Hypercubes Latins maximin pour l'echantillonage de systèmes complexes

Kaourintin Le Guiban

NNT : 2018SACLC008

Thèse de doctorat de l'Université Paris-Saclay préparée à CentraleSupelec

Ecole doctorale n°580
Sciences et Technologies de l'Information et de la Communication
Spécialité de doctorat: Informatique

par

M. Kaourintin LE GUIBAN

Hypercubes Latins maximin pour l'echantillonage de systèmes complexes

Thèse présentée et soutenue à Gif-sur-Yvette, le 24 janvier 2018.

Composition du Jury :

M.	Jarosław Byrka	Docent Uniwersytet Wrocławski, Pologne	(Rapporteur)
M.	Tristan Cazenave	Professeur Université Paris-Dauphine	(Rapporteur)
Mme	Cristina Bazgan	Professeur Université Paris-Dauphine	(Président du jury)
M.	Yannis Manoussakis	Professeur Université Paris-Sud	(Examinateur)
Mme	Joanna Tomasik	Professeur LRI/CentraleSupélec	(Directeur de thèse)
M.	Arpad Rimmel	Professeur assistant LRI/CentraleSupélec	(Encadrant)
M.	Marc-Antoine Weisser	Professeur assistant LRI/CentraleSupélec	(Encadrant)

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title : Maximin Latin hypercubes for experimental design

Keywords : Latin Hypercube Design, NP-completeness, approximation algorithm, Simulated Annealing

Abstract : A Latin Hypercube Design (LHD) is a set of n points in dimension k with integer coordinates contained in a hypercube of size n^k, such that its points do not share a coordinate on any dimension. In maximin LHDs the separation distance, *i.e.* the minimal distance between two points, is maximal. Maximin LHDs are widely used in metamodeling thanks to their space filling and non-collapsing properties which make them appropriate for sampling. As most work concerning LHDs focused on heuristic algorithms to produce them, we decided to make a detailed study of this problem, including its complexity, approximability, and the design of practical heuristic algorithms.

To conduct this study, we generalized the maximin LHD construction problem by defining the maximin partial Latin Hypercube completion problem: given a partial LHD (an LHD with missing points), complete it with the maximum separation distance possible. The subproblem where the partial LHD is initially empty corresponds to the classical LHD construction problem.

We studied the complexity of the completion problem and proved its NP-completeness for all norms in dimensions $k \geq 3$, and for usual norms (*i.e.* norms $\mathcal{L}_p$, with $p \in \mathbb{N}$ and norm $\mathcal{L}_\infty$) on the plane. As we did not determine the complexity of the subproblem, we searched for performance guarantees of algorithms which may be designed for both problems.

On the one hand, we found that the completion problem is inapproximable for all norms in dimensions $k \geq 3$. We also gave a weaker inapproximation result for norm $\mathcal{L}_\infty$ in dimension $k = 2$. On the other hand, we designed an approximation algorithm for the construction problem which we proved using two new upper bounds we introduced.

Besides the theoretical aspect of this study, we worked on heuristic algorithms adapted for these problems, focusing primarily on the Simulated Annealing metaheuristic. We proposed a new evaluation function for the construction problem and new mutations for both the construction and completion problems, improving the results found in the literature. We observed that the behaviour of the completion problem changed depending on the number of points in the initial pLHD, calling for the use of different mutations. Taking advantage of this fact, we enriched the Simulated Annealing algorithm by using a bandit method to choose the most appropriate mutation on the fly, outperforming both mutations for intermediate number of points preset.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Hypercubes Latins maximin pour l'echantillonage de systèmes complexes

Keywords : Hypercube latins maximin, NP-complet, algorithme d'approximation, recuit simulé

Résumé : Un hypercube latin (LHD) est un ensemble de n points en dimension k, à coordonnées entières, contenus dans un hypercube de taille n^k, et tel que les points ne partagent pas de coordonnées sur aucune dimension. Un LHD maximin est un LHD tel que la distance de séparation, c'est-à-dire la distance minimale entre deux points, est maximale. Les LHDs maximin sont particulièrement utilisés pour la construction de métamodèles en raison de leurs bonnes propriétés pour l'échantillonnage. Comme la plus grande partie des travaux concernant les LHD se sont concentrés sur leur construction par des algorithmes heuristiques, nous avons décidé de produire une étude détaillée du problème, et en particulier de sa complexité et de son approximabilité en plus des algorithmes heuristiques permettant de le résoudre en pratique.

Pour conduire cette étude, nous avons généralisé le problème de construction d'un LHD maximin en définissant le problème de compléter un hypercube latin entamé en respectant la contrainte maximin: étant donné un LHD partiel (un LHD auquel il manque des points), lui ajouter des points de manière à obtenir un LHD avec une distance de séparation maximum. Le sous-problème dans lequel le LHD partiel est vide correspond au problème de construction de LHD classique.

Nous avons étudié la complexité du problème de complétion et avons prouvé qu'il est NP-complet pour toutes les normes en dimension $k \geq 3$, et pour les normes usuelles (*i.e.* les normes $\mathcal{L}_p$, avec $p \in \mathbb{N}$ et la norme $\mathcal{L}_\infty$) dans le plan. N'ayant pas déterminé la complexité du sous-problème, nous avons cherché des garanties de performances pour les algorithmes résolvant les deux problèmes.

D'un côté, nous avons prouvé que le problème de complétion n'est approximable pour aucune norme en dimensions $k \geq 3$. Nous avons également prouvé un résultat d'inapproximabilité plus faible pour la norme $\mathcal{L}_\infty$ en dimension $k = 2$. D'un autre côté, nous avons proposé un algorithme d'approximation pour le problème de construction, et avons calculé le rapport d'approximation grâce à deux bornes supérieures que nous avons établies.

En plus de l'aspect théorique de cette étude, nous avons travaillé sur les algorithmes heuristiques, et en particulier sur la méta-heuristique du recuit simulé. Nous avons proposé une nouvelle fonction d'évaluation pour le problème de construction et de nouvelles mutations pour les deux problèmes, permettant d'améliorer les résultats rapportés dans la littérature. Nous avons observé que le comportement du problème de complétion change en fonction du nombre de points initialement présent dans le LHD partiel, faisant varier la mutation la plus adaptée au problème. Nous avons pris ce fait en compte et amélioré le recuit simulé en utilisant une méthode de bandit pour choisir la mutation la plus appropriée pendant le déroulement de l'algorithme, dépassant chaque mutation individuelle lorsque le nombre de points présents initialement est intermédiaire.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Contents

1 Introduction **11**

2 Designs **15**
2.1 Metamodeling . . . 15
2.1.1 Metamodeling techniques . . . 15
Polynomial regression . . . 15
Response surface . . . 16
Kriging metamodels . . . 16
Radial basis function . . . 17
Conclusion . . . 17
2.1.2 Sampling techniques . . . 17
Fractional factorial designs . . . 17
Orthogonal Array . . . 18
Latin Hypercube Designs . . . 18
2.2 Latin Hypercube Designs problems . . . 18
2.3 Conclusion . . . 19

3 Problem complexity analysis **21**
3.1 State of the art . . . 21
3.1.1 The LHD-CP . . . 21
3.1.2 Problems related to the pLHD-CP . . . 22
3.2 Complexity of the pLHD-CP . . . 23
3.2.1 Completion of a partial LHD with forbidden coordinates . . . 23
3.2.2 Complexity of the pLHD-CP verifier . . . 27
3.2.3 Definitions concerning completion problems . . . 28
3.2.4 NP-completeness in dimensions $k \geq 3$ . . . 28
Results for norm $\mathcal{L}_1$ . . . 28
3.2.5 Dimension $k = 2$ for norms $\mathcal{L}_1$ and $\mathcal{L}_p$ . . . 32
NP-completeness for norm $\mathcal{L}_1$ . . . 32
Results for any norm $\mathcal{L}_p$ . . . 35
3.2.6 Dimension $k = 2$ for norm $\mathcal{L}_\infty$ . . . 38
NP-completeness for norm L_∞ . . . 38
3.3 Conclusion . . . 43

4 Guarantees of performance for algorithms **47**
4.1 State of the art . . . 47
4.2 Completion . . . 48
4.2.1 Dimension $k \geq 3$ . . . 48
4.2.2 Dimension $k = 2$, norm $\mathcal{L}_\infty$ . . . 49
4.3 Construction . . . 51
4.3.1 Bounds . . . 51
Bound for large-size LHD. . . . 52

Upper bound for LHD of any size. . . . 52
Comparison of the bounds. . . . 53
4.3.2 IES algorithm . . . 54
Description of the IES algorithm . . . 54
Complexity of the algorithm . . . 55
4.3.3 Separation distance . . . 55
4.3.4 Approximation ratio . . . 62
Using the bound for large-size LHD. . . . 62
Using the bound for LHD of any size. . . . 63
Using van Dam's bound. . . . 63
4.3.5 Comparison of the approximation ratios . . . 63
4.4 Generalized algorithms . . . 64
4.4.1 Fixed-layer extension . . . 64
4.4.2 Adapted Layers . . . 65
4.4.3 Computational results . . . 66
Algorithm used to solve the LHD-CP . . . 67
Experimental settings . . . 67
Results of IES extensions . . . 67
4.4.4 Comparison with Simulated Annealing . . . 67
4.5 Conclusion . . . 70

5 Heuristic algorithms **73**
5.1 State of the art . . . 73
5.1.1 Algorithm components . . . 73
Evaluation functions . . . 73
Mutations . . . 74
5.1.2 Heuristic algorithms . . . 75
Genetic Algorithms . . . 75
Iterated Local Search . . . 75
Translational Propagation . . . 76
Symmetric LHDs . . . 76
Periodic LHDs . . . 77
Simulated Annealing . . . 77
5.1.3 Conclusion . . . 77
5.2 Simulated Annealing for the LHD-CP . . . 78
5.2.1 New mutation for the LHD-CP . . . 78
Principle of the mutation . . . 78
Effect of the mutation . . . 79
Performance Evaluation . . . 79
Conclusion . . . 79
5.2.2 New evaluation function targeting Maximin . . . 80
Presentation of a Maximin effect: narrowing the distribution of distances 80
Definition of evaluation function ψ . . . 82
5.2.3 Tuning of parameter σ and its justification . . . 82
5.2.4 Experimental results . . . 84
5.2.5 Conclusion . . . 84
5.3 SA for completion . . . 84
5.3.1 Mutation targeting "relatively empty" hypercubes . . . 85
5.3.2 Bandit-driven mutation . . . 87
5.3.3 Numerical experiments . . . 88
5.4 Conclusion . . . 90

6 Conclusion **91**
6.1 Summary of contributions 91
6.2 Directions for further research 91

Bibliography **93**

Symbols and abbreviations

- $\varnothing$: the empty set.
- $\mathbb{N}$: the set of all natural numbers.
- $\mathbb{N}^*$: the set of all positive natural numbers.
- LHD: Latin Hypercube Design.
- pLHD: partial Latin Hypercube Design.
- $[\![a, b]\!]$: the set of all integers n such that $a \leq n \leq b$.
- $\mathcal{D}_n^k$: the set of all LHDs of size n and dimension k.
- $(p^{(1)}, p^{(2)}, ..., p^{(m)}, ..., p^{(k)})$: the coordinates of point p in a k-dimensional space.
- $||e||$: the length of vector e according to a given norm.
- $\delta(p_1, p_2)$: the distance between points p_1 and p_2 according to a given norm.
- $\Delta_{\mathcal{L}}(D)$: the separation distance of LHD D for a norm $\mathcal{L}$.
- $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\infty$: the Manhattan, Euclidean and Tchebychev norms.
- $\mathcal{L}_p$, with $p \in \mathbb{N}$: the norms such that $||(x^{(1)}, ..., x^{(k)}|| = \sqrt[p]{\sum_{i=1}^{k}(x^{(i)})^p}$.

Problems studied in this thesis

Name	Abbreviation	Definition		Complexity		Approximability		Algorithms	
Principal problems									
maximin Latin Hypercube Construction Problem	LHD-CP	Pr. 2.3	p. 19	No results		N/A		Sec. 4.3.2 [6], [22], [53], [60], [37], [51] Sec. 5.2	p. 54 p. 78
maximum maximin Latin Hypercube Construction Problem	max-LHD-CP	Pr. 2.4	p. 19	N/A		Sec. 4.3.4	p. 62	N/A	
partial Maximin Latin Hypercube Completion Problem	pLHD-CP	Pr. 2.1	p. 19	Th. 3.3 Th. 3.6 Th. 3.9 Th. 3.10 Th. 3.12 Th. 3.14	p. 27 p. 31 p. 32 p. 32 p. 37 p. 43	N/A		Sec. 5.3	p. 84
maximum partial Maximin Latin Hypercube Completion Problem	max-pLHD-CP	Pr. 2.2	p. 19	N/A		Th. 4.2 Th. 4.4	p. 48 p. 51	N/A	
Auxiliary problems									
partial Maximin Latin Hypercube Completion Problem with forbidden coordinates	pLHD-FC-CP	Pr. 3.4	p. 23	Th. 3.2 Th. 3.4 Th. 3.8 Th. 3.11 Th. 3.13	p. 27 p. 28 p. 31 p. 35 p. 38	N/A		Not concerned	
Maximum partial Maximin LHD completion problem with Forbidden Coordinates	max-pLHD-FC-CP	Pr. 3.5	p. 23	N/A		Th. 4.1 Th. 4.3	p. 48 p. 49	N/A	

Figure 1: Melancholia I, Albrecht Dürer, 1514. A magic square can be seen in the top right corner.

Chapter 1

Introduction

Complex physical systems often play a large part in engineering processes, and need to be optimized relatively to their parameters. We take as an example the conception of the front cradle of a car (the part supporting the engine of the car). The eigenfrequencies of the cradle need to be optimized to avoid the resonance phenomenon occurring due to the vibrations of the engine. As it would be impractical to build multiple systems to perform an optimization process, these systems are simulated. Such a simulation uses a model, which is a function y of the parameters $\mathrm{x} = (x_1, x_2, ..., x_k)$ of the system. In the example of the front cradle of the car, the eigenfrequencies are the output of the system, while the parameters are the density of the material used to build the cradle, its stiffness, and the stiffness of the connections to the rest of the car's frame. However, this function is usually unknown and simulations involve complex mathematical computations to produce the output, and thus necessitate huge amounts of computing time. Despite the advances in computing power, the increasing complexity of the simulations makes them impractical for performing an optimization process which needs thousands of simulation runs. To overcome this issue, metamodels, also called surrogate models, have been developed. A metamodel is a model of the simulation with a simple output, and thus fast to compute. It can be seen as a function $\hat{y}$ of the parameters $\mathrm{x} = (x_1, x_2, ..., x_k)$ of the system. The goal is to find a function $\hat{y}$ approximating y that can be computed fast. The metamodel is adapted to an optimization process which can then be performed without a prohibitive cost.

Several techniques exist to build a metamodel. All of them consist in a mathematical model with parameters to be fitted to the original model, by using a sample of the latter, called *design of computer experiments*. A design is a set of points, each representing the parameter values of one simulation run. The output of each simulation will be used to fit the metamodel to the original model. As the sample determines the parameters of the metamodel, it is critical to the metamodeling process. Two properties are needed for a design to be efficient. The first property is that the points of a design should be evenly spread to have a good coverage of the parameter space. This allows the metamodel to be accurate in all regions of this space. On the contrary, if a region of the parameter space does not contain points of the design, the metamodel is likely to be inaccurate in this region. Designs respecting this criterion are referred to as *space-filling designs*.

The second property is that the design points should not collapse. As evaluating the output of the simulation is costly, we want to evenly cover each parameter with as little evaluations as possible. As some parameter may not be meaningful, each parameter value should only be covered with one point to avoid meaningless simulations. This also allows one to diversify the values of parameters used as more points, and thus more costly simulations, are needed to cover more values if some values are covered more than once. Designs respecting this criterion are referred to as *non-collapsing designs*.

Maximin Latin Hypercube Designs (LHD) form a class of designs possessing both properties and largely used for sampling in metamodeling. However, building such designs can be

difficult, and furthermore, very few results exist on the algorithmic complexity of constructing maximin LHDs. To this end, we generalized this problem by considering partial LHD, which we define as LHDs with missing points. The problem then becomes completing a partial LHD while maximizing the minimal distance between any pair of points. From a sampling perspective, completing a partial LHD can be seen as completing a set of experiments already performed.

