*) \leq \texttt{CR}_n^* \leq \texttt{CR}(D_n) = 2\texttt{PR}(D_{n+1})$$

which proves that $\texttt{PR}(D_{n+1}) \geq \texttt{PR}_{n+1}^*/2$ and concludes the proof.

Proof of Lemma 4.6. Without loss of generality, we prove that the function $F : D \mapsto \texttt{AC}(D)$ if $D \neq \emptyset$ and d otherwise is non-increasing. Observing that for any point $x \in \mathcal{X}$ and any sets $\emptyset \neq D \subseteq D' \subseteq \mathcal{X}$, we have that $d_H(x, D) \geq d_H(x, D')$. Thus, summing over all the points $x \in \mathcal{X}$, it follows that

$$F(D) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} d_H(x, D) \geq \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} d_H(x, D') = F(D')$$

which proves that the function is non-increasing for any $D \neq \emptyset$. Now, since $F(D) \leq d/2$ whenever $|D| > 0$ and $F(\emptyset) = d$, the result also holds when $D = \emptyset$. Now, to prove that the function $-F$ is submodular, consider the function $F_x(D) = -d_H(x, D)$ for any $x \in \mathcal{X}$. Pick any point $e \in \mathcal{X}/D$ and observe first that $F_x(D \cup \{e\}) = -\min(d_H(x, D), d_H(x, e))$. Thus, it follows that

$$\begin{aligned} F_x(D \cup \{e\}) - F(D) &= d_H(x, D) - \min(d_H(x, D), d_H(x, e)) \\ &= \max(0, d_H(x, D) - d_H(x, e)). \end{aligned}$$

Now, for any $\emptyset \neq D \subseteq D' \subset X$ and $e \in \mathcal{X}/D'$, observe that:

$$\begin{aligned} F_x(D \cup \{e\}) - F_x(D) &= \max(0, d_H(x, D) - d_H(x, e)) \\ &\geq \max(0, d_H(x, D') - d_H(x, e)) \\ &= F_x(D' \cup \{e\}) - F_x(D'). \end{aligned}$$

Hence, we deduce that the function F_x is submodular for any $x \in \mathcal{X}$. Finally, since $-F(D) = (1/|\mathcal{X}|) \sum_{x \in \mathcal{X}} F_x(D)$ is a sum of submodular functions, we deduce that $-F$ is submodular.

Proof of Theorem 4.7. Recall that by virtue of Lemma 4.6, we know that the function $F : D \mapsto -\texttt{AC}(D)$ is submodular. Now denote by $D_n^* = (x_1^*, \dots, x_n^*)$ an optimal design and by $D_n = (x_1, \dots, x_n)$ a design generated by the GAC algorithm and by $\Delta(e|D) = \texttt{AC}(D) - \texttt{AC}(D \cup \{e\})$ the marginal improvement of adding a points e to a design D. Now note that for any $i \in \{1, \dots, n-1\}$, we have:

$$\begin{aligned} -\texttt{AC}_n^* &= -\texttt{AC}(D_n^*) && (53) \\ &\leq -\texttt{AC}(D_i \cup D_n^*) && (54) \\ &= -\texttt{AC}(D_i) + \sum_{t=1}^{n} \Delta(x_t^* | D_i \cup \{x_1^*, \dots, x_{t-1}^*\}) && (55) \\ &\leq -\texttt{AC}(D_i) + \sum_{t=1}^{n} \Delta(x_t^* | D_i) && (56) \\ &\leq -\texttt{AC}(D_i) + \sum_{t=1}^{n} \Delta(x_{i+1} | D_i) && (57) \\ &= -\texttt{AC}(D_i) + n\Delta(x_{i+1} | D_i) && (58) \end{aligned}$$

where we used on the second line that the average covering is monotonic. On the fourth line $-\texttt{AC}(\cdot)$ is submodular. On the fifth line that $\Delta(x_{i+1}|D_i) \geq \Delta(x_t^*|D_i)$ since $x_{i+1} \in \arg\max_{x\in\mathcal{X}} \Delta(x_{i+1}|D_i)$ by definition of the algorithm. Thus, we have proved so far that:

$$\Delta(x_{i+1}|D_i) \geq \frac{1}{n}(\texttt{AC}(D_i) - \texttt{AC}_n^*) \;\; \forall i \in \{1, \dots, n-1\}.$$