The goal of this thesis is twofold: on the one hand, study the algorithmic complexity of constructing maximin LHDs, and on the other hand, designing new algorithms for constructing LHDs and improve the performance of existing ones. This document is organized as follows:

In Chapter 2, we give an overview of metamodeling. We describe the most used metamodels: polynomial regression, the response surface methodology, Kriging metamodels, and radial basis functions. This survey allows us to highlight the need for sampling method of good quality which brings us to describe the most popular design techniques: fractional factorial designs, orthogonal arrays, and maximin Latin hypercube designs. While the latter are widely used due to their good properties, they are difficult to build and only few complexity results exist. For these reasons, we decide to study this problem from a complexity perspective. We formulate a generalization of this problem, the completion of partial LHDs, which are LHDs with missing points. We define the problem of constructing LHDs as a subproblem of the former.

Chapter 3 begins with a presentation of the existing results concerning the complexity of constructing LHDs, as well as a presentation of related problems. After this introduction, we perform an analysis of the complexity of the general problem, finding that it is NP-complete for norm $\mathcal{L}_1$ in dimensions $k \geq 3$. We are able to extend this result to all norms in dimensions $k \geq 3$. We then prove that it is NP-complete in the two-dimensional space for norm $\mathcal{L}_1$, norms $\mathcal{L}_p$ with $p \in \mathbb{N}$ and norm $\mathcal{L}_\infty$.

As the complexity of the subproblem was not found and the generalized problem is NP-complete, we search in Chapter 4 for algorithms with guarantees of performance. On the one hand, we show that the generalized problem is inapproximable for all norms in dimensions $k \geq 3$ and also find an upper bound for an approximation ratio in the two-dimensional space for norm $\mathcal{L}_\infty$. On the other hand, we describe an approximation algorithm for constructing LHDs and prove its approximation ratio by giving two new upper bounds for the problem.

Chapter 5 contains our achievements in solving the completion and construction problems with heuristic algorithms. We start it by describing the various algorithms that have been used to this effect. We expose genetic algorithms and iterated local search, two metaheuristic methods which have been adapted to construct maximin LHDs. We follow with heuristics specific to this problem, the translational propagation algorithm and periodic LHD algorithm. We also describe symmetric LHDs, a subclass of LHDs used for their good separation distance and the algorithm used to produce them. We then concentrate on the Simulated Annealing metaheuristic, which have been shown to be the most efficient one. After the state of the art, we present our contribution to this topic with improvements to the Simulated Annealing algorithm. We describe a new mutation which makes the search space smoother by reducing the effect of the mutation over the distances between points before showing an evaluation function which takes advantage of the shape of the distribution of distances in good LHDs. Both elements make the Simulated Annealing method construct LHDs more efficiently. We then adapt the Simulated Annealing algorithm to the completion problem. We find that the best performing mutation depends on the instances of the problem, and we design a hyper-heuristic inspired by bandits methods to choose the mutation, and over-performing each individual mutation in some cases.

Chapter 6 concludes our work by summarizing the contributions we made in this thesis, and giving directions for future research concerning maximin LHDs and the problems we defined. One of the objectives of this thesis has been attained: we designed an approximation algorithm for constructing maximin LHDs and improved metaheuristic algorithms by providing

a new evaluation function and a new mutation. While the second objective, finding the algorithmic complexity of constructing maximin LHD has not been entirely met, advances have been made by defining a more general problem, the completion problem, and proving its NP-completeness. Additionally, as an approximation algorithm has been found for constructing maximin LHDs, it is at least approximable.

Chapter 2

Designs

We give an overview of metamodeling, focusing on the most used metamodeling techniques and their sensitivity to sampling. We then describe the most used sampling techniques, paying attention to one in particular, maximin Latin hypercube designs. We finally formally define the problems of constructing and completing maximin LHDs.

2.1 Metamodeling

Computer simulations of a system often cost high amounts of resources, usually computing time [10] [23]. This is due to the fact that solving the underlying equations of the system may require time-consuming methods such as finite elements analysis [43]. When the goal is to optimize a system, numerous simulations have thus to be performed. To reduce their cost, a metamodel can be used [39] [33]. The metamodel is a mathematical model of the original simulation, designed to be faster to compute. It is capable of giving an approximate result of the simulation for any set of parameters in a much cheaper manner. Thus, it allows the optimization process to take place in a reasonable amount of time.
The first simulation can be seen as an output function $y(\mathrm{x})$, where $\mathrm{x} = (x_1, x_2, ..., x_k)$ is the set of parameters of the system, each x_i taking values in $[a_i, b_i]$. The metamodel replaces $y(\mathrm{x})$ by its approximation $\hat{y}(\mathrm{x})$. A metamodel has two essential components: the underlying mathematical model used to approximate y by $\hat{y}$ and the sample used to train it, *i.e.* adapt the parameters of the metamodel to approximate the original model. This sample is called **design**.
We give an overview of the most used metamodels, pointing out that their quality depends on a sample of the original model. We follow by an overview of the most popular techniques to produce designs.

2.1.1 Metamodeling techniques

A metamodel is a mathematical function $\hat{y}(\mathrm{x})$ which approximates the output function $y(\mathrm{x})$ of a system, depending on the values $\mathrm{x} = (x_1, x_2, ..., x_k)$ of the k parameters of the system. We describe commonly used metamodels, starting with polynomial regressions [11], a simple model which is used as a basis for two other methods we describe next: the response surface methodology [31] [17] [38] and Kriging metamodels [34] [47] [61]. We also outline radial basis function methods [16] [42] [55].

Polynomial regression

The model which produces the output is a polynomial function of the parameters of the model $\mathrm{x} = (x_1, x_2, ..., x_k)$. The coefficients of the polynomial function are the parameters of this metamodel and are to be fitted to the model. The advantage of this approach is that

the coefficients directly indicate the importance of each parameter. However, the number of coefficients to choose is exponential in the degree of the polynomial selected, which makes this model unfit for non-linear systems requiring high degree polynomials to be appropriately approximated.
An example of the model with a second order polynomial:

$$\hat{y}(\mathrm{x}) = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \sum_{j=1}^{i} \beta_{ij} x_i x_j.$$

Each β term is a parameter to be fitted to the original model. A sample of the original model is therefore needed to this end. This sample will greatly influence the metamodel and thus its precision.

Response surface

This methodology improves polynomial regression by estimating the importance of each parameter with a first low degree polynomial regression and eliminating the least significant parameter, based on the values of the coefficients of the polynomial regression. Indeed, a parameter for which the coefficients of the polynomial regression is small has little importance in the original model and can be neglected. This process is iterated until only meaningful parameters are left. This procedure allows a polynomial regression of higher degree to be performed, as diminishing the number of parameters drastically decreases the number of coefficients of the regression. This method requires, however, that the original model does not have a high number of significant parameters.

Kriging metamodels

One of the issues with polynomial regression is that it does not take into account the noise of the original system if it exists, nor the noise of the first model if the latter is a stochastic model. Kriging metamodels solve this by adding a random factor accounting for this noise to a low order polynomial regression. A Kriging metamodel is defined as follows:

$$\hat{y}(\mathrm{x}) = \sum_{i=1}^{k} \beta_i x_i + Z(\mathrm{x}).$$

The first term represents a first order polynomial regression and $Z(\mathrm{x})$ stands for a random function with zero mean. The correlation $\mathrm{cor}(Z(\mathrm{x}_i), Z(\mathrm{x}_j))$ is given by:

$$\mathrm{cor}(Z(\mathrm{x}_i), Z(\mathrm{x}_j)) = \sigma^2 R(\mathrm{x}_i, \mathrm{x}_j).$$

Function $R(\mathrm{x}_i, \mathrm{x}_j)$ is a function to be chosen. It is usually a Gaussian Radial Basis Function:

$$R(\mathrm{x}_i, \mathrm{x}_j) = \exp(-\sum_{m=1}^{k} \theta_m |x_i^{(m)} - x_j^{(m)}|^{p_m}),$$

where θ_m and p_m are parameters to be fitted to maximize a likelihood function. Note that the correlation is higher when two sample points are close one to another.
The additional parameters of this method compared to a polynomial regression make it more flexible, however, they also make it computationally more expensive. The correlation function is also a significant part of this model and it has to match with the underlying correlation function of the original system, which necessitates knowledge of this system.

Radial basis function

The radial basis function method uses the sample points to interpolate the output of the system. The output is calculated in function of the distances between the interpolated point and the sample points. It can be expressed as:

$$\hat{y}(\mathrm{x}) = \sum_{j=1}^{N} w_j \psi(||\mathrm{x} - \mathrm{x}_j||_2),$$

where x_j are the sample points, w_j are the weights and ψ is the basis function. Typical basis functions are $\psi(r) = \sqrt{r^2 + d^2}$, $\psi(r) = \log(r^2 + d^2)$ or $\psi(r) = r^2 \log(r)$, d being a constant. All these functions are decreasing relatively to the distance between two sample points.
The weights are calculated in such a way that for all sample points x_j, $\hat{y}(\mathrm{x}_j) = y(\mathrm{x}_j)$. This translates into a simple linear system, which can be solved to obtain the weights.

Conclusion

All the above-mentioned techniques include a number of parameters needing to be fitted to the original model in order to approximate it. The fitting procedure necessitates a sample of the original model. Consequently, the sample should be carefully chosen, as it will have a high impact on the accuracy of the metamodel. Multiple techniques are used to obtain a good sample. We describe them in the next section.

2.1.2 Sampling techniques

After describing various metamodels, we present techniques used to create samples. A design is the set of parameters values used to run the original simulation to train the model. A design for a simulation using k parameters can be seen as a set of points in a k-dimensional space, each dimension representing one parameter. Each point represents a simulation and each coordinate of a point represents the value of the parameter corresponding to the coordinate's dimension. The sample is the set of results obtained after running the simulation for each combination of parameters contained in the design. We call *sampling technique* the method used to produce a design. According to [37] and [28], two properties are needed for a design that will produce a good model. The first property is the coverage of the space of the parameters. All directions of this space should be covered and each parameter should have a high number of values. This property is called the **space-filling** property. The second is that they should have good projective properties: two points of the design should be separated even on a projection of the design. This property is called the **non-collapsing** property. As some parameters may be meaningless, two experiments which only differ by such a parameter would give the same result and would be a waste of computer time. This issue is prevented by the non-collapsing property. Additionally, as experiments are costly, a design should not have too many points. We present here three sampling techniques: fractional factorial designs [12] [54] [13], orthogonal arrays [41] [59] [62], and —finally— Latin hypercube designs [36] [56] [18] whose construction is the subject of this thesis.

Fractional factorial designs

A full-factorial design is a design containing all combinations of a chosen number l of values for each parameter. This principle implies that the design has l^k points, k being the number of parameters of the model. A fractional factorial design aims to lower that number by removing a fraction p of the combination. The hierarchical ordering principle [58] indicates that high order interactions do not contribute in a significant manner to the system. This means that the points selected for the design should only vary one from another for a low number of parameters. The design contains then l^{k-p} points.

By construction, most directions of the parameter space is covered. This makes them useful to determine the significant parameters of the system and the low order interactions between them. They are used to this end in the surface response methodology. However, the number of points in the design is very high, even for low values of l. The consequence of this fact is that only a few values can be selected for each parameter. Additionally, they are highly collapsing. This makes them not suited to properly train a model and other designs are usually used instead.

Orthogonal Array

An orthogonal array is a design of n points in dimension k, taking l different values. It can be seen as an $n \times k$ matrix with elements from $[\![0, l-1]\!]$. Moreover, an additional parameter of this type of designs is its strength t: any submatrix composed of t row contains a fixed number λ of occurrences of every combination of t parameter values. The number of experiments for such a design is thus λl^t. Considerably less experiments are needed to get the same number l of parameter values than with fractional design, especially for high dimensions, since the number of dimensions k does not change the number of experiments in this type of designs. By construction, they also have very good projective properties, as a projection is essentially a submatrix composed of a certain number of rows. However, the construction constraints of orthogonal array limits their space-filling properties.

Latin Hypercube Designs

A Latin Hypercube Design (LHD) of size n is a design of n points such that each parameter has n different values. For every parameter, each point takes one different value. Among those designs, maximin LHDs, *i.e.* LHDs for which the minimum distance between any pair of points is maximal, are used to properly fill the parameter space. This allows one to have much more different parameter values than the techniques we presented above, while still evenly covering the parameter space. While their projective properties are not as good as orthogonal array, they are more space-filling. This makes Latin hypercube designs widely used for metamodeling. However, the algorithmic complexity of constructing maximin LHDs is unknown. As they are important to the metamodeling field and present an interesting combinatorial problem, we decided to focus our study on them. In the next section, we give formal definitions of LHDs seen as combinatorial entities.

2.2 Latin Hypercube Designs problems

As LHD is of high interest in the field of metamodeling and as a combinatorial object, they have been the main topic of this thesis. We start by formally defining them, as well as the problems under study.

Formally speaking, a *Latin Hypercube Design* (LHD) of size n and dimension k is a set of n points $p_i \in [\![0, n-1]\!]^k$ respecting the Latin constraint: points do not share the same coordinate on any dimension. We note $\mathcal{D}_n^k$ the set of all LHD of size n and dimension k.

As the designs should be space filling, we try to get maximin LHDs, which are LHDs with maximum minimal distance between two points. Maximin LHDs are defined for a given distance. In this thesis, we only consider distances induced by a norm.

For any $D \in \mathcal{D}_n^k$, we define the separation distance $\Delta_{\mathcal{L}}(D)$ as the minimal distance (for the distance induced by norm $\mathcal{L}$) between any pair of points of D.

The maximin LHD of size n and dimension k for norm $\mathcal{L}$ is thus an LHD with maximum separation distance, *i.e.* $D^* \in \mathcal{D}_n^k$ such that

$$D^* = \underset{D \in \mathcal{D}_n^k}{\arg\max}\ \Delta_{\mathcal{L}}(D).$$

To help us study the complexity of constructing LHD, we also define partial LHDs. A partial LHD (pLHD) is an LHD with missing points. Its points can be seen as experiments already performed. We want to build a complete LHD containing those points while retaining the highest separation distance possible.
Formally, a partial Latin Hypercube Design of size n and dimension k is a set of $m \leq n$ points $p_i \in [\![0, n-1]\!]^k$ respecting the Latin constraint.
The separation distance objective allows us to define the following optimization and decision problems, which are the main problems studied in this thesis.

Problem 2.1 Partial Maximin LHD Completion Problem (pLHD-CP)**.** *Given a pLHD M of size n in a k-dimensional space with m points and a distance $d \in \mathbb{N}$, is it possible to complete M to obtain $D \in \mathcal{D}_n^k$ such that $\Delta_{\mathcal{L}}(D) \geq d$?*

Problem 2.2 Maximum Partial Maximin LHD Completion Problem (max-pLHD-CP)**.** *Given a pLHD M of size n in a k-dimensional space with m points, complete M to obtain $D \in \mathcal{D}_n^k$ such that $\Delta_{\mathcal{L}}(D)$ is maximum.*

Another way of seeing these two problems is to find another pLHD $M^{'}$ such that $M \cup M^{'}$ is an LHD and that $\Delta_{\mathcal{L}}(D)$ is maximum. Note that when $m = 0$, the partial LHD is empty. This means that an LHD is built from scratch.

Problem 2.3 Maximin LHD Construction Problem (LHD-CP)**.** *Given a k-dimensional space,a size n, and a distance $d \in \mathbb{N}$, is it possible to construct $D \in \mathcal{D}_n^k$ such that $\Delta_{\mathcal{L}}(D) \geq d$?*

Problem 2.4 Maximum Maximin LHD Construction Problem (max-LHD-CP)**.** *Given a k-dimensional space and a size n, construct $D \in \mathcal{D}_n^k$ such that $\Delta_{\mathcal{L}}(D)$ is maximum.*

As we study the complexity of these problems, we need to determine the size of their instances. In particular, as the only entries of the LHD-CP are the positive natural numbers n and k, the size of one of its instances should be $\log n + \log k$. However, the complexity of any algorithm producing an LHD is at least the size of the produced output: $\mathcal{O}(kn \log n)$, which is exponential as the size of an instance is $\log n + \log k$. Thus we consider that n and k are encoded in unary, which makes the size of an instance of the LHD-CP equal to $n + k$. As the pLHD-CP additionally contains an input m points with k coordinates whose values are at most n , the size of its instances become $n + k + km \log n$. Since $m \leq n$, the instance size can be simplified as $\mathcal{O}(kn \log n)$, the same as for the construction problem.