Now, setting $\delta_i = \texttt{AC}(D_i) - \texttt{AC}_n^*$, it implies that $\delta_i - \delta_{i+1} = \texttt{AC}(D_i) - \texttt{AC}(D_{i+1}) = \Delta(x_{i+1}|D_i)$. Plugging this into the previous translates into $\delta_i - \delta_{i+1} \geq \delta_i/n$, implying that

$$\delta_{i+1} \leq \left(1 - \frac{1}{n}\right)\delta_i, \;\; \forall i \in \{1, \dots, n-1\}.$$

Applying this inequality recursively gives that:

$$\texttt{AC}(D_n) - \texttt{AC}_n^* \leq \left(1 - \frac{1}{n}\right)^{n-1} (\texttt{AC}(D_1) - \texttt{AC}_n^*).$$

Now using the fact that $(1 - 1/n)^{n-1} \leq 1/2$ for any $n \geq 2$, gives

$$\texttt{AC}(D_n) - \texttt{AC}_n^* \leq \frac{(\texttt{AC}(D_1) - \texttt{AC}_n^*)}{2}.$$

Now observing that $\texttt{AC}(D_1) = \texttt{AC}_1^*$, it translates into:

$$\texttt{AC}(D_n) - \texttt{AC}_1^* + \texttt{AC}_1^* - \texttt{AC}_n^* \leq \frac{1}{2}(\texttt{AC}_1^* - \texttt{AC}_n^*)$$

which gives

$$\frac{1}{2}(\texttt{AC}_1^* - \texttt{AC}_n^*) \leq \texttt{AC}_1^* - \texttt{AC}(D_n)$$

and proves the result.

D. Supplementary material for Section 5

Here, we fully describe the setup of the numerical experiments presented in Section 5 and provide associated complexity results.

D.1. Numerical complexity of the methods

Here we discuss the theoretical and empirical complexity of the algorithms introduced in the paper.

Numerical complexity (theoretical). Table D.1 reports the worst-case complexity of the algorithms presented in the paper. LP(X, Y) denotes solving a linear program with X variables and Y constraints. Exact corresponds to the methods introduced in Section 3. Approximate corresponds to GIPPR for the PR and CR and to GAC for the AC. Approximate (Exhaustive) corresponds to these algorithms where we replaced the resolution of the linear program by an exhaustive search. Last, in order to have another comparison, it is also interesting to recall that the complexity of generating a k-DPP using the standard Cholesky decomposition is of order $O(|\mathcal{X}|^3)$ as pointed out in (Calandriello et al., 2020).

	PR	CR	AC
Exact (exhaustive)	$\binom{\lvert\mathcal{X}\rvert}{n} n^2 d$	$\binom{\lvert\mathcal{X}\rvert}{n} nd\lvert\mathcal{X}\rvert$	$\binom{\lvert\mathcal{X}\rvert}{n} nd\lvert\mathcal{X}\rvert$
Exact (LP)	LP(dn^2, dn^2)	LP$(n\lvert\mathcal{X}\rvert, n\lvert\mathcal{X}\rvert)$	LP$(n\lvert\mathcal{X}\rvert, n\lvert\mathcal{X}\rvert)$
Approximate (LP)	nLP$(d+1, n)$	nLP$(d+1, n)$	nLP$(d, \lvert\mathcal{X}\rvert)$
Approximate (Exhaustive)	$dn^2\lvert\mathcal{X}\rvert$	$dn^2\lvert\mathcal{X}\rvert$	$dn^2\lvert\mathcal{X}\rvert^2$

Table 1. Numerical complexity of identifying an optimal design for the different covering measures. LP(x, y) denotes solving a standard integer linear program with x variables and y constraints.

Numerical complexity (practical). To provide an example of the time required to create a design in practice, we recorded the time to generate each design. Of course, we point out that these results are only indicative in the sense that they can

drastically vary depending on both the hardware and the specific implementation of the algorithms. All the times have been recorded on the same laptop computer with an Intel i7 CPU @ 1.80GHz 1.99 GHz with 16GB of RAM and all the linear programs have been solved using the open-source SCIP solver, in python 3. results are collected in Tables 2, 3 and 4.

Note that by using an exact linear program, we are able to obtain an exact optimal design up to dimension $d = 8$ in less than 10 minutes. Moreover, using an approximate algorithm such as GIPPR or GAC, we are able to create designs up to $d = 50$ dimensions.

	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$	$d=15$	$d=20$	$d=30$	$d=40$	$d=50$
$n=3$	0.1361	0.1427	0.1356	0.1236	0.1376	0.1530	0.1530	0.1376	0.137	0.127	0.1346
$n=4$	0.1659	0.1256	0.1396	0.2107	0.1538	0.1371	0.1778	0.1547	0.187	0.125	0.2103
$n=5$	0.2032	0.3478	0.1928	0.2849	0.5785	0.3081	1.3928	1.0887	1.081	1.731	1.5282
$n=6$	0.2336	0.4767	0.1587	0.3625	1.7036	0.9732	1.6523	0.8137	9.951	3.683	3.8064
$n=7$	0.4571	0.6411	0.3111	0.9800	-	-	-	-	-	-	-
$n=8$	3.1896	4.162	0.5724	1.9579	-	-	-	-	-	-	-
$n=9$	160.3	230.2	178.2	-	-	-	-	-	-	-	-
$n=10$	530.2	580.5	479.3	-	-	-	-	-	-	-	-
$n=15$	-	-	-	-	-	-	-	-	-	-	-
$n=20$	-	-	-	-	-	-	-	-	-	-	-
$n=30$	-	-	-	-	-	-	-	-	-	-	-
$n=40$	x	-	-	-	-	-	-	-	-	-	-
$n=50$	x	-	-	-	-	-	-	-	-	-	-
$n=75$	x	x	-	-	-	-	-	-	-	-	-
$n=100$	x	x	-	-	-	-	-	-	-	-	-

Table 2. Time (s) to generate an exact optimal design maximizing the packing radius using the linear programming formulation with different values of n and d. x denotes the case where $n > |\mathcal{X}|$ and $-$ denotes the case where the optimization did not terminate before 10 minutes.

	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$	$d=15$	$d=20$	$d=30$	$d=40$	$d=50$
$n=3$	0.016	0.018	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019
$n=4$	0.016	0.018	0.019	0.019	0.019	0.019	0.0249	0.026	0.028	0.027	0.033
$n=5$	0.016	0.018	0.019	0.019	0.019	0.019	0.045	0.034	0.039	0.050	0.082
$n=6$	0.016	0.018	0.019	0.019	0.019	0.019	0.0551	0.045	0.055	0.073	0.129
$n=7$	0.016	0.018	0.020	0.020	0.020	0.020	0.0652	0.0594	0.070	0.097	0.166
$n=8$	0.016	0.018	0.020	0.020	0.020	0.020	0.0770	0.0690	0.0843	0.118	0.208
$n=9$	0.016	0.018	0.020	0.020	0.020	0.020	0.0916	0.0928	0.099	0.145	0.246
$n=10$	0.016	0.018	0.020	0.020	0.020	0.020	0.1016	0.1023	0.117	0.169	0.286
$n=15$	0.017	0.019	0.020	0.020	0.021	0.021	0.1898	0.2773	0.220	0.304	0.474
$n=20$	0.017	0.019	0.020	0.022	0.022	0.022	0.3178	0.4725	0.335	0.476	0.673
$n=30$	0.017	0.019	0.021	0.023	0.024	0.031	0.497	3.8640	0.797	1.111	1.276
$n=40$	x	0.020	0.022	0.026	0.028	0.040	0.833	7.2363	3.255	11.710	9.526
$n=50$	x	0.020	0.023	0.027	0.034	0.052	1.223	11.10	21.67	30.56	41.14
$n=75$	x	x	0.032	0.035	0.049	0.095	2.627	48.25	219.5	269.5	512.3
$n=100$	x	x	0.038	0.048	0.070	0.15	4.55	84.05	534.7	712.5	1122

Table 3. Time (s) to generate a design with the GIPPR algorithm with different values of n and d. x denotes the case where $n > |\mathcal{X}|$.

	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$	$d=15$	$d=20$	$d=30$	$d=40$	$d=50$
$n=3$	0.005	0.012	0.022	0.057	0.1539	0.3361	5.910	10.59	16.87	54.77	77.08
$n=4$	0.009	0.018	0.026	0.072	0.2055	0.3481	5.927	11.48	18.51	58.63	86.19
$n=5$	0.007	0.029	0.036	0.092	0.2049	0.4817	7.136	12.82	21.27	63.88	100.4
$n=6$	0.009	0.029	0.053	0.093	0.2933	0.4635	6.668	13.12	23.70	66.57	118.9
$n=7$	0.015	0.029	0.049	0.105	0.2863	0.4801	7.175	14.46	29.85	76.05	137.9
$n=8$	0.013	0.029	0.050	0.091	0.2555	0.5851	6.684	16.77	25.92	77.79	160.3
$n=9$	0.015	0.031	0.066	0.097	0.2802	0.7503	7.011	17.12	26.85	92.44	187.4
$n=10$	0.014	0.031	0.071	0.113	0.3518	0.7731	8.104	19.07	27.96	97.97	182.8
$n=15$	0.021	0.042	0.069	0.131	0.4771	0.9520	8.567	22.41	37.97	151.7	280.0
$n=20$	0.028	0.048	0.087	0.158	0.4537	1.0901	8.423	29.99	50.92	168.7	382.9
$n=30$	0.029	0.064	0.139	0.168	0.5658	1.3500	8.625	37.25	79.35	222.5	484.2
$n=40$	x	0.083	0.190	0.208	0.6566	1.5073	9.151	41.92	94.47	314.3	583.3
$n=50$	x	0.090	0.219	0.341	0.7371	1.8781	11.87	54.18	106.7	356.6	686.4
$n=75$	x	x	0.347	0.858	1.0122	2.3584	12.24	67.78	146.3	514.4	937.1
$n=100$	x	x	0.383	1.404	2.5266	2.8363	12.43	87.91	182.5	682.7	1193