2.3 Conclusion

We started by giving an overview of metamodeling techniques, showing the importance of the quality of the sampling on their accuracy. We described the most used sampling techniques, also called designs, and focused on one in particular: Maximin Latin Hypercube Designs. We formally defined two problems related to this technique, the completion and the construction problems. After defining these problems, we proceed with their study. We start by exploring their complexity, focusing on the completion problem. We then find guarantees of performance for both problems, giving upper bounds and an approximation algorithm for the construction problem, and finding inapproximation results for the completion problem. We finish by giving methods to solve these problems with heuristic algorithms, focusing on the Simulated Annealing metaheuristic.

Chapter 3

Problem complexity analysis

In the previous chapter we introduced two maximin LHD problems: its construction, the LHD-CP, and its completion, the pLHD-CP. We will now study their complexity [48].
This chapter starts by the state of the art which contains a part covering the complexity of the LHD-CP and a part treating problems related to the pLHD-CP which will be used in subsequent proofs.
Next, we introduce the concept of forbidden coordinates with the use of which we formulate a generalization of the pLHD-CP.
We prove that this generalization and the original pLHD-CP are in the same class of complexity. We will take advantage of this fact in our proofs.
Before starting the complexity analysis of the pLHD-CP we state formally two notions concerning coordinate relationships in instances of both completion problems.
We show the NP-completeness of the pLHD-CP in dimensions $k \geq 3$, first for norm $\mathcal{L}_1$ before extending this result to all norms. We follow by studying the complexity of this problem on the plane, proving first its NP-completeness for norms $\mathcal{L}_1$ and then, using a similar method, for norms $\mathcal{L}_p$, with $p \in \mathbb{N}^*$. As the proof cannot be extended to norm $\mathcal{L}_\infty$, we use a different one for this case. These results are summarized in Table 3.1

Dimension	**Norm**	**Complexity**
$k > 2$	all norms	NP-complete
$k = 2$	$\mathcal{L}_p$, $p \in \mathbb{N}^*$	
	$\mathcal{L}_\infty$	

Table 3.1: Complexity of the pLHD-CP

3.1 State of the art

We present the state of the art concerning the complexity of the LHD-CP and the problems related to the pLHD-CP.

3.1.1 The LHD-CP

As seen in Chapter 2, Maximin LHDs have required properties for metamodeling and have therefore been extensively discussed in the literature. However, most studies have focused on methods and algorithms to build such designs. The complexity of the LHD-CP has only been studied in [51] and [52]. The authors of [51] proved that the LHD-CP is polynomial in dimension $k = 2$ for norms $\mathcal{L}_1$ and $\mathcal{L}_\infty$ by giving an exact algorithm for both cases. Both algorithms are linear and consist in making a regular covering of the square. The same researchers also found in [52] a polynomial-time algorithm producing optimal solutions for

norm $\mathcal{L}_\infty$ for particular values of n and k such that $n = b^k$. However, the majority of authors make the assumption that this problem is NP-complete [22], [44].

3.1.2 Problems related to the pLHD-CP

The completion problem, pLHD-CP, can be seen as a sub-problem of the k-dimensional maximum matching problem [19], defined as the following:

Problem 3.1. *k-dimensional Maximum Matching Problem*
Given a k-uniform, k-partite hypergraph $G = (V, E)$, find $M \in E$ such that for any $m_1 \in M$ and $m_2 \in M$, $m_1 \cap m_2 = \varnothing$ and $|M|$ is maximum.

Taking an incomplete LHD M, we construct an hypergraph, which is actually a k-partite k-uniform hypergraph.
We take an instance I of the pLHD-CP and transform it into an instance of the k-dimensional matching problem. Each unused parameter value on each dimension is represented by a vertex. We partition the vertices in such a way that each set contains all vertices corresponding to a parameter value of the same dimension. As adding a point to I is equivalent to choosing an unused parameter value for each dimension, it is also equivalent to adding an hyperedge between each of the vertices corresponding to these parameter values to the matching.
In particular, for $k = 2$, this matching problem becomes a bipartite matching, which is known to be in P [57]. For $k = 3$, we obtain the 3-dimensional matching problem which was proved NP-complete and even APX-complete[1] [30], as defined in [4].
However, since the graph is a complete k-partite k-graph, a perfect matching can always be found. Besides, we are interested in the distance between points, which is not represented in the matching problem. For these reasons, the k-dimensional matching problem is not practical to build a reduction from.
We can try to represent the minimum separation distance by adding constraints to the matching. The maximum matching problem with conflict graph (MMCG), which was shown to be NP-complete in [15], allows us to forbid pairs of edges and thus we can, for the two-dimensional case, model our problem as a sub-problem of the former. It is defined as follows:

Problem 3.2 Maximum Matching problem with Conflict Graph (MMCG)**.** *Given a bipartite graph $G = (V, E)$ and a conflict graph $G_c = (V_c, E_c)$ such that the vertices V_c of the conflict graph represent the edges E of G, find a maximum matching $M \in E$ such that no pair of edges of M is connected in G_c.*

While it is easy to reduce the pLHD-CP to the MMCG, the opposite is not true, as the distance constraint cannot be used in a simple manner to forbid any pair of points to be chosen simultaneously. However, we will use a similar idea for the polynomial reduction when demonstrating the NP-completeness of the pLHD-CP on the plane with norm L_∞.
Another related problem is the partial Latin square problem, first studied in [29] and [45], which is the classical puzzle of the same name: a grid with numbers that has to be completed such that each number is represented once and only once in each row and column. More formally:

Definition 3.1. *A Latin Square (LS) is an $n \times n$ matrix containing elements from $[\![1, n]\!]$, and such that each rows and columns contain each number exactly once.*

While LHDs are often referred to as Latin squares in the literature, in this thesis Latin squares only refer to the definition we have just stated.

[1] APX is the class of all NP optimization problems for which there is a polynomial-time ε-approximation algorithm, for a certain $\varepsilon > 0$.

Problem 3.3 Partial Latin Square Completion Problem (pLS-CP)**.** *Given an $n \times n$ table S, each cell being empty or containing an element from $[\![1, n]\!]$, is it possible to complete S (**i.e.** put an integer from $[\![1, n]\!]$ in each empty cell) to obtain a Latin square?*

This problem is proved to be NP-complete [14]. We use it as a starting point for polynomial reductions in several proofs of this thesis.
After giving an overview of the problems related to the pLHD-CP, we expose our study of its complexity.

3.2 Complexity of the pLHD-CP

We prove the NP-completeness of the pLHD-CP for various cases depending on the norm used and the dimension of the instance. To this end, we define a similar problem, the pLHD-FC-CP, and show that it belongs to the same complexity class as the pLHD-CP. We demonstrate that both problems are in NP and give definitions used in proofs of the following theorems. We then reduce pLHD-FC-CP to the pLS-CP for norm $\mathcal{L}_\infty$ and any dimension $k \geq 3$, and generalize the reduction to any norm $\mathcal{L}$. We then study the two-dimensional case, proving the NP-completeness of the pLHD-FC-CP and the pLHD-CP for norm $\mathcal{L}_1$, and using a similar demonstration for norms $\mathcal{L}_p$, for any $p \in \mathbb{N}^*$. However, the same method cannot be used to prove the NP-completeness of the pLHD-FC-CP and the pLHD-CP for norm $\mathcal{L}_\infty$. For this reason, we use a reduction from a different problem to prove it.

3.2.1 Completion of a partial LHD with forbidden coordinates

We start by defining another problem that we will use in the proofs: besides points already fixed by a pLHD M, we introduce forbidden coordinates. A certain number of coordinates on each dimension cannot be used when completing the pLHD. As the number of forbidden coordinates is the same on all dimensions and each coordinate can only be forbidden once, the forbidden coordinates can be modeled as a pLHD F, each point of F containing one forbidden coordinate on each dimension. Moreover, as the forbidden coordinates cannot be coordinates of points of M, $M \cup F$ is also a pLHD and completing M with respect to the forbidden coordinates means completing $M \cup F$ with a pLHD M'. However, the points of F are not actual points and thus are not involved in the computation of the separation distance. The separation distance is the separation distance of the pLHD $M \cup M'$.
We formally define this problem as such:

Problem 3.4 Partial Maximin LHD completion problem with Forbidden Coordinates (pLHD-FC-CP)**.** *Given a pLHD M of size n in a k-dimensional space with m points, another pLHD F of size n and dimension k such that $F \cap M = \varnothing$ and $F \cup M$ is a pLHD and a distance $d \in \mathbb{N}$, is it possible to complete $F \cup M$ with a pLHD $M^{'}$ such that $\Delta_{\mathcal{L}}(M \cup M^{'}) \geq d$?*

We also define the associated maximization problem:

Problem 3.5 Maximum partial Maximin LHD completion problem with Forbidden Coordinates (max-pLHD-FC-CP)**.** *Given a pLHD M of size n in a k-dimensional space with m points, another pLHD F of size n and dimension k such that $F \cap M = \varnothing$ and $F \cup M$ is a pLHD and a distance $d \in \mathbb{N}$, is it possible to complete $F \cup M$ with a pLHD $M^{'}$ such that $\Delta_{\mathcal{L}}(M \cup M^{'}) \geq d$?*

The completion problem, pLHD-CP, is a sub-problem of the newly defined pLHD-FC-CP. Indeed, if we take an instance of the pLHD-CP, it will also be an instance of the pLHD-FC-CP for which $F = \varnothing$.
We intend to prove that a polynomial-time algorithm exists for solving the pLHD-CP if and only if such an algorithm exists for solving the pLHD-FC-CP. Similarly, an approximation

algorithm for the pLHD-CP exists if and only if an approximation algorithm with the same approximation ratio ε exists for the pLHD-FC-CP.
In order to produce these results, we construct for every instance I of the pLHD-FC-CP an instance I' of the pLHD-CP with the same minimal distance and such that any manner to complete instance I can be used to complete instance I' with the same separation distance. Moreover, the size of instance I' is a polynomial function of the size of instance I.
This goal is attained by constructing consecutive instances of the pLHD-FC-CP with one forbidden coordinate less on each dimension until no forbidden coordinates are left. This means that the pLHD containing the forbidden coordinates in the new instance has one point less than the one in the original instance.

Lemma 3.1. *A polynomial function P exists such that for any instance I of the pLHD-FC-CP of size $s(I)$, an instance I' of the pLHD-FC-CP of size at most $P(s(I))$ exists with no forbidden coordinates and such that any pLHD M completing I with separation distance d also completes I' with the same separation distance d.*

Proof. Our reasoning is performed in spaces $\mathbb{R}^k$ with any norm $\mathcal{L}$.
We take an instance I of the pLHD-FC-CP of size n in dimension k, pLHD M, and f forbidden coordinates on each dimension. We note $(e_1, \dots, e_k)$ the standard basis of $\mathbb{R}^k$, and Δ^* the maximum separation distance that can be attained when completing M.
We construct a pLHD M_1 with $f-1$ forbidden coordinates and iterate this construction to obtain a pLHD M_f with no forbidden coordinates.
We remove a forbidden coordinate on each dimension by putting a point which has a coordinate value equal to this forbidden coordinate on the corresponding dimension. To be able to remove the forbidden coordinates as described above, without violating the Latin and separation distance constraints, we need supplementary points. More precisely speaking, we remove one forbidden coordinate on each dimension by adding $4k^2n^2+1$ points to the initial pLHD, k of which actually use one forbidden coordinate. We give below details of this procedure together with Fig. 3.1 providing its illustration.
To place those points, we add $4kn^2+1$ coordinates to the pLHD M in each dimension. We insert them after the already existing coordinates. On each dimension, we thus have the n previously existing coordinates with values from 0 to $n-1$ and then $4k^2n^2+1$ "new" coordinates with values from n to $4k^2n^2$. The strips of added coordinates intersect. We divide the square obtained by this intersection in the first two dimensions into 16 squares of size $kn^2 \times kn^2$. In both dimensions, the central coordinate, $n+2k^2n^2$, separates the strips formed by two consecutive squares. We then fill four of the squares with LHDs $D_{n,k}$ of size k^2n^2, which are LHDs with a separation distance at least Δ^*.
We now provide the explanation of the $D_{n,k}$ construction.
To begin with we suppose, without loss of generality, that for norm $\mathcal{L}$, the lengths of the basis vectors are:

$$||e_1|| \geq ||e_2|| \geq \cdots \geq ||e_k||. \tag{3.1}$$

Any LHD $D \in \mathcal{D}_n^k$ has a separation distance

$$\Delta_{\mathcal{L}}(D) \leq ||e_1|| + (k-1)n||e_2||. \tag{3.2}$$

Indeed, if we take two points of D for which the first coordinate differs by one, their other coordinates differ by at most n. Then, the triangle inequality along with Eq. (3.1) leads to Eq. (3.2). Since an LHD of size n can only have a separation distance respecting Eq. (3.2), we can suppose that $\Delta^* \leq ||e_1|| + (k-1)n||e_2||$.
The construction of $D_{n,k}$ is as follows (see Fig. 3.2 for $n=2$ and $k=2$): for every value of $(i,j) \in [\![0, kn-1]\!]^2$ we add a point whose first two coordinates are $((i+1)kn-(j+1), i+jkn)$. The coordinates of the other points do not matter. This construction ensures that each point is at a distance at least Δ^* on at least one dimension from all other points. Indeed, two

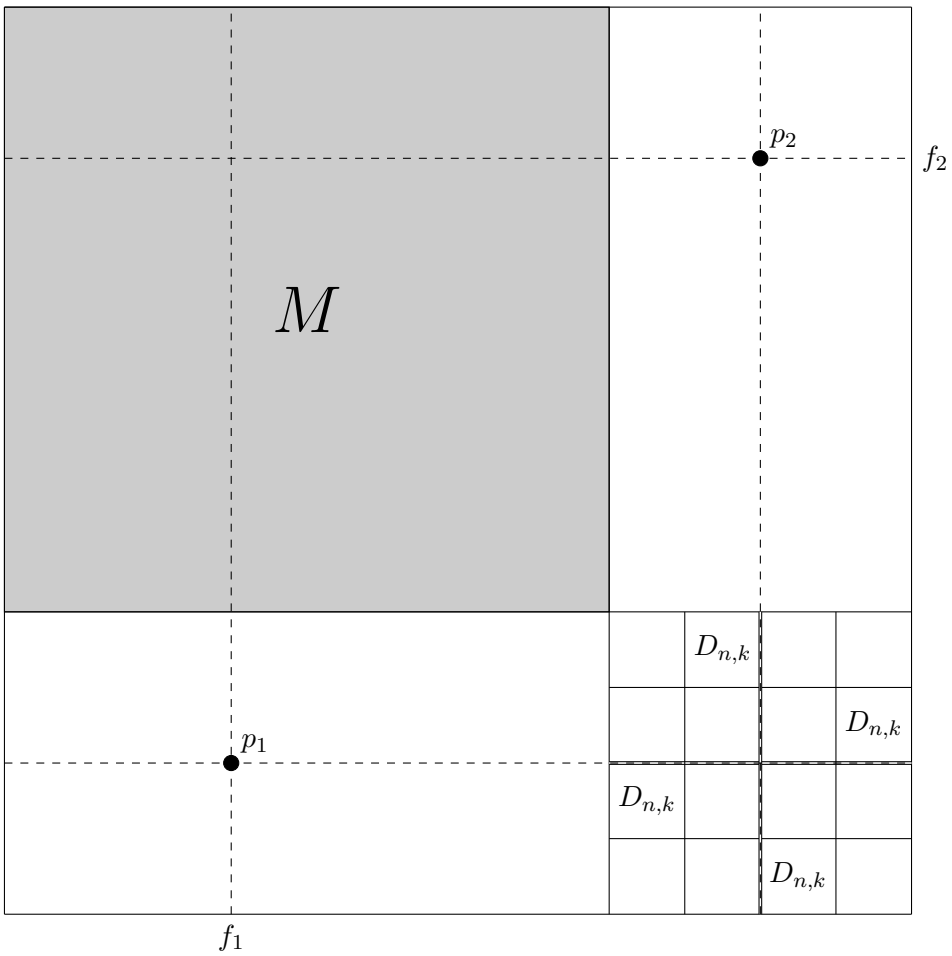

Figure 3.1: The transformation of M, a pLHD with f forbidden coordinates, into M_1, a pLHD with $f-1$ forbidden coordinates. Both pLHD can be completed with the same points, resulting in the same separation distance. Repeating the depicted process f times results in a pLHD with no forbidden coordinates. In this example, forbidden coordinates f_1 and f_2 are removed by adding points p_1 and p_2, which do not reduce the separation distance.

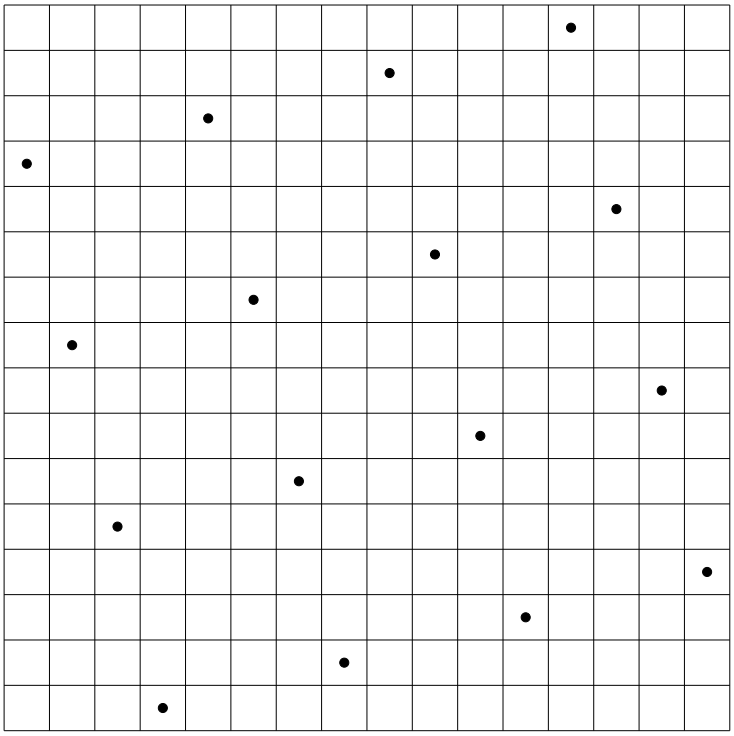

Figure 3.2: Construction of $D_{n,k}$ for $n = 2$ and $k = 2$.

different points are created using either different values of i or j, and thus either their first coordinates or their second coordinates differ by at least kn. The distance between the two points is then $kn||e_1|| + ||e_2|| \leq d$ in the first case and $||e_1|| + kn||e_2|| \leq d$ in the second case. This leads to:

$$\Delta^* \leq d.$$

The four squares in which we position our squares $D_{n,k}$ with the Δ^* guarantee form an optimal LHD of size 4×4. We then insert two last points, whose coordinates on the first two dimensions are $(f_1, n + 2k^2n^2)$ and $(n + 2k^2n^2, f_2)$, f_1 and f_2 being forbidden coordinates on the first dimension (for f_1) and second dimension (for f_2). As we added $4n + 2k^2n^2 + 2$ points and $4n + 2k^2n^2 + 1$ coordinates on each dimension, all the points we constructed use the $4n + 2k^2n^2 + 1$ new coordinates and one forbidden coordinate on each of the remaining dimensions. As long as the coordinates of each point on the remaining dimensions are pairwise different (to respect the Latin constraint), the choice of these coordinates does not matter. By construction, each of the added points are at distance at least Δ^* from another, which means that they will not take part in the separation distance. The result is a pLHD with $f - 1$ forbidden coordinates, depicted in Fig. 3.1.

As the completion of this pLHD M_1 only adds points inside the copy of pLHD M, adding the same points to M will result in the same separation distance. Conversely, any pLHD M' which completes the initial pLHD M also completes the resulting pLHD M_1.

We iterate this process f times, adding $4k^2n^2+1$ new coordinates and removing one forbidden coordinate in each dimension for each iteration, and obtain a pLHD M_f of size $n+4fk^2n^2+f$ with no forbidden coordinates. As for M_1, any pLHD M' completing M also completes M_f with the same separation distance. Conversely, any pLHD M' completing M_f completes M with the same separation distance.

An instance I' of the pLHD-FC-CP is defined by M_f, which is a pLHD of size $n+4fk^2n^2+f$, dimension k, with all coordinates permitted, giving $F = \varnothing$. The bound on the separation distance is the same as the one of I. The size of I' is then

$$s(I') = s(I) + 4fk^2n^2.$$

The size of instance I' is thus polynomial in the size of the instance I, which concludes the proof.

□

Theorem 3.1.

- *A polynomial-time algorithm exists for solving the pLHD-CP if and only if such an algorithm exists for solving the pLHD-FC-CP.*
- *For any $\varepsilon > 0$, an approximation algorithm with an approximation ratio ε for the max-pLHD-CP exists if and only if an approximation algorithm with an approximation ratio ε exists for the max-pLHD-FC-CP.*

Proof. Suppose that a polynomial-time algorithm A solving the pLHD-CP exists. Let I_F be an instance of the pLHD-FC-CP. As stated in Lemma 3.1, an instance I'_F of the pLHD-FC-CP with no forbidden coordinates exists such as any pLHD completing it also completes I_F with the same separation distance. Conversely, any pLHD completing I_F also completes I'_F with the same separation distance. This means that I_F is a positive instance of the pLHD-FC-CP if and only if I'_F is a positive instance of the pLHD-FC-CP.
As all coordinates of I'_F are allowed, I'_F is also an instance of the pLHD-CP. This means we can apply A to solve it and also solve I_F since the two instances are either both positive or both negative.
Again, as the pLHD-CP is a subproblem of the pLHD-FC-CP, any algorithm solving the latter also solves the former.
The same reasoning also establishes that an ε-approximation algorithm for the max-pLHD-CP exists if and only if an ε-approximation algorithm exists for the max-pLHD-FC-CP.

□

As the reader will see in the next section, both problems belong to the NP class.

3.2.2 Complexity of the pLHD-CP verifier

The complexity of the pLHD-FC-CP certificate verification depends on the complexity of the distance computation.

Theorem 3.2. *The pLHD-FC-CP is in NP for all norms $\mathcal{L}$ which induce a distance $\delta(p_1, p_2)$ computed in a polynomial time in the highest coordinate of points p_1 and p_2.*

Proof. Let $\mathcal{L}$ be a norm whose distance δ between two points p_1 and p_2 whose coordinates are all lower than an integer m can be computed in $L(m)$ operations, where L is a polynomial function. Let I_M be a positive instance of the pLHD-FC-CP for $\mathcal{L}$, with a lower bound for the separation distance $d_{\min}$, pLHD M of size n, and forbidden coordinates F.
Since I_M is a positive instance, $M \cup F$ can be completed into an LHD with a pLHD M_c, the separation distance of $M \cup M_c$ being $\Delta \geq d_{\min}$.
The partial LHD M_c is a certificate for I_M: indeed, it suffices to verify that $M \cup M_c \cup F$ is an LHD of size n (n operations to check the size of $M \cup M_c \cup F$ and $k\frac{n(n-1)}{2}L(n)$ operations to check that all coordinates are pairwise different) and the separation distance of $M \cup M_c$ $\Delta \geq d_{\min}$. The latter takes $\frac{n(n-1)}{2}L(n)$ operations, where $L(n)$ is the number of operations needed to compute the distance δ between two points whose coordinates are all smaller than n.

□

Theorem 3.3. *The pLHD-CP is in NP for all norms $\mathcal{L}$ which induce a distance $\delta(p_1, p_2)$ computed in a polynomial time in the size of p_1 and p_2.*

Proof. As the pLHD-CP is a subproblem of the pLHD-FC-CP, this result is immediate. □

3.2.3 Definitions concerning completion problems

After proving that both problems, the pLHD-CP and the pLHD-FC-CP, are in NP, we give two definitions and which will be used to prove the NP-completeness of the pLHD-FC-CP and the pLHD-CP in dimensions $k \geq 3$.

Definition 3.2. *A **free coordinate** for a given dimension is a value which a point can take in this dimension as it is neither used by points already set nor forbidden.*

Note that because of the Latin constraint, there is as many free coordinates in each dimension as points to be constructed. Moreover, each free coordinate must be taken by one point inserted.
We also introduce a notion of closeness between two points or coordinates on a dimension:

Definition 3.3. *Two points or coordinates are (d, m)-close if their coordinates on dimension m differ by at most d.*

From a geometric point of view, if two points are (d, m)-close, it means that they are both on a strip of width d on dimension m. Note that two points (d, m)-close on all dimensions m are indeed close to each other, according to the usual definition. By contrast, points (d, m)-close on all dimensions but one may be far from each other (according to the distance).
These definitions are used in the proofs of the following sections.

3.2.4 NP-completeness in dimensions $k \geq 3$

We demonstrate that the LHD completion problem for dimension $k \geq 3$ is NP-complete. We will first prove this fact for norm $\mathcal{L}_1$ and, next, extend this result to all norms on $\mathbb{R}^k$.

Results for norm $\mathcal{L}_1$

We start by studying the problem for norm $\mathcal{L}_1$. The choice of this norm is arbitrary as the reduction we use would be similar to the reductions for other usual norms.
After stating that the pLHD-FC-CP is NP-complete for $k = 3$, we extend this result to $k \geq 3$, and subsequently use it to prove that the pLHD-CP is also NP-complete for $k \geq 3$ by taking advantage of the fact that the pLHD-CP and the pLHD-FC-CP are equivalent in terms of the complexity (Theorem 3.1 in Section 3.2.1).

Theorem 3.4. *The pLHD-FC-CP is NP-complete for $k = 3$ and norm $\mathcal{L}_1$.*

Proof. To prove the NP-completeness we reduce the pLS-CP (Problem 3.3) to the pLHD-FC-CP. Let S be an instance of the pLS-CP of size $n \times n$. We construct a corresponding instance M of the pLHD-FC-CP in a three-dimensional space.
We note $d_{\min}$ a minimal separation distance between points in M which will be determined later.
Each tile of S will be represented by a gadget. Each gadget will be composed of a number of points in a specific region of the hypercube we construct, placed in such a way to force the positions of the points to be added when completing M. Since S contains $n \times n$ tiles, M will be constructed with $n \times n$ gadgets. Each gadget will initially have one missing point, and for each tile of S containing a number, we add to the corresponding gadget a point coding that number.
We note M_0 the pLHD containing the gadgets before adding a coding point to them.
We build a gadget which represents a tile of S. This gadget is a square cuboid coding the value inside the corresponding tile, while its position codes the position of the tile. The square side of this cuboid is its orthogonal projection on the plane spanned on the second and third dimensions of M_0. In the remainder of this proof we will refer to this projection as

a block. The third side of this cuboid will have the same size as a full side of M_0. The initial pLHD M_0 is an $n \times n$ square (in the second and third dimensions) of gadgets positioned in the same way as the tiles of S they represent and separated one from another by $d_{\min}$ forbidden coordinates.
In the first dimension of M_0, we have n^2 free coordinates whose values differ by $d_{\min} + n^4$. They are also free coordinates for the gadgets.
We will proceed with the construction of these gadgets. An illustration of this process is given in Fig. 3.3.
On a block, we have n free coordinates in both dimensions. Our goal is twofold: to force points to be put on the block diagonal and to avoid putting more than one point per block. Once we have done that, the position of a point on the diagonal will code the value of the corresponding tile inside the partial Latin square.
We achieve this goal by putting n^2 points $(n^4, 2)$-close and $(n^4, 3)$-close to each intersection of rows and columns whose coordinates are free, except for the intersections on the diagonal of this block. For these intersections, we put only $n^2 - 1$ points $(n^4, 2)$-close and $(n^4, 3)$-close.
We count the number of points $(n^4, 2)$-close (or $(n^4, 3)$-close) to one free coordinate. A free coordinate crosses a line of n blocks. In each block, this coordinate intersects with n others. We put n^2 points $(n^4, 2)$-close (or $(n^4, 3)$-close) to each intersection on both dimensions. Therefore, we have n^4 points $(n^4, 2)$-close (or $(n^4, 3)$-close) to this coordinate. This means we can put the points within a distance of n^4 from each intersection.
Now let us consider the expansion of points laying on the plane into the three-dimensional space. We place all the n^2 points we constructed for each intersection $(n^4, 1)$-close to a free coordinate of the first dimension. For the intersections on the diagonal, we put one point less, which means that one free coordinate of the first dimension has no point $(n^4, 1)$-close to it. This coordinate is the same for each point on the diagonal of a same block but differs from one gadget to another.
Each free coordinate differs from another by at least $d_{\min} + n^4$. Note that if we put one point on the diagonal of a block and we choose for the first coordinate the free coordinate with no points $(n^4, 1)$-close to it, this point will be at distance at least $d_{\min}$ of all other points. If two or more points are on the diagonal of each block, or if a point is not on the diagonal, at least two points will be at a distance at most $3n^4$.
Observe that each point is distant from at least $d_{\min}$ from any other. Finally, for each gadget, we add one point on the block diagonal on position (a, a) if the corresponding tile of the Latin square contains value a. If the tile is empty, we do not do anything. This point is the coding point, and codes the value a.
Now suppose that we can complete the Latin square S. As long as we take $d_{\min}$ such that $d_{\min} > 3n^4$, it is possible to put one point on the block diagonal of each gadget. Thus, the separation distance is greater than $d_{\min}$. Similarly, if we can complete the gadget structure keeping a separation distance greater than $d_{\min}$, we only put points on the diagonal of each block and complete S.
As the pLHD-FC-CP is in NP for $\mathcal{L}_1$ (Theorem 3.3 in 3.2.2), we obtain its NP-completeness. □

We extend this result to all dimensions $k > 3$:

Theorem 3.5. *The pLHD-FC-CP is NP-complete for $k \geq 3$ and norm $\mathcal{L}_1$.*

Proof. We perform the same construction as in the proof of Theorem 3.4 with dimensions added. We put all the constructed points and free coordinates $(n^6 + n^2, m)$-close on all additional dimensions m. Since the number of points we construct is n^6 and the number of free coordinates is n^2, we can put them all $(n^6 + n^2, m)$-close to each other. In this case, we chose $d_{\min} = (k-3)(n^6 + n^2) + 3n^4$, where k is the number of dimensions. The reduction is

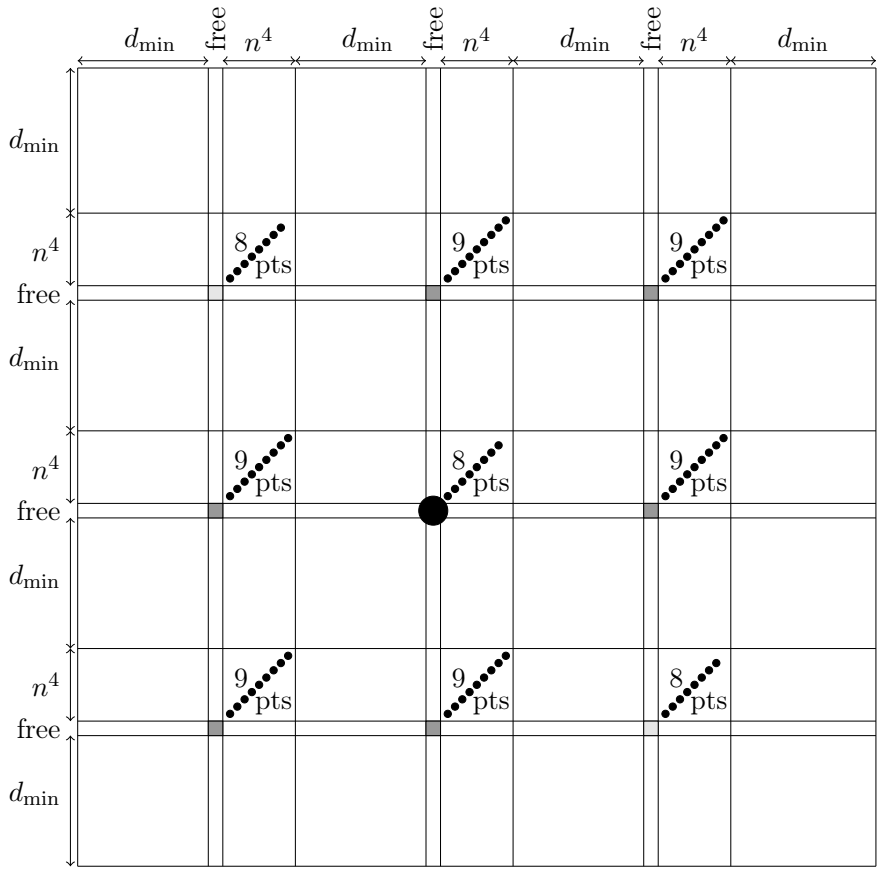

Figure 3.3: Dimensions 2 and 3 of a gadget (a block) for the reduction for $n = 3$ and $k = 3$. The intersection of free coordinates outside of the block diagonal (dark gray areas) are $(n^4, 2)$-close and $(n^4, 3)$-close to $3^2 = 9$ points. The intersections on the block diagonal (light gray areas) are $(n^4, 2)$-close and $(n^4, 3)$-close to 8 points. The big point represents the coding point which codes here the value 2.

immediate: completing the pLHD we obtained with a separation distance $d_{\min}$ is equivalent to completing the Latin square.