Table 4. Time (s) to generate designs with the GAC algorithm for different values of n and d. x denotes the case where $n > |\mathcal{X}|$.

D.2. Extension to non-binary variables and different cardinality

The derivation of the GAC algorithm to non-binary is straightforward since it only involves the direct computation of the distances $d_H(x, x_i)$ for $x_i \in D_n$. However, to adapt both the exact linear program solvers of Section 3 and the GIPPR algorithm of Section 4, the main bottleneck to adapt the linear programming formulation lies in the linearization of the standard XOR formulation to non-binary variables. We now provide the derivation. Consider any variables x and y that can take $D \geq 3$ different categories, i.e. $x \in \{1, \ldots, D\}$ and $y \in \{1, \ldots, D\}$. Then, $z = \mathbb{I}\{x \neq y\}$ is the solution of the

following linear equations:

$$-Dz \leq x - y \leq Dz$$

$$z - (D+1)\delta \leq x - y \leq -z + (D+1)(1-\delta)$$

where $\delta \in \{0, 1\}$ is an additional extra binary variable. Thus, one can directly adapt all the linear formulations (GRIPPR and exact solvers) using this trick, at the price of two additional constraints and one extra variable per XOR operator.

As an example, the figure below (Figure 6) reports the same results as Figure 5 withcategorical variables with cardinality 3 (Top) and 4 (Bottom) when $d = 5$.