□

We go back to the original problem:

Theorem 3.6. *The pLHD-CP is NP-complete for $k \geq 3$ and norm $\mathcal{L}_1$.*

Proof. The straightforward consequence of Theorems 3.1 and 3.4.

□

We extend the previous results to all norms on $\mathbb{R}^k$ by using the norm equivalence in a finite-dimensional space theorem [26]:

Theorem 3.7 (taken from [26])**.** *Let $\mathcal{L}$ and $\mathcal{L}'$ be two norms on $\mathbb{R}^k$, and δ and δ' the distances induced by them.*
Then, it exists $c_1 > 0$ and $c_2 > 0$ such that for any point $p_1 \in \mathbb{R}^k$ and $p_2 \in \mathbb{R}^k$, we have

$$c_2\delta'(p_1, p_2) \geq \delta(p_1, p_2) \geq c_1\delta'(p_1, p_2). \tag{3.3}$$

This theorem will be used in the proof of the following one:

Theorem 3.8. *The pLHD-FC-CP is NP-complete in spaces of dimension $k \geq 3$ for any norm.*

Proof. We use Theorem 3.7 to prove that the pLHD-FC-CP remains NP-complete for any norm.
Let $\mathcal{L}$ be a norm and δ the distance induced by it. In particular, we note δ_1 the distance for norm $\mathcal{L}_1$.
Let S be an instance of the pLS-CP and M be an instance of the pLHD-FC-CP for norm $\mathcal{L}$ constructed with the same method as in the proof of Theorem 3.4. We show that this construction is also a reduction for norm $\mathcal{L}$ with an appropriate choice of $d_{\min}$ and of the lower bound on the separation distance.
The distance chosen for the construction of M is $d_{\min} = \frac{c_2}{c_1}(3n^4 + 1)$. Note that since $\frac{c_2}{c_1} \geq 1$, then $d_{\min} \geq 3n^4 + 1$. We select as a lower bound for the separation distance $\Delta_{\min} = c_2(3n^4 + 1)$.
Suppose that we can complete S. The proof of Theorem 3.4 ensures that we can complete M into an LHD M_c with separation distance $\Delta_1 \geq d_{\min}$ for norm $\mathcal{L}_1$ and Δ for norm $\mathcal{L}$.
Let $p_1, p_2 \in M_c$ be two points such that $\delta(p_1, p_2) = \Delta$. We know that $\delta_1(p_1, p_2) \geq \Delta_1 \geq d_{\min}$. Combining it with inequality (3.3) we obtain:

$$\delta(p_1, p_2) \geq c_1 d_{\min},$$

and thus

$$\Delta \geq c_2(3n^4 + 1).$$

Now suppose that we can complete M into an LHD M_c with separation distance $\Delta \geq \Delta_{\min}$ for norm $\mathcal{L}$ and Δ_1 for norm $\mathcal{L}_1$. If we show that $\Delta_1 \geq 3n^4 + 1$, we will be able to complete S into a Latin square in the same way as in the proof of Theorem 3.4.
As before, let $p_1, p_2 \in M_c$ be two points such that $\delta_1(p_1, p_2) = \Delta_1$. As before, $\delta(p_1, p_2) \geq \Delta \geq \Delta_{\min}$, which, together with inequality (3.3), gives:

$$\Delta_1 = \delta_1(p_1, p_2) \geq \frac{\delta(p_1, p_2)}{c_2} \geq \frac{\Delta_{\min}}{c_2},$$

and finally:

$$\Delta_1 \geq 3n^3 + 1.$$

Thus, we can complete S into a Latin square.
To conclude the proof, we use Theorem 3.3 in Section 3.2.2.

□

Theorem 3.9. *The pLHD-CP is NP-complete in spaces of dimension $k \geq 3$ for any norm.*

Proof. The straightforward consequence of Theorems 3.1. and 3.8.

□

3.2.5 Dimension $k = 2$ for norms $\mathcal{L}_1$ and $\mathcal{L}_p$

After treating dimensions $k \geq 3$, we focus on dimension $k = 2$. We start by proving the NP-completeness of the pLHD-CP for norm $\mathcal{L}_1$. The result is next extended for norms $\mathcal{L}_p$.

NP-completeness for norm $\mathcal{L}_1$

The proof of the following theorem consists in reducing the pLS-CP to the pLHD-CP.

Theorem 3.10. *The pLHD-CP is NP-complete in the two-dimensional space with norm $\mathcal{L}_1$.*

Proof. Let S be an instance of the pLS-CP of size n.
We start by constructing a pLHD which codes S then verify that what our construction produced is indeed a pLHD with a certain separation distance. We finish by proving that it is possible to complete the constructed pLHD with a certain separation distance if and only if S can be completed into a Latin square.
We build a pLHD M composed of $n \times n$ gadgets. Each gadget is a $4n \times 4n$ square block which contains a point coding the value inside the corresponding tile. The size of M is thus $4n^2 \times 4n^2$. A gadget is illustrated in Fig. 3.4. The gadgets are arranged in M in the same way as the tiles of that$S$ they code.
When building a gadget, we pay attention to the four square blocks of size $n \times n$ on its antidiagonal. We index them bottom-up noting them B_1, B_2, B_3, and B_4. Any of the middle blocks may be used to code the value of a tile of S. We arbitrarily chose block B_2 to this effect. The point inside B_2 is called a coding point. The other blocks are used to constrain the positions we can choose when adding a coding point and to ensure we can combine the gadgets together to build M.
In order to code a tile value, we force the points of B_2 to be on its diagonal. This allows us to use the Latin constraint to prevent other gadgets, either on the same line or the same column, from coding a tile with an identical value.
To ensure that the point of B_2 is on its diagonal, we put a point on the diagonals of B_1, B_3, and B_4. The distance constraint will force us to put a point on the diagonal of B_2. The coordinates of this point in the $n \times n$ block will code the value in a tile of the Latin square S. If the corresponding tile is empty, this block will remain empty too. If it is not, we put a point in position (a, a), where a is the value of the corresponding tile in S.
We arrange n^2 gadgets in a grid of size $n \times n$. This construction guarantees that the original Latin square can be completed if and only if we can complete the constructed pLHD into an LHD with separation distance $d_{\min} = 2n$. We explain this fact formally.
We note $p(b, i, j)$ the point put in block B_b in the gadget at position (i, j). Into blocks $b = 1, 3$, and 4 of a gadget at position (i, j) we insert the following points:

$$p(b, i, j) = (4ni + (b-1)n + 1 + (i+j) \bmod n, 4nj + bn - 1 - (i+j) \bmod n).$$

These points constrain the positions on which the remaining points can be added. Note that the remaining points must be placed in the coding blocks of the gadgets, as all other

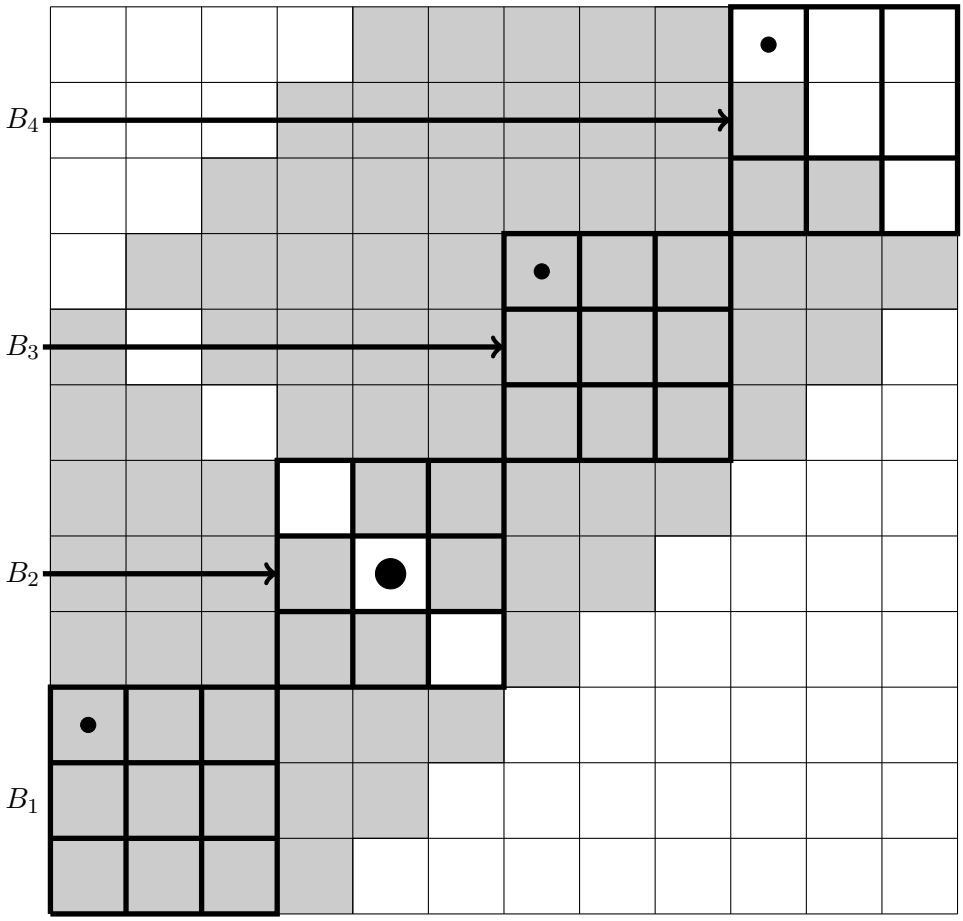

Figure 3.4: Construction of a gadget for the reduction for $n = 3$, $k = 2$, and norm $\mathcal{L}_1$. Block B_2 contains a coding point. In this example, it codes the value 2. The gray areas indicate the positions where a point would be closer than $d_{\min} = 2n = 6$ to the points in B_1 or B_3.

coordinates have been already taken by the points we have just inserted into B_1, B_3, and B_4.
We note $S(i,j)$ the value of the tile (i,j) of S, if it exists. The point coding this tile value is:

$$p(2,i,j) = (4ni + n + S(i,j) - 1, 4nj + 2n - S(i,j)).$$

After finishing our construction, we should verify whether all added points form a pLHD M and are separated one from another by a distance at least $2n$. We start by computing the distance between points.
Let $p_1 = p(b_1, i_1, j_1)$ and $p_2 = p(b_2, i_2, j_2)$ be two different points of M separated by distance $\delta_1(p_1, p_2)$. If both points are in the same gadget, $i_1 = i_2 = i$, $j_1 = j_2 = j$ and $b_1 \neq b_2$ then $p_1 = p(b_1, i, j)$ and $p_2 = p(b_2, i, j)$. We note $R_s^{(t)} = p_s^{(t)} \bmod n$, where $p_s^{(t)}$ signifies the t^{th} coordinate of p_s. This notation is used to express the distance between p_1 and p_2:

$$\begin{aligned}\delta_1(p_1,p_2) = |4ni + (b_1-1)n + R_1^{(1)} - 4ni - (b_2-1)n - R_2^{(1)}| + \\ |4nj + (b_1-1)n + R_1^{(2)} - 4nj - (b_2-1)n - R_2^{(2)}|,\end{aligned}$$

which may be rewritten as:

$$\delta_1(p_1,p_2) = |(b_1-b_2)n + R_1^{(1)} - R_2^{(1)}| + |(b_1-b_2)n + R_1^{(2)} - R_2^{(2)}|.$$

Knowing that $|R_1^{(1)} - R_2^{(1)}| < n$ and $|R_1^{(2)} - R_2^{(2)}| < n$, we can remove these terms from the absolute value and group them differently:

$$\delta_1(p_1,p_2) = 2|b_1-b_2|n + R_1^{(1)} + R_1^{(2)} - (R_2^{(1)} + R_2^{(2)}).$$

If $b_1 \neq 2$ then $R_1^{(1)} = i+j \bmod n$ and $R_1^{(2)} = n-1-(i+j \bmod n)$ and thus $R_1^{(1)}+R_1^{(2)} = n-1$.
Similarly, if $b_2 \neq 2$ then $R_2^{(1)} + R_1^{(2)} = n-1$.
If $b_1 = 2$, we get $R_1^{(1)} = S(i,j) - 1$ and $R_1^{(2)} = n - S(i,j)$ and thus $R_1^{(1)} + R_1^{(2)} = n-1$. In the same way, if $b_2 = 2$ then $R_2^{(1)} + R_2^{(2)} = n-1$.
The distance between points p_1 and p_2 becomes:

$$\delta_1(p_1,p_2) = 2|b_1-b_2|n \geq 2n.$$

After checking the distance, we verify that M is indeed a pLHD by establishing that any two points do not share a coordinate. As before, $p_1 = p(b_1, i_1, j_1)$ and $p_2 = p(b_2, i_2, j_2)$ are two different points of M. We need to examine several cases:
First case, both points are in different blocks, $b_1 \neq b_2$: in this situation,

$$p_1^{(1)} \bmod 4n \neq p_2^{(1)} \bmod 4n$$

$$p_1^{(2)} \bmod 4n \neq p_2^{(2)} \bmod 4n.$$

Thus, we get $p_1^{(1)} \neq p_2^{(1)}$ and $p_1^{(2)} \neq p_2^{(2)}$.
Second case, both points are in the same block, $b_1 = b_2 = b$: we either have $i_1 \neq i_2$ or $j_1 \neq j_2$. We examine here these two sub-case.
In the first sub-case, the quotients of the division of $p_1^{(1)}$ and $p_2^{(1)}$ by n are different, which means that $p_1^{(1)} \neq p_2^{(1)}$. Similarly, in the second sub-case, the quotients of the division of $p_1^{(2)}$ and $p_2^{(2)}$ by n are different and therefore $p_1^{(2)} \neq p_2^{(2)}$.
We still have to prove that $i_1 \neq i_2$ and $j_1 = j_2 = j$ leads to $p_1^{(1)} \neq p_2^{(1)}$, and symmetrically, $j_1 \neq j_2$ and $i_1 = i_2 = i$ gives $p_1^{(2)} \neq p_2^{(2)}$.

When $b \neq 2$, we have $i_1 + j_1 \neq i_2 + j_2$, and thus $p_1^{(1)} \bmod n \neq p_2^{(1)} \bmod n$, which gives $p_1^{(1)} \neq p_2^{(1)}$. The same reasoning allows us to prove the symmetrical part.
If $b = 2$, then

$$p_1^{(1)} \bmod n = S(i_1, j),$$

$$p_2^{(1)} \bmod n = S(i_2, j).$$

In this situation, if $p_1^{(1)} = p_2^{(1)}$, we would have $S(i_1, j) = S(i_2, j)$, which is impossible as S is a partial Latin square. Similarly, as

$$p_1^{(2)} \bmod n = n - S(i, j_1)$$

$$p_2^{(2)} \bmod n = n - S(i, j_2),$$

if $p_1^{(2)} = p_2^{(2)}$, we would have $S(i, j_1) = S(i, j_2)$, which is impossible.
We conclude that M is a pLHD with separation distance $d_{\min} = 2n$.
We now prove the equivalence of completing S and completing M with a separation distance $d_{\min} = 2n$.
We now suppose that S can be completed into a Latin square S_c. We then put points in the empty blocks B_2 to represent values in the tiles of S_c and we get a complete LHD of size $4n^2$ with separation distance $d_{\min} = 2n$.
Starting from the pLHD-CP, we suppose that M can be completed in an LHD of size $4n^2$ with separation distance $d = 2n$. The construction described above ensures that coding points can be placed only on the diagonal of the block B_2 of each gadget. Indeed, a point outside of this block would share a coordinate with another point. Moreover, points in this block, but outside its diagonal, would be at a distance less than $2n$ of any other point. The distance constraint also prevents us from putting more than one point in each block. As the coding points are on the diagonal of a block B_2, their coordinates inside that block are (a, a). We put the value a in the corresponding, initially empty, tile of S. Since M lacks a point for each empty tile of S, and two coding points either on the same line or on the same column cannot code the same value, S is completed into a Latin square. □

The construction made in the proof above can be extended to demonstrate that the pLHD-CP is also NP-complete for all norms $\mathcal{L}_p$.

Results for any norm $\mathcal{L}_p$

In the two-dimensional space the pLHD-CP is NP-complete for any norm L_p. We perform a similar reduction from the pLS-CP as for norm $\mathcal{L}_1$ in the proof of Theorem 3.10.
Points which are at the same distance from a given point according to the metrics derived from norms $\mathcal{L}_p$ form a "classical circle" when $p = 2$ and a hyper-circle which becomes more and more alike a square with rounded corners when p goes up. The proof of the following theorem takes advantage of certain concepts elaborated in the proof of Theorem 3.10 for norm $\mathcal{L}_1$ for which a "circle" is actually a square whose diagonals lie on the vertical and horizontal axes of the Cartesian system.

Theorem 3.11. *The pLHD-FC-CP is NP-complete for any norm $\mathcal{L}_p$ with $p \in \mathbb{N}^*$ and $k = 2$.*

Proof. In the reduction, we use the pLHD-FC-CP to forbid certain coordinates in the pLHD we will construct (as we did in the proof of Theorem 3.6). The idea is to have a distance $d_{\min}$ large enough to approximate the circle for norm $\mathcal{L}_p$ as a straight line, so the only places where we can put a point are on the diagonal of the block of free coordinates, as it was naturally the case for norm $\mathcal{L}_1$ in the previous construction. Once this is done, we can use the same

method as in the proof of Theorem 3.10 to reduce the pLS-CP into the pLHD-FC-CP. As in the above-mentioned proof, a gadget is a square containing a coding point. The position of this point codes a value inside a tile of a Latin square, while the position of the gadget codes the position of the tile. The complete pLHD will be an $n \times n$ array of gadgets separated by $d_{\min}$ forbidden coordinates. The gadgets are thus sufficiently far one from another to prevent two points from different gadgets from intervening in the separation distance.
We build three blocks $n \times n$ on the antidiagonal of each gadget. To make a gadget large enough to approximate a circular arc by a line segment inside it, we separate its consecutive antidiagonal blocks by f forbidden coordinates. The middle block has n free coordinates (see Definition 3.2) in each dimension. This block contains a coding point if the corresponding tile of the partial Latin square has a value inside it. We put this point on the diagonal, in position (a, a), where a is the value inside the tile, as we did for the reduction to prove Theorem 3.10. This construction is illustrated in Fig. 3.5.
The two other squares contain a point on their diagonal. To be sure that a coding point can be inserted only on the diagonal of the block in the middle of any gadget, we check the distance between a coding point on the diagonal in position (i, i) in the coding block and a fixed point in position (j, j) in one of the two other square blocks of the gadget:

$$d_{i,j} = \sqrt[p]{(f+n+i-j)^p + (f+n+j-i)^p}.$$

The shortest distance is obtained when $i = j$:

$$d_{\min} = \sqrt[p]{(f+n)^p + (f+n)^p}.$$

We also check the distance between a point in position $(i, j), i \neq j$, and the closest fixed point in position (k, k) in a corner square. We suppose without loss of generality that $i < j$:

$$d_{i,j,k} = \sqrt[p]{(f+n+i-k)^p + (f+n+k-j)^p}.$$

The greatest distance is reached for $i = 0, j = 1, k = n-1$:

$$d_{\max} = \sqrt[p]{(f+2n-1)^p + (f+1)^p}.$$

If we prove that $d_{\min} > d_{\max}$, the reduction will be accomplished, as completing the pLHD we constructed in an LHD with separation distance $d_{\min}$ will be equivalent to complete the partial Latin square in the same manner as in the reduction in the proof of Theorem 3.10.
We consider the polynomial: $P(f) = d_{\min}^p - d_{\max}^p$. We rewrite it as:

$$P(f) = 2(f+n)^p - (f-1)^p - (f+2n-1)^p. \tag{3.4}$$

Let us represent $P(f)$ as $\sum_{i=0}^{p} a_i f^i$. We examine its coefficients in order to determine its degree.
We first consider the coefficient of the term of degree p:

$$a_p = 2 - 1 - 1 = 0.$$

We continue with the coefficient of the term of degree $p-1$:

$$a_{p-1} = 2pn + 2p - (2pn - 2p) = 4p.$$

The highest degree term of $P(f)$ is thus $4pf^{p-1}$. Observe that $P(f) > 0$ for any f beyond the greatest root $r_{\max}$. To bound $r_{\max}$ with polynomial coefficients we use the Rouché theorem which allows us to say that for any root r:

$$|r| \leq 1 + \frac{1}{a_{p-1}} \max\Big(|a_0|, |a_1|, ..., |a_{p-2}|\Big).$$

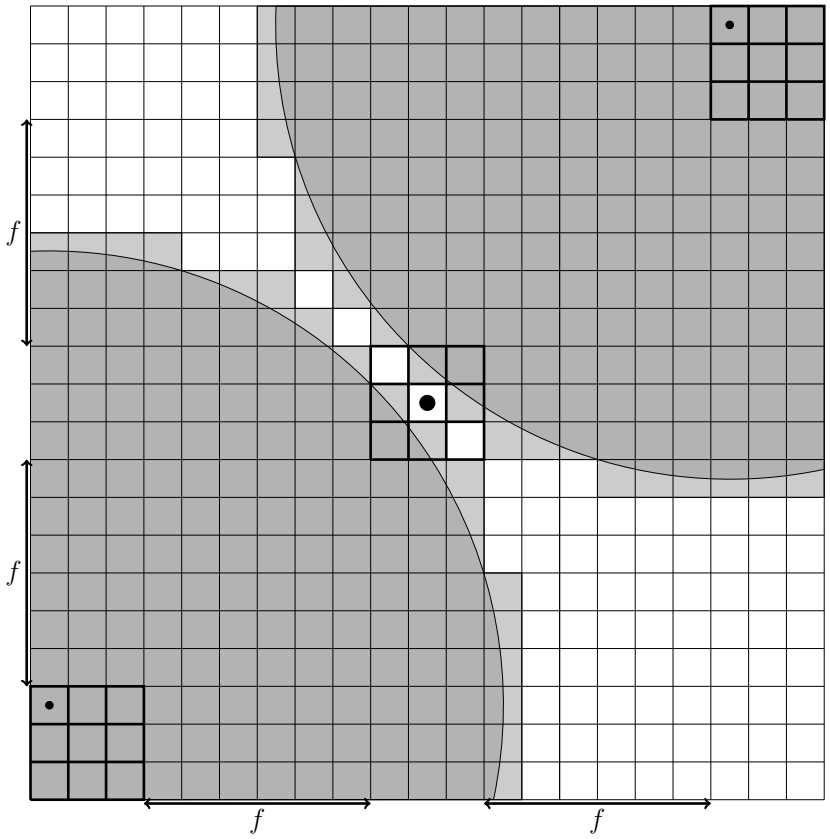

Figure 3.5: Construction of a gadget for dimension $k = 2$ and norm $\mathcal{L}_2$. The squares in bold are intersections of free coordinates. The coding point represents here the value 2. The gray areas represent the area in which a coding point would be at a distance shorter than $d_{\min}$ from one of the points either in the bottom left or in the upper right squares.

The Newton binomial theorem applied to Eq. (3.4) for $i \in [\![0, p-2]\!]$ provides us with the coefficients of $P(f)$:

$$a_i = 2\binom{p}{i} n^{p-i} - \binom{p}{i}(-1)^{p-i} - \binom{p}{i}(2n-2)^{p-i}.$$

Knowing that $\binom{p}{i} < 2^p$, we get the bound on all the coefficients:

$$|a_i| < 2^p 2^p n^p + 2^p = 2^{2p} n^p + 2^p.$$

Therefore, the choice of $f_p = 2 + \frac{2^{2p} n^p}{2p} + 1$ would lead to $d_{\min} > d_{\max}$ for all n.
We thus have a pLHD with the same properties as in the proof of Theorem 3.10. Indeed, the distance constraint forces us to put points on the diagonal of the central block. The Latin constraint prevents us from coding the same value more than once in the same line or same column. Coding points can be put on the diagonal if and only if the corresponding Latin square can be completed and therefore this construction is a reduction of the pLS-CP instances to a subset of those of the pLHD-FC-CP. Note that p is not an entry of the problem, and thus the size of M is still polynomial. □

We now go back to the original pLHD-CP.

Theorem 3.12. *The pLHD-CP is NP-complete for any norm $\mathcal{L}_p$ with $p \in \mathbb{N}^*$ and $k = 2$.*

Proof. The straightforward consequence of Theorems 3.1. and 3.11. □

3.2.6 Dimension $k = 2$ for norm $\mathcal{L}_\infty$

We proved that the pLHD-CP is NP-complete for all norms $\mathcal{L}_p$ by adapting the reduction used for norm $\mathcal{L}_1$. This result, however, cannot be extended to norm $\mathcal{L}_\infty$. While we could use the fact that for any norm $\mathcal{L}_p$ the circle has a tangent with a diagonal slope at certain points in order to adapt the reduction used for norm $\mathcal{L}_1$, this is not the case for norm $\mathcal{L}_\infty$. To prove the NP-completeness of the pLHD-CP for this norm, we reduce from a different problem.

NP-completeness for norm L_∞

The proof of the following theorem contains a reduction from the $(3, B_2)$-SAT problem which is an NP-complete variant of the 3-SAT problem, where each variable appears four times, twice negated and twice not negated [9]. As for Theorems 3.6 and 3.12, we build a reduction for the pLHD-FC-CP.

Theorem 3.13. *The pLHD-FC-CP is NP-complete for norm $\mathcal{L}_\infty$ on the plane.*

Proof. Let S be an instance of the $(3, B_2)$-SAT problem with C clauses and n variables x_i, $i = 1, 2, ..., n$. We consider a certain distance $d_{\min}$ which will be determined later. First of all, we need a specific notation to address the elements of S:

- C_j — the j^{th} clause of S,
- l_j^r — the r^{th} literal of clause C_j, $r = 1, 2, 3$,
- x_i — the i^{th} variable of S,
- x_i^m — the m^{th} occurrence of the literal x_i, $m = 1, 2$,
- $\bar{x}_i^m$ — the m^{th} occurrence of the literal $\bar{x}_i$, $m = 1, 2$.

To reduce the pLHD-FC-CP to the $(3, B_2)$-SAT problem, we use an intermediate representation for S. It will be seen as an array containing elements from $\{0, 1\}$. This array will be transformed into a pLHD M. The lines and columns of S will be reduced to free coordinates in M. Consequently, the entries of S become intersections of free coordinates in M. The '1's will become points in M, while intersections with '0' cannot accept a point as they would be too close to another point. The following construction is illustrated in Fig. 3.6.
Lines of S are numbered by clauses and their three literals. More precisely speaking, a line corresponding to C_j is followed by three lines of its literals l_j^1, l_j^2 and l_j^3. Line C_j together with lines l_j^1, l_j^2, l_j^3 form a horizontal group.
Columns are also organized in groups. Each vertical group is composed of three columns and corresponds either to the first occurrence of literals x_i and $\bar{x}_i$ or to their second occurrence. Columns of the group representing the first occurrence of x_i and $\bar{x}_i$ are noted x_i^1, c_i^1 and $\bar{x}_i^1$. We attract the reader's attention to the extra column c_i^1 in the middle. The three-column bands corresponding to the second occurrence of variable x_i are organized in the same way: x_i^2, c_i^2 and $\bar{x}_i^2$.
The vertical groups of the array are, in a natural way, aligned according to the schema: groups x_i^1, c_i^1 and $\bar{x}_i^1$ for all variables x_i, followed by the groups x_i^2, c_i^2 and $\bar{x}_i^2$ for all x_i, $i = 1, 2, ..., n$.
The necessity of transformation of S into an instance M of the pLHD-FC-CP determines the method of filling up S.
First of all, the Latin constraint has to be respected, which means that array S may contain at most a single '1' in each line and each column. Additionally, the distance constraint of

M is represented by the groups in S: two '1's cannot be located in a single intersection of a vertical and horizontal groups.
Figure 3.6 illustrates the fragment of M relative to variable x_1, which includes the free coordinates corresponding to the two groups of columns referencing x_1 and the four groups of lines corresponding to the clauses containing x_1, which we assumed to be C_1, C_2, C_3, and C_4.
Despite the fact that the lines and columns we defined are already sufficient to code the SAT instance, we need to inflate our array, composed of $4C$ lines and $6n$ columns at the moment, into a square array.
We add $3n$ lines and $n+C$ columns outside of groups, placing them arbitrarily at the "ends" of the array. They will be considered individually.
As S contains each variable four times, it means that it has $4n$ literals. We also know that each clause is composed of three literals, which means that S contains $3C$ literals, hence $4n = 3C$. We conclude that S is a square array which has $3n + 4n + C = 7n + C$ lines and columns.
We need to represent the satisfaction of a clause C_j of S. We use line C_j to this effect: it contains a '1' if the corresponding clause is satisfied by giving a logical value to the variable corresponding to the column of the '1'. We also need to avoid assigning the opposite values to the same variable.
To reach this goal, we put the value '0' in every entry of array S except the intersections between:

- line C_j and column x_i^1 if clause C_j contains x_i^1,
- line C_j and column $\bar{x}_i^1$ if clause C_j contains $\bar{x}_i^1$,
- line C_j and column c_i^2 if clause C_j contains either x_i^2 or $\bar{x}_i^2$,
- line l_j^k and column c_i^1, if literal l_j^k is either x_i^1 or $\bar{x}_i^1$,
- line l_j^k and column x_i^2, if literal l_j^k is either x_i^1 or x_i^2.
- line l_j^k and column $\bar{x}_i^2$, if literal l_j^k is either $\bar{x}_i^1$ or $\bar{x}_i^2$.

The intersections of $3n$ supplementary lines are treated differently. We do not put '0' in the intersections between:

- the first n supplementary lines and columns x_i^1,
- the next n supplementary lines and columns $\bar{x}_i^1$,
- the last n supplementary lines and columns c_i^2.

For the supplementary columns we refrain from putting '0' on the intersections between:

- the first n supplementary columns, indexed for sake of clarity x_i^3, and lines l_j^k, if literal l_j^k is either x_i^2 or $\bar{x}_i^2$,
- the last C supplementary columns and all $3n$ supplementary lines.