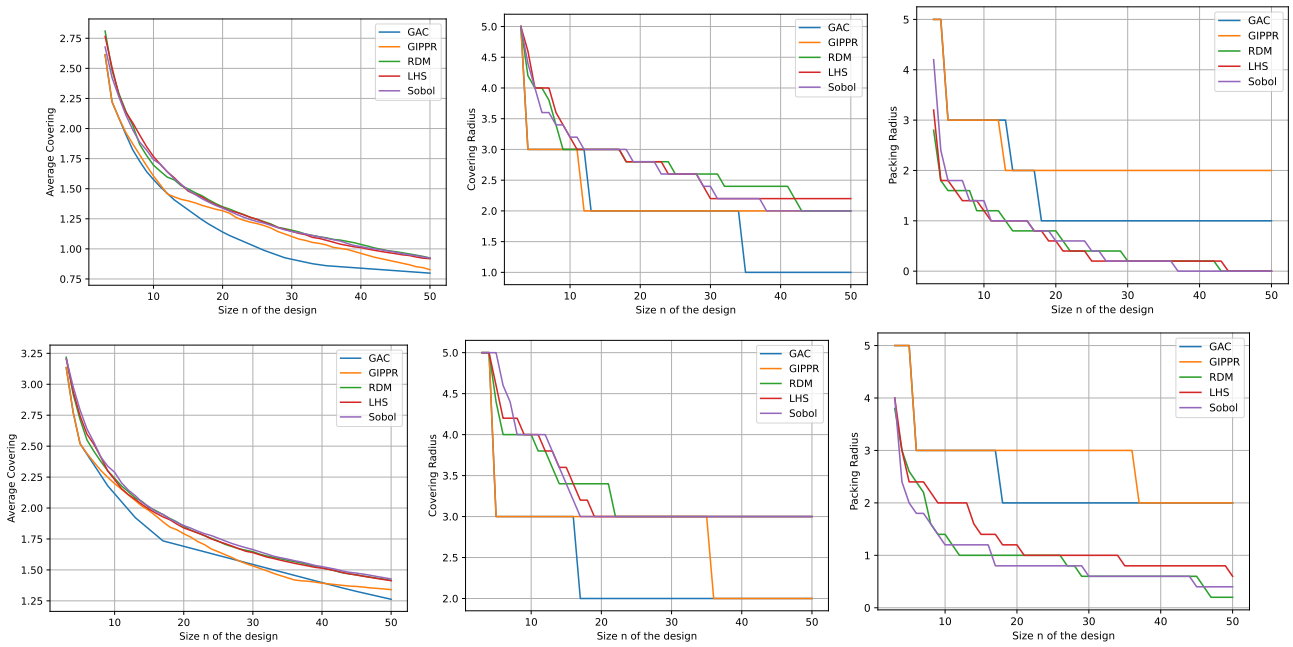


Figure 6. The graph displays the evolution of the diversity measures $\texttt{PR}(D_n)$, $\texttt{CR}(D_n)$ and $\texttt{AC}(D_n)$ for different design sizes $n \in \{2, \ldots, 50\}$. The bold line represents the average value computed over 10 runs. The top line considers the case when we have 3 categories and the bottom line considers the case when we have 4 categories.

D.3. Addition of constraints

We point out that additional constraints can be embedded in the diversity program as long as they can be cast into a standard linear programming form. As an example, for the constraint $d(x, \vec{0}) \leq K$ and the ball constraint $d(x, x^*) \leq K$ for some K, one can respectively cast them as $\sum_{i=1}^{d} x_i \leq K$ and $\sum_{i=1}^{d} x_i(1 - x_i^*) + (1 - x_i)x_i^* \leq K$ and we can directly add them to the Linear Program described in Prop 3.4 and 4.1.

D.4. Comparison to optimal designs

Due to the size of the problem, we only compare the quality of the empirical designs to the optimal solutions for very small problems. The next table (D.4) shows the different values of approximate values compared to the optimal one.

n	1	2	3	4	5	6	7	8
$\texttt{CR}^\star$	3	1	1	1	1	1	1	0
$\texttt{PR}^\star$		3	2	2	1	1	1	1
CR GIPPR	3	1	1	1	1	1	1	0
PR GIPPR		3	2	1	1	1	1	1

Table 5. Comparison of the covering and packing radius of approximate methods compared to the optimal one for a design with $d = 3$.

D.5. Details of the numerical experiments

All the experiments were performed on a desktop computer with Intel i7 CPU @ 1.80GHz 1.99 GHz with 16GB of RAM. First, we described the algorithms used in the benchmark and then describe the test problems.

Random designs. A random design $D_n = (x_1, \dots, x_n)$ of size $n \geq 1$ defined on $\mathcal{X}$, simply consists of a collection of n points $x_1, \dots, x_n \sim \mathcal{U}(\mathcal{X})$ uniformly and independently distributed over $\mathcal{X}$.

k-DPP. For the implementation of k-DPP, we used the implementation of (Gautier et al., 2019) with a kernel set to $k(x, x') = \exp(-d_H(x, x'))$ to be as close as possible to our setting.

d-Halton. For the d-Halton, we used a design $D_n = (x_1, \dots, x_n)$ of $n \geq 1$ points generated according to the standard Halton sequence (Halton & Smith, 1964) from the scipy QMC library (Virtanen et al., 2020) over $\mathcal{X} = [0, 1]^d$. Then, we simply discretize the design using the 1/2 threshold, i.e¿., $D_n = (\mathbb{I}\{x_1 > 1/2\}, \dots, \mathbb{I}\{x_n > 1/2\})$.

GIPPR. For the GIPPR algorithm, we directly used the implementation provided in Algorithm 2 with the open-source SCIP solver. We ran the algorithm at least up to $d = 50$ in our experiments.

GAC. For the GAC algorithm, we used the implementation provided in Algorithm 7 below. For this implementation, first we compute the pairwise distance Q between all the points of $\mathcal{X} = (x_1, \dots, x_{|\mathcal{X}|})$ with $x_i = \text{binary}_d(i)$ with $i \in \{1, \dots, 2^d\}$ and we keep updated the distance of the points of the domain to the current set in C. Then, at each time step, the line 5 performs the computation of the vectorized minimum of each element i of the design using the fact that $\min(Q_i, C) = (Q_i + C - |Q_i - C|)/2$ and we take the summation (Sum) over the columns of the matrix to compute the average covering. The argmin then selects the point which greedily minimize the average covering. Finally, we update the novel distance vector C and we add the novel point i_t. In practice, the empirical computation cost of this algorithm comes from the computation of Q which is $|\mathcal{X}|^2$. In practice, we were able to run this algorithm up to $d = 20$ using int8 for the elements of the matrix which can be further improved using sparse matrix. For larger dimensionalities, one can employ sub-sampling techniques over the full space which are detailed below and have some guarantees:

- Using the stochastic approximation of Proposition A.2 to compute $\widehat{AC}_M(D)$. This approach simply consists in sub-sampling the space $\mathcal{X}$ with M points sampled uniformly over the input space in order to approximate the computation of $\texttt{AC}(D_n)$ in Algorithm 5. Using this approximation, we have a complexity of at most $O(n|\mathcal{X}|Md)$ to run GAC, and by using arguments as in the proof of Theorem 4.7, one can easily show that with probability at least $1 - \delta$:

$$\frac{\texttt{AC}_1^* - \texttt{AC}_n^*}{2} \leq \texttt{AC}_1^* - \widehat{AC}_M(D_n) + \varepsilon$$

  where $\varepsilon = 2d\sqrt{\ln(2/\delta)/2M}$. Thus, taking $M = O(d^2)$ allows to have a small error regardless of the dimension.