Suppose that S can be satisfied. As in each clause C_s there is at least one literal which makes it be satisfied, we select this literal. If it is its first occurrence (*i.e.* either x_k^1 or $\bar{x}_k^1$) we put '1' in line C_s and the column indexed by it. In contrast, if it is the second occurrence, we put '1' in column c_k^2.
Logical values assigned to literals of clause C_s are expressed by '1's in lines l_s^r and column:

- if $l_s^r = x_k^1$, column x_k^2 for $x_k = T$ and column c_k^1 for $x_k = F$,

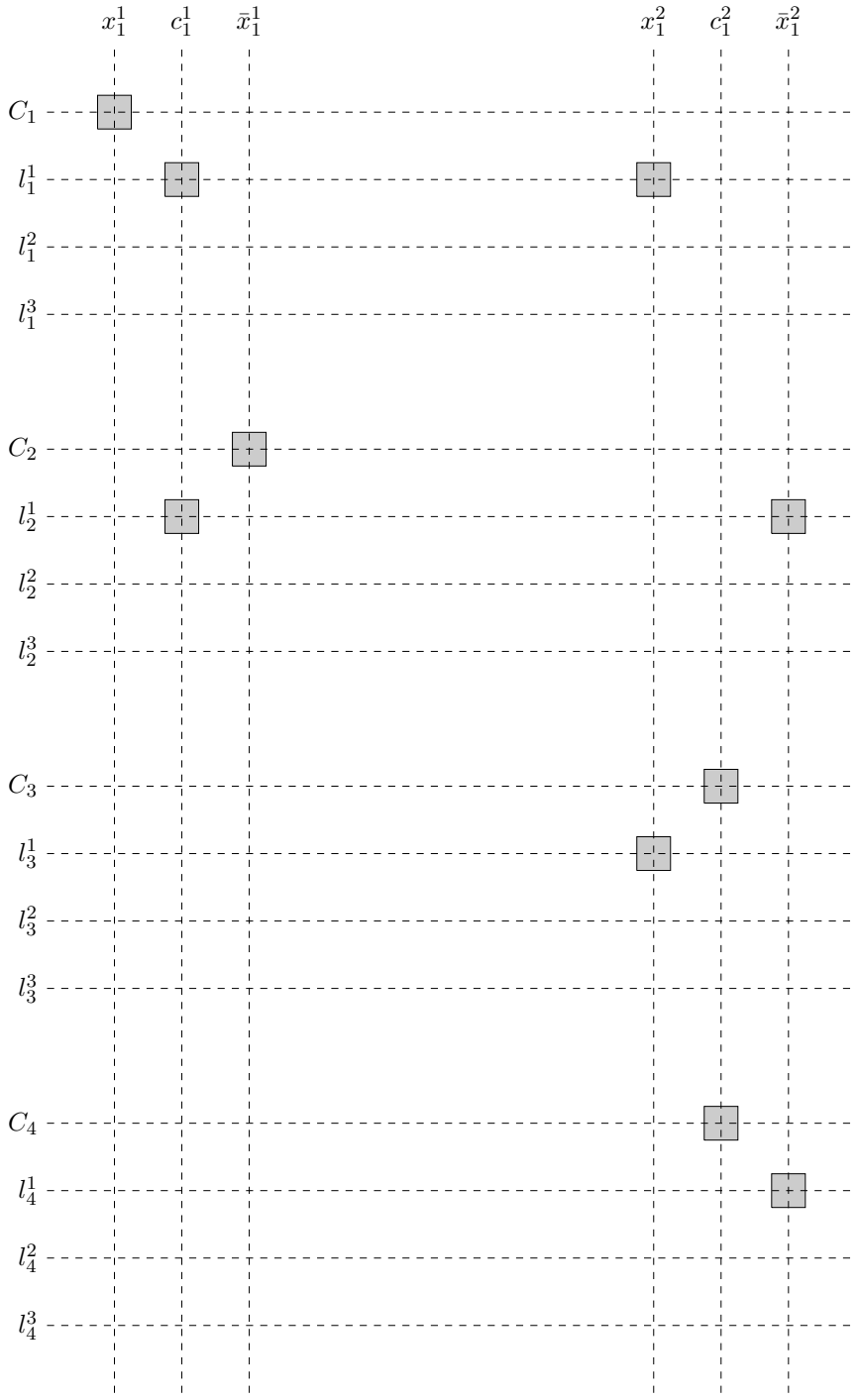

Figure 3.6: Lines, columns, and intersections for variable x_1 in S. Seen as M: dotted lines and columns are its free coordinates. Gray squares represent intersections in which it is possible to put a point in M, as the corresponding entries in S do not contain '0's.

- if $l_s^r = \bar{x}_k^1$, column c_k^1 for $x_k = T$ and column $\bar{x}_k^2$ for $x_k = F$,
- if $l_s^r = x_k^2$, column x_k^3 for $x_k = T$ and column x_k^2 for $x_k = F$,
- if $l_s^r = \bar{x}_k^2$, column $\bar{x}_k^2$ for $x_k = T$ and column x_k^3 for $x_k = F$.

In each supplementary line there is exactly one intersection either with column x_k^1 or with column $\bar{x}_k^1$ or with column c_k^2. If the corresponding column does not have a '1' already, we insert '1' in this place, otherwise we insert it in the first supplementary column without '1'. Since C columns among those numbered with x_k^1, $\bar{x}_k^1$ or c_k^2 are already filled before we start to fill the supplementary $3n$ lines, we can fit a '1' in each of them. We then put '0' in each entry which is still empty. Note that no intersection of groups of lines and columns contains more than one '1', and each line and column contains exactly one '1'.
Now suppose that we can complete array S while respecting both the Latin and distance constraints.
By construction, each '1' is in a previously empty entry. Moreover, only one '1' can be put in each intersection of groups of lines and columns. Since each line contains one '1', we have one point in each line C_s. We then assign the value T to the literals corresponding to the column of the '1' in each line noted C_s. Thus, we have an assignment that satisfies S. However, we still have to check that we did not assign T to a variable and its negation. This verification is illustrated on Fig. 3.7.
We show this by using the Latin and distance constraints multiple times to prove that if a '1' is in the intersection between line C_s and column x_k^1 (which means that we assign T to variable x_k), there is no '1' in neither the intersection between line C_u and column $\bar{x}_k^1$, nor in the intersection between line C_u and column c_2^k.
Suppose we have a '1' in line C_s in column x_k^1. We thus do not have a point in line l_s^r (if literal $l_s^r = x_k^1$) in column c_k^1, since they would be in the same intersection of groups. This means that we have a '1' in line l_u^t (if $l_u^t = \bar{x}_k^1$) and column c_k^1, since there are only two empty entries in column c_k^1. This implies that there is no '1' in line C_u in column $\bar{x}_k^1$ because this '1' is in the same intersection of groups containing l_u^t and c_k^1, which means that we cannot assign T to $\bar{x}_k^1$. This also means that there is no '1' in column $\bar{x}_k^2$ in line l_u^t. Therefore, there is a '1' in column $\bar{x}_k^2$ in line l_w^v (if $l_w^v = \bar{x}_k^2$). This means that there is no '1' in line C_w in column c_k^2 (because they are in the same intersection of groups containing column $\bar{x}_k^2$ and line l_w^v), which leads to the impossibility of assigning T to $\bar{x}_k^2$.
The same reasoning prevents us from assigning T either to x_k^1 or to x_k^2 if $\bar{x}_1 = T$.
Finally, if there is a '1' in line C_j (which contains x_k^2) and column c_k^2, there is no '1' in line $C_{j'}$ in column c_k^2 (if C_u contains $\bar{x}_k^2$), which ensures that we cannot assign T to both x_k^2 and $\bar{x}_k^2$.
After excluding the possibility of assigning the opposite logical values to a variable, we build a pLHD M from array S. We construct it in the following manner: each line and each column from S is represented by a free coordinate. Two lines (or columns) inside the same group are $(d_{\min}, 1)$-close (or $(d_{\min}, 2)$-close), according to Definition 3.3, with the number 1 attributed to the vertical axis and the number 2 attributed to the horizontal axis, which means that these free coordinates differ by less than $d_{\min}$. Additionally, we forbid each intersection whose corresponding entry in S contains a '0' by adding a point at a distance lower than $d_{\min}$ from all these intersections. Another way to see it is that we need to cover these intersections with circles of radius $d_{\min}$, which are, as a matter of fact, squares for norm $\mathcal{L}_\infty$. We thus have to cover the forbidden intersections with squares such that the center of each square is not covered by any other square.
The method of covering intersections depends on the configuration of the group of lines and the group of columns the intersections belong to.
Lines come either in groups of 4 or alone. Similarly, columns come either in groups of 3 or alone. Obviously, the configurations of intersections are: 4×3, 4×1, 1×3, and 1×1.

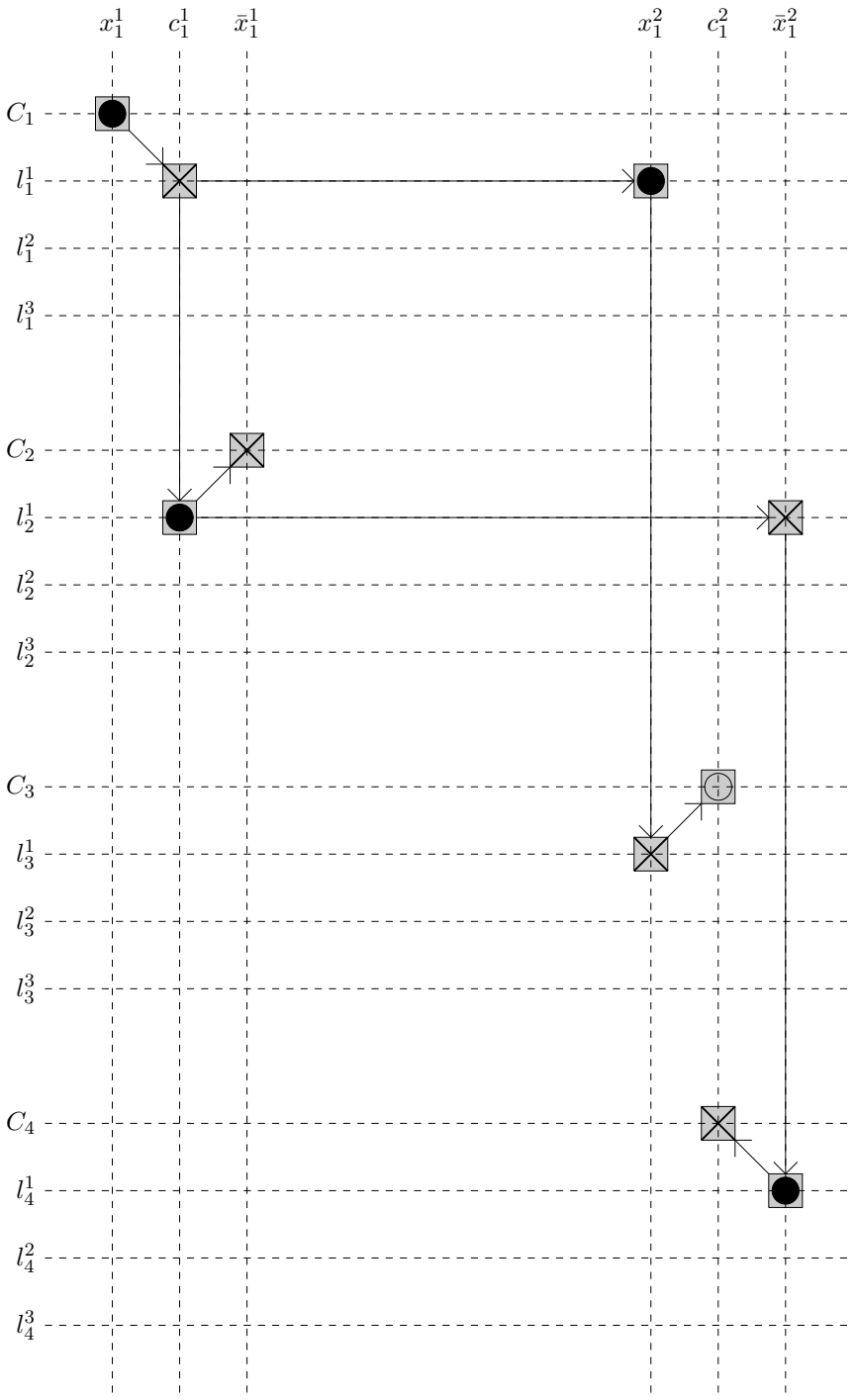

Figure 3.7: Lines, columns, and intersections for variable x_1 in M. We verify that assigning T to x_1 and using it to satisfy clause C_1 (putting a point in line C_1 and column x_1^1) prevents from using it to satisfy clauses C_2 and C_4 containing its negation. The arrows represent constraints, vertical and horizontal arrows representing the Latin constraint and diagonal arrows representing the distance constraint. A solid black dot inside an intersection means that a point has to be put in it. A cross means that no point can be put in the intersection. An empty circle means that a point can be put in the intersection but is not necessary.

The last three configurations allow at most one intersection. We cover them using either one square to cover the whole configuration if no intersection is allowed or two squares if one intersection is allowed, this allowed intersection is produced by the edges of both squares.
The last type is the 4×3 configuration, obtained by the intersection of groups of the initial $4C$ lines and $6n$ columns. In configurations of this type, we have either none, one or two allowed intersections, which are on different lines and columns. Additionally, if two intersections are allowed, one of those is in the upper line (C_j) of the group. Since the configurations are three lines wide and we have two points on different columns, one of the points is on the border of the configuration.
If no intersection is allowed, we can put one point in the middle of the configuration. Since each line and each column of this configuration are either $(d_{\min}, 1)$-close or $(d_{\min}, 2)$-close, the distance between them is smaller than $d_{\min}$, and thus each intersection of this group will be covered by that point.
If one and only intersection is allowed, we put one point in each direction at a distance $d_{\min}$ from the allowed point, on a line or column $(d_{\min}, 1)$-close (or $(d_{\min}, 2)$-close) to the allowed point. Each intersection but the allowed one is at a distance less than $d_{\min}$ from at least one of the added points.
If two intersections are allowed, we put two points such that the square of size $d_{\min}$ is tangent to both of them. If the allowed intersection in line C_j is not in the center column, we put the squares in such a way that they are both tangent to the second intersection, one vertically and the other horizontally. This case is illustrated on Fig. 3.8. If the allowed intersection in line C_j is in the center column, we put one square vertically tangent to it, and the last square horizontally tangent to the second intersection. This case is illustrated on Fig. 3.9.
We still have to choose $d_{\min}$. Since we have to put at most four points to cover each group, and we have $(3n + C)^2$ groups, we can choose $d_{\min} = 4(3n + C)^2 + 1$.
Since pLHD M respects the same constraints as S, it can be completed with separation distance $d_{\min}$ if and only if S can be satisfied. We have a reduction from $(3, B_2)$-SAT to the pLHD-FC-CP. □

The equivalence of the pLHD-FC-CP and the pLHD-CP allows us to formulate the following:

Theorem 3.14. *The pLHD-CP is NP-complete for norm $\mathcal{L}_\infty$ on the plane.*

Proof. The straightforward consequence of Theorems 3.1 and 3.13. □

3.3 Conclusion

We proved the NP-completeness of the pLHD-CP for the following cases: for all norms in dimensions $k \geq 3$, and on the plane for norms $\mathcal{L}_\infty$ and norms $\mathcal{L}_p$, with $p \in \mathbb{N}^*$. This ensures that , unless P=NP, no polynomial-time exact algorithm exists to solve this problem. In the next section, we continue this study with the search for performance guarantees, including upper bounds and approximation algorithms. On the one hand, we show an approximation algorithm for the LHD-CP and two new upper bounds we use to prove the approximation. On the other hand, we prove that no approximation algorithm exists for the pLHD-CP in dimension $k \geq 3$. We also give a bound for the approximation ratio of an approximation algorithm for completing a pLHD in the plane for norm $\mathcal{L}_\infty$.

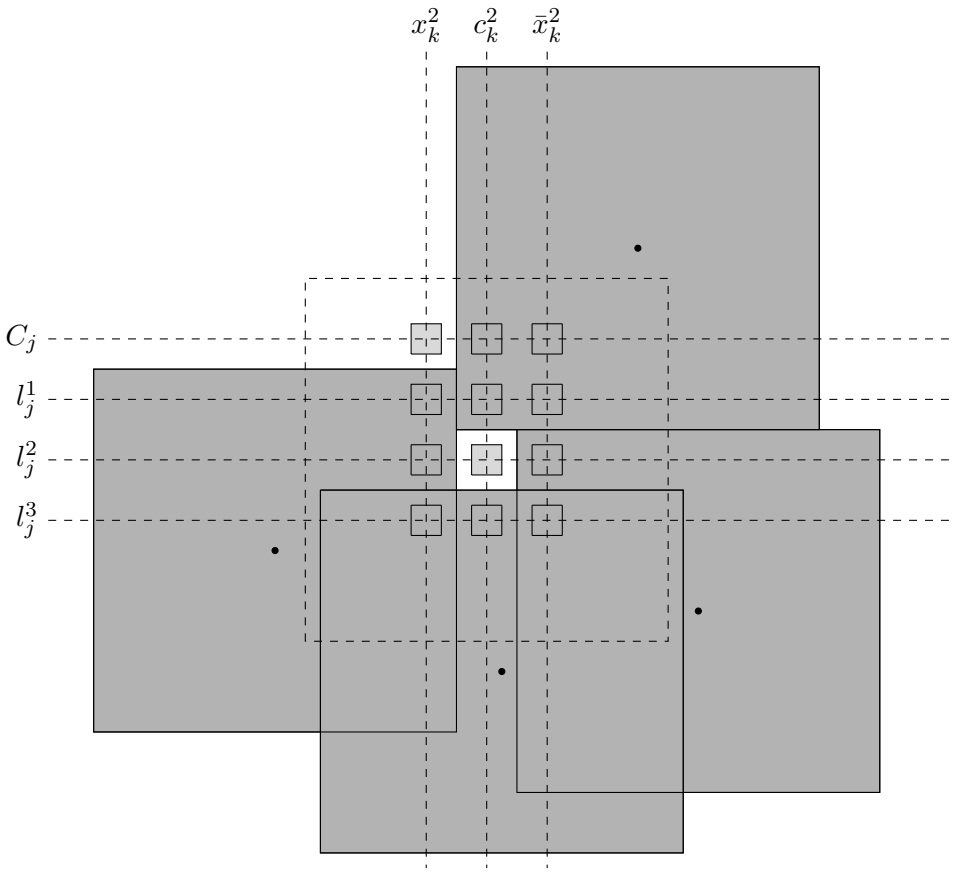

Figure 3.8: Configuration 4×3 of intersections covered by four gray squares (circles according to $\mathcal{L}_\infty$). The dashed square is a circle of radius $d_{\min}$ centered in the allowed intersection between c_k^2 and l_j^2: the centers of the four covering squares are outside of it.