- Using stochastic greedy evaluations as in (Mirzasoleiman et al., 2015). It consists in sub-sampling M points $\widehat{\mathcal{X}}_M = \{X_1, \dots, X_M\}$ to compute the $\arg\min_{x \in \widehat{\mathcal{X}}_M}$ of line 3 of GAC (Algorithm 5) instead of computing the minimum $\arg\min_{x \in \mathcal{X}}$ on the full space $\mathcal{X}$. For this algorithm, it is known that, in expectation, we have an approximation error at least of order $(1/2 - e^{-nM/|\mathcal{X}|})$ instead of the standard $1/2$. We refer to (Mirzasoleiman et al., 2015) for more details on the derivation of this result.

For larger dimensions ($d > 20$), we thus used the previous statements using the same algorithm as described in Algorithm 7 with a sub-sample of $\widehat{\mathcal{X}}_M$ of $M = 10d^2$ examples instead of the full space $\mathcal{X}$. Last, we also point out that the popular method of (Minoux, 1978) could also be used to speed up the computations. Second, we detail the empirical protocol used for each task.

Diversity. For the diversity task, we simply computer the metrics $\texttt{AC}(D_n)$, $\texttt{CR}(D_n)$ and $\texttt{PR}(D_n)$ for each design D_n provided by the methods above with $n \in \{1, \dots, 50\}$ and $d \in \{7, 8\}$. To have uncertainty measures, we repeated the experiments 100 times when the designs D_n are random.

Numerical Integration. For the numerical integration task, we recorded the empirical average $\widehat{F}_n(D_n) = (1/|D_n|) \sum_{x \in D_n} f(x)$ over two functions f commonly encountered in the genetic optimization literature (Doerr et al., 2019) for which the ground truth $\mathbb{E}_{X \in \mathcal{U}(X)}[f(X)]$ is known:

- **OneMax.** The OneMax is probably the best-studied benchmark problem in the context of discrete optimization. It is described by the function $f(x) = \sum_{i=1}^{d} \mathbb{I}\{x_i = 1\}$, which computes the number of ones in a vector. The problem has

Algorithm 7 GAC with fast updates

Require: Dimensionality $d \geq 1$ of the categorical space, number $n \geq 2$ of design points

$D_0 = \emptyset$
Compute the matrix $Q \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ where $Q_{i,j} = d_H(x_i, x_j)$
$C \leftarrow [d, \ldots, d] \in \mathbb{R}^d$
for $t = 1, ..., n-1$ **do**
    Get any point that greedily minimizes the average covering:

$$i_t \leftarrow \arg\min_{i=1\ldots n} \text{Sum}[Q + C - |Q - C|]/2$$

    Update the distance vector $C \leftarrow (Q_{i^*} + C - |Q_{i^*} - C|)/2$
    $D_t \leftarrow D_{t-1} \cup \{x_{i_t}\}$
end for
Return the design D_n

a very smooth and non-deceptive objective landscape. Note that since it puts equal weights on all the dimensions, if the expectation of an estimate is equal to $d/2$ then, the estimate evenly covers each dimension of the space.

- **Harmonic.** $f(x) = \sum_{i=1}^{d} i^2 \mathbb{I}\{x_i = 1\}$ which puts harmonic weights on the dimensions with $d = 10$. Note that each weight on each dimension is different which makes it more challenging than the OneMax version.

For each test problem f and each design D_n, we computer the empirical average $\widehat{F}_n(D_n)$ with a size $n \in \{1, \ldots, 50\}$ and with dimension $d = 10$. We repeated each experiment 100 times to have an uncertainty measure.

Data Collection. We detail here the full setup for the experimental design experiments. For the target objective f, we used the following problems:

- **Ising.** The Ising Spin Glass model arose in solid-state physics and statistical mechanics, aiming to describe simple interactions within many-particle systems. The classical Ising model considers a set of spins placed on a regular lattice, where each edge $< i, j >$ is associated with an interaction strength $J_{i,j}$. In essence, a problem instance is defined upon setting up the coupling matrix $J_{i,j}$. Each spin directs up or down, associated with a value ± 1, and a set of d spin glasses is said to form a configuration, denoted as $S = (s_1, \ldots, s_d)$. The configuration's energy function is described by the system's Hamiltonian, as a quadratic function of those d spin variables: $-\sum_{i<j} J_{i,j} s_i s_j - \sum_{i=1}^{d} h_i s_i$ where h_i is an external magnetic field. The problem of interest is the study of the minimal energy configurations, which are termed ground states, on a final lattice. The implementation of the Ising model of (Doerr et al., 2020) was taken, assuming zero external magnetic fields, and applying periodic boundary conditions (PBC). To formally define the function, we adopt a strict graph perspective, where $G = (V, E)$ is undirected and $|V| = d$. We apply an affine transformation $\{-1, +1\} \to \{0, 1\}$ where the d spins become binary decision variables. A generalized, compact form for the quadratic objective function is written as follows:

$$f(x) = \sum_{(u,v)\in E} [x_u x_v - (1 - x_u)(1 - x_v)]$$

  where the graph G is defined as follows $e_{i,j} = 1$ if and only if $j = i + 1$ for all $i \in \{1, \ldots, d-1\}$ or $j = d$ and $i = 1$. For this problem, we took the implementation of (Doerr et al., 2020).

- **MIS.** Given a graph $G = (V, E)$ with $V = \{1, \ldots, |V|\}$, a maximum independent vertex set (which generally is not equivalent to a maximal independent vertex set) is a subset of vertices where no two vertices are direct neighbors. A maximum independent vertex set (MIS) is defined as an independent vertex set V' having the largest possible size. Using the standard binary encoding $V' = \{i = 1 \ldots d \mid x_i = 1\}$, MIS can be formulated as follows:

$$f(x) = \sum_{i=1}^{d} x_i - d \sum_{i,j} x_i x_j e_{i,j}.$$

where $e_{i,j} = 1$ if $(i,j) \in E$ and 0 otherwise. In particular, following (Back & Khuri, 1994), a specific, scalable problem instance was considered and defined as follows:

$$\begin{aligned} e_{i,j} = 1 \Leftrightarrow & j = i + 1 \; \forall i \in \{1, \dots d\} - \{d/2\} \\ & \text{or } j = i + d/2 + 1 \; \forall i \in \{1, \dots, d/2 - 1\} \\ & \text{or } j = i + d/2 - 1 \; \forall i \in \{2, \dots, d/2\}. \end{aligned}$$

For this problem, we took the implementation from (Doerr et al., 2020). Here, the problem is given a subset of graphs and their MIS score, predicting the scores of other MIS.

- **LABS.** Obtaining binary sequences possessing a high merit factor, also known as the Low Autocorrelation Binary Sequence (LABS) problem, constitutes a grand combinatorial challenge with practical applications in radar engineering and measurements. It poses a non-linear system over a binary sequence space, with the goal to model the reciprocal of the sequence's autocorrelation:

$$f(x) = \frac{d^2}{E(x)} \text{ where } E(x) = \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} s_i \times s_{i+k} \right)^2$$

with s_i set to $s_i = 2x_i - 1$ to cast the problem in $\{-1, 1\}^d$ when $x_i \in \{0, 1\}^d$. For this problem, we took the implementation of (Doerr et al., 2020). Here the problem consists of collecting a dataset of sequences and their associated autocorrelations to predict the autocorrelations of other sequences.

Training. For each problem f, we can get the values $f(x)$ over the whole domain $x \in \mathcal{X}$. For the gaussian process model, we used a standard ARD kernel used in (Wan et al., 2021) defined by:

$$k_\theta(x, x') = l \times \exp\left(-\sum_{i=1}^{d} \sigma_i \mathbb{I}\{x_i \neq x'_i\} \right)$$

where $\theta = (l, \sigma_1, \dots, \sigma_d)$ are $d + 1$ learnable parameters. When we have a series of evaluations $(x_1, y_1), \dots, (x_n, y_n)$, the parameters are learned by minimizing the log-likelihood of the data, which is proportional to

$$L(\theta | (x_1, y_1), \dots, (x_n, y_n)) \propto -\frac{1}{2} \log |K_{\theta,n}| + y_n^T K_{\theta,n}^{-1} y$$

where $K_{\theta,n}$ is the $n \times n$ matrix where each entry (i,j) is equal to $k_\theta(x_i, x_j)$. In practice, we learned the parameters $\hat{\theta}_n$ solving:

$$\hat{\theta}_n \in \arg\max_{\theta \in \mathbb{R}^{d+1}} L(\theta | (x_1, y_1), \dots, (x_n, y_n))$$

using the Adam optimizer with 100 steps and a learning rate set to 10^{-3}, where $(x_1, y_1), \dots, (x_n, y_n)$ denotes the labeled data set.

Evaluation. Finally, once the parameters $\hat{\theta}_n$ are learned, we computed the log-likelihood over the whole space $\mathcal{X}$ as well as the mean square error:

$$\text{MSE}(\hat{f}_{\theta_n}, \mathcal{X}) = (1/|\mathcal{X}|) \sum_{x \in \mathcal{X}} (\hat{f}_{\theta_n}(x) - f(x))^2$$

Parameter tuning. We detail here the full setup for the parameter tuning experiments.

Graph Neural Network tuning. For the second set of experiments, we considered the problem of tuning the architecture of a standard three-layers graph neural network displayed in Figure D.5. The task here consists in choosing for each layer between two activation functions $A \in \{\text{ReLU}, \text{TanH}\}$, to put a dropout of value 0.3 or not and finally to activate the skip-connection or not for each layer. It results in a total of nine binary variables. To record the quality of a set of configuration parameters $x \in \{0, 1\}^9$, we record the average gain over each problem instance:

$$\text{SCORE}(x) = \frac{\text{Test}(\vec{0}) - \text{Test}(x)}{\text{Test}(\vec{0})}$$

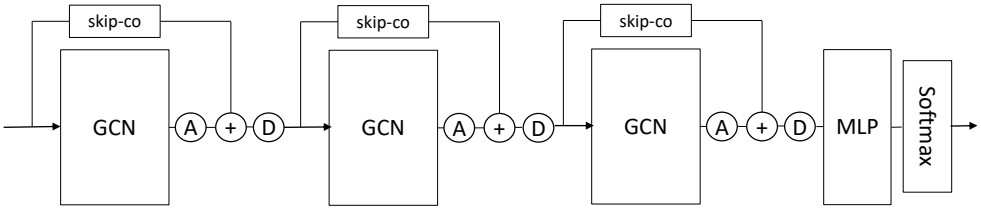


Figure 7. Architecture of the graph neural network. A denotes the activation function, D denotes a dropout and skip-co denotes a skip connection.

where Test denotes the value on the classification error on the test set of the network learned after $T = 50$ epochs, with the Adam optimizer with $1e-3$ learning rate and a batch size of the size of the training set. We used two datasets to perform the experiments, Cora and CiteSeer which are encountered in most graph neural nets tasks. For each class of problem and each algorithm, we recorded the best score using a budget of $n = 20$ different configurations, defined as follows:

$$\max(\text{SCORE}(x_1), \ldots, \text{SCORE}(x_n))$$

where $D_n = (x_1, \ldots, x_n)$ denotes a design. Moreover, for each design class (load balancing, item placement and anonymous), we reproduced the results $M = 10$ times to have an estimation of the average gain as well as the variation.

SCIP tuning. The third task of the NeurIPS 2021 ML4CO competition (Configuration taks) consisted in finding the hyperparameters of a MILP solver (here the SCIP solver) that provide the best results over a given class of problem instances: item placement and load balancing. Here the combinatorial solver SCIP is parameterized by a collection of a short-listed set of 25 hyperparameters. Among them, there are nine binary categorical parameters: branching/scorefunc, branching/preferbinary, branching/lpgainnormalize, lp/pricing, nodeselection/childsel, separating/maxcuts and branching/scorefac. For each class of problem (item, load, anonymous), we consider 10 problems in each class denoted here by $f_1, \ldots, f_{10}$ and we denote by $f_i(x)$ the result of the SCIP solver over the problem f_i tuned with the categorical parameters x. Here, $f_i(x)$ corresponds to the primal-dual gap of the solver when the solver is parameterized with x and we denote by $\bar{f}_i$ the primal-dual gap of the solver with default parameters. To record the quality of a set of parameters x, we record the average gain over each problem instance:

$$\text{SCORE}(x) = \frac{1}{10} \sum_{i=1}^{10} \frac{f_i(x) - f_i(\vec{0})}{f_i(\vec{0})}$$

For each class of problem and each algorithm, we recorded the best score using a budget of $n = 20$ different configurations, defined as follows:

$$\max(\text{SCORE}(x_1), \ldots, \text{SCORE}(x_n))$$

where $D_n = (x_1, \ldots, x_n)$. Moreover, for each design class (load balancing, item placement and anonymous), we reproduced the results $M = 10$ times to have an estimation of the average gain as well as the variation.