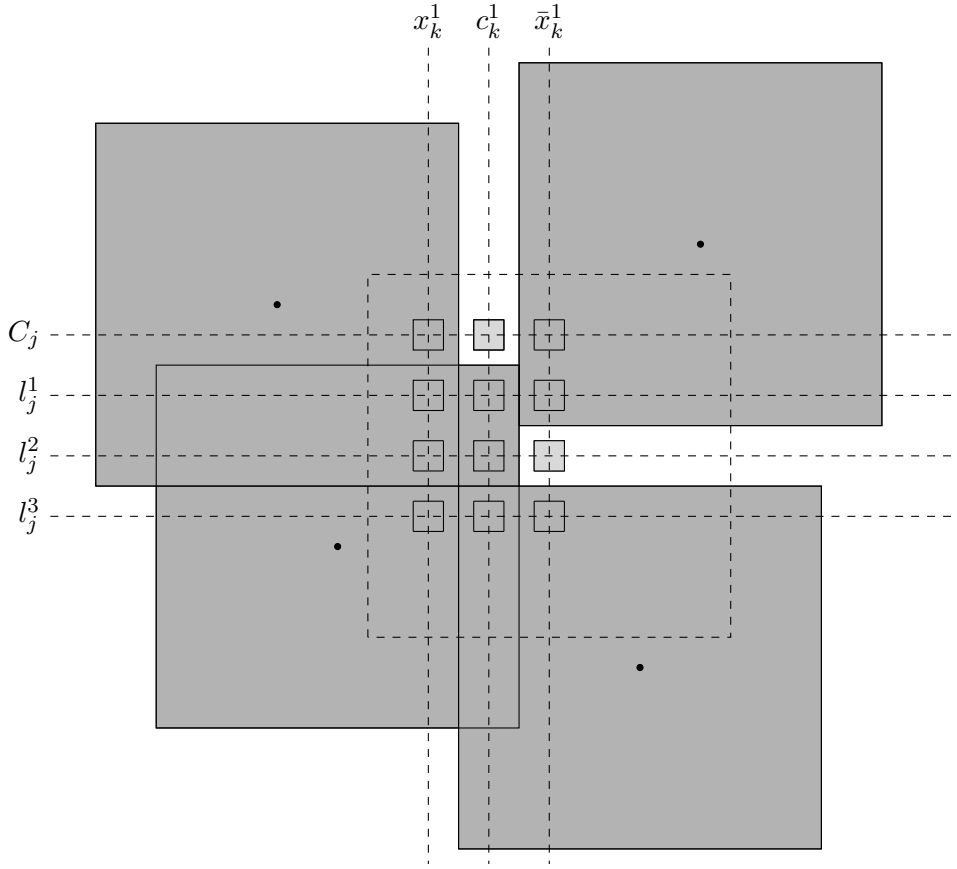

Figure 3.9: Configuration 4×3 of intersections covered by four squares (circles according to $\mathcal{L}_\infty$). The dashed square is a circle of radius $d_{\min}$ centered in the allowed intersection between $\bar{x}_k^1$ and l_j^2: the center of the covering squares are outside it.

Chapter 4

Guarantees of performance for algorithms

After proving the NP-completeness of the pLHD-FC-CP and the pLHD-CP, we search for guarantees of performances of algorithms for both problems, in the form of upper bounds and approximation ratios. We start by finding out that the pLHD-CP and the pLHD-FC-CP are inapproximable for dimensions $k \geq 3$, and we give an upper bound for an approximation ratio in dimension $k = 2$ for norm $\mathcal{L}_\infty$. These results are summarized in Table 4.1 [48]. We follow by studying the LHD-CP, focusing on norm $\mathcal{L}_2$. We show two new upper bound and use them to prove an approximation algorithm [49].

Norm	Dimension	Approximability
Any norm	$k \geq 3$	No $\epsilon-$approximation for $\epsilon > 0$
Norm $\mathcal{L}_\infty$	$k = 2$	No $\epsilon-$approximation for $\epsilon \geq \frac{2}{3}$
Other norms		No results

Table 4.1: Approximability of the pLHD-CP

4.1 State of the art

Optimal algorithms have been designed for the LHD-CP on the plane for norms $\mathcal{L}_1$ and $\mathcal{L}_\infty$ in [51], making this problem P for this subcases. Both algorithms have a linear complexity $\mathcal{O}(n)$, and are based on a regular covering: each point is placed in a regular manner, along almost horizontal lines for norm $\mathcal{L}_\infty$ and along diagonal lines for norm $\mathcal{L}_1$. The authors of [51] also proposed an upper bound for $\mathcal{L}_2$ by applying Oler's theorem [40] to the square. However, no polynomial optimal algorithm has been found, nor any approximation algorithm.
In [52], numerous upper bounds have been formulated for norms $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_\infty$. For each of these norms, an upper bound tight when the number of dimensions is high relatively to the size of the LHD, has been proposed. The bound for norm $\mathcal{L}_2$ was found thanks to the fact that the sum of squared distances, and thus their mean, is constant among all LHDs of the same size and dimension. The squared minimal distance has to be lower than the mean. The bound obtained is:

$$d_{\min}^2 \leq \left\lfloor \frac{n(n+1)k}{6} \right\rfloor.$$

The same principle is used to find an upper bound for norm $\mathcal{L}_1$, without squaring the distance. This gives:

$$d_{\min} \leq \left\lfloor \frac{(n+1)k}{3} \right\rfloor.$$

A similar idea is used for norm $\mathcal{L}_\infty$ by counting the number of points that can be separated by a certain distance on each dimension, and stating that this number multiplied by the

number of dimensions has to be higher than the number of points. The minimal distance has to respect the following inequality:

$$k(n - d_{\min})(n - d_{\min} + 1) \geq n(n - 1).$$

Additionally, the authors of [52] have shown an upper bound for $\mathcal{L}_\infty$ that is tight when the size n of the LHD is close to b^k, b being a positive integer, by using a bound for unrestricted maximin designs found in [5], obtaining that

$$d_{\min} \leq \left\lfloor \frac{n-1}{\sqrt[k]{n-1}} \right\rfloor.$$

They also proposed an optimal algorithm for the case where $n = b^k$, based on a mapping from $[\![0, b-1]\!]^k$ to an LHD. Moreover, they constructed an upper bound for the three-dimensional space. Their idea to this construction is as follows: if an LHD has a separation distance d, then the projection on two dimensions of a layer of width $d-1$ (*i.e.* all the points whose third coordinate is between some value m and $m + d - 1$) is an incomplete LHD with separation distance d.
While several bounds have been found and algorithms have been developed for specific cases, notably norms $\mathcal{L}_1$ and $\mathcal{L}_\infty$, no approximation algorithm has been found for the general case of the construction problem. The completion problem was never studied before. We continue with the study of its approximability.

4.2 Completion

We prove that the pLHD-CP is inapproximable for all norms in dimensions $k \geq 3$. We first prove this fact for norm $\mathcal{L}_1$ and use the norm equivalence as we did in the proof of Theorem 3.8 to extend this result to all norms. We next give an upper bound for an approximation ratio in dimension $k = 2$ for norm $\mathcal{L}_\infty$ [48].

4.2.1 Dimension $k \geq 3$

The completion problem does not admit an approximation algorithm in spaces with norm $\mathcal{L}_1$. To prove this fact we first prove the inapproximability of the completion problem with forbidden coordinates (the max-pLHD-FC-CP).

Theorem 4.1. *There is no polynomial ε-approximation algorithm for the max-pLHD-FC-CP for $k \geq 3$ and norm $\mathcal{L}_1$ for any $\varepsilon > 0$.*

Proof. The proof is based upon the *reductio ad absurdum* argumentation. Let us suppose that there is an ε-approximation algorithm for this problem. Let S be an instance of pLS-CP. We construct a corresponding instance M of the completion problem according to the scheme described in the proof of Theorem 3.6 taking

$$d_{\min} = \frac{(k-3)(n^6 + n^2) + 3n^4}{\varepsilon} + 1.$$

If the result of the approximation algorithm has a separation distance

$$\Delta > (k-3)(n^6 + n^2) + 3n^4,$$

there will be one and only one point on the diagonal of each block. Consequently, it is possible to complete the Latin square. If the result has a separation distance

$$\Delta \leq (k-3)(n^6 + n^2) + 3n^4,$$

the best separation distance we can attain is

$$\Delta^* \leq \frac{(k-3)(n^6+n^2)+3n^4}{\varepsilon} < d_{\min}.$$

In this case we cannot obtain the pLHD with a separation distance $\Delta = d_{\min}$, which means that we cannot complete the Latin square S. □

Next, we use the theorem above to prove the following one which addresses the max-pLHD-CP inapproximability.

Theorem 4.2. *There is no polynomial ε-approximation algorithm for the max-pLHD-CP for $k \geq 3$ and norm $\mathcal{L}_1$ for any $\varepsilon > 0$.*

Proof. The consequence of Theorems 3.1 and 4.1.

□

4.2.2 Dimension $k = 2$, norm $\mathcal{L}_\infty$

We follow the same methodology as in the previous section: the main theorem (Theorem 4.3) is formulated for the completion problem with forbidden coordinates. The theorem concerning the completion problem (Theorem 4.4) is its direct consequence thanks to Theorem 3.1. The reduction described in Theorem 3.13 can be modified to produce an upper bound on the ratio of an approximation algorithm solving the max-pLHD-FC-CP on the plane for norm $\mathcal{L}_\infty$.

Theorem 4.3. *There is no polynomial $\left(\frac{2}{3}+\varepsilon\right)$-approximation algorithm for the max-pLHD-FC-CP for $k = 2$ and norm $\mathcal{L}_\infty$ for any $\varepsilon > 0$.*

Proof. Once again, the proof is based upon the *reductio ad absurdum* argumentation. Let S be an instance of the $(3, B_2)$-SAT. We suppose that we have a polynomial-time approximation algorithm for the max-pLHD-FC-CP with an approximation ratio ε, $\varepsilon > \frac{2}{3}$. We show that this approximation algorithm could be used to solve $(3, B_2)$-SAT.
We build M, a pLHD with the same structure as the one described in the proof of Theorem 3.13, except that we transpose lines C_j with lines l_j^1 and do not choose $d_{\min}$ yet. We need to proceed with a series of verification to show that the algorithm, whose existence was assumed, gives the optimal solution. The Latin constraint still holds, thus we only need to verify the distance constraints. This means that we need to check that we can still cover the intersections which cannot contain a point (corresponding to '0' in array S defined in the proof of Theorem 3.13), and that each intersection configuration can only contain a single point. Next, we show that we can still cover the 4×3 configurations modified by the aforementioned line transposition.
To verify the covering we observe that we can have none, one or two free intersections. The first two cases are exactly the same as without the line transposition, and we cover them as we did in the proof of Theorem 3.13. In the third case, we distinguish two sub-cases: either the two free intersections are on lines C_j and l_j^2, or they are somewhere else. In the latter, the cover can be made with four squares (which are actually circles with $\mathcal{L}_\infty$), as it is equivalent to the covering used in the aforementioned proof and illustrated in Figs. 3.8 and 3.9. In the first sub-case, we need five points to cover all forbidden intersections, as shown in Fig. 4.1.
After describing the covering of intersections that cannot contain a point, we study the separation distance given by the approximation algorithm whose existence is supposed.
We note d the distance between two consecutive free coordinates inside a group (of columns or of lines). We can choose any value for d under the condition that we have enough space to put the covering points. Since we have at most five points to cover each configuration and there is $(3n+C)^2$ configurations to be covered, we can choose any $d > 5(3n+C)^2$. We

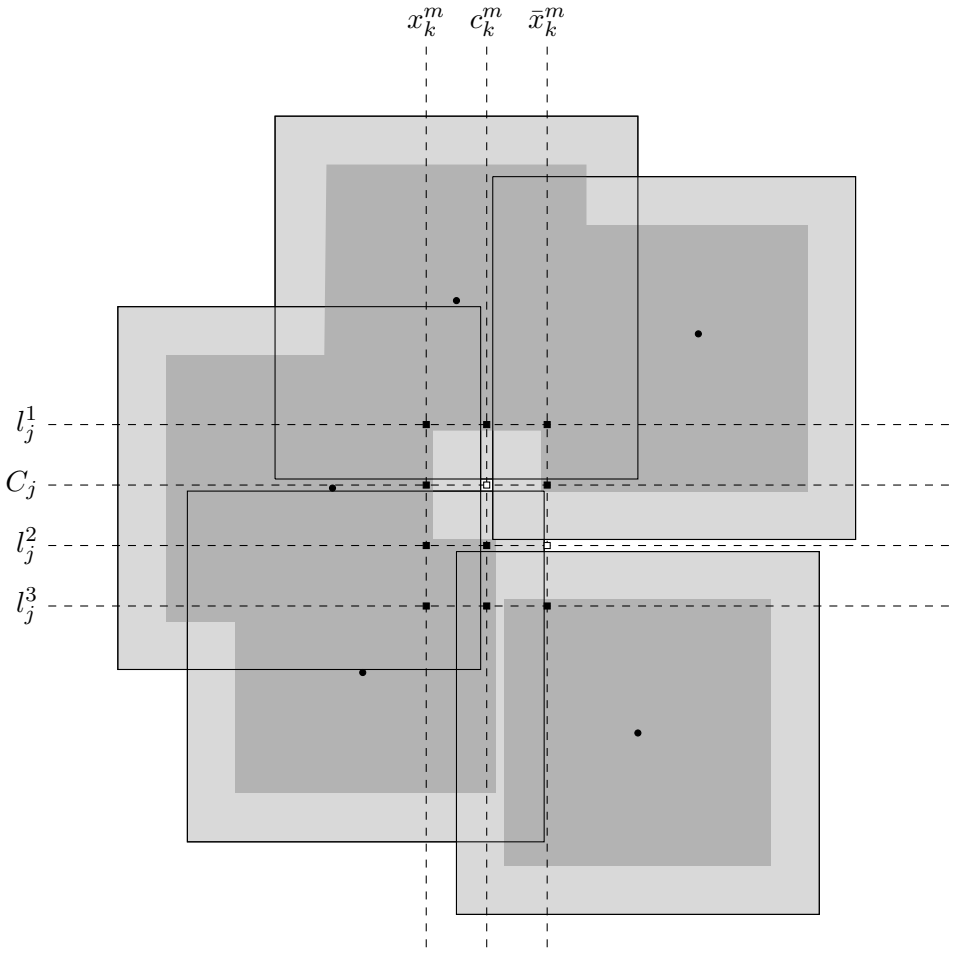

Figure 4.1: Configuration 4×3 of intersections covered by five squares (circles according to $\mathcal{L}_\infty$). The order of lines l_j^1 and C_j follows the construction made in the proof of Theorem 4.3. The light gray areas represent circles of radius $d_{\min}$, while dark gray areas represent circles of radius $\varepsilon d_{\min}$. Points representing intersections are covered by either both or none of the two circles.

also fix $d_{\min} = 3d$. We now suppose that the minimum distance given by the approximation algorithm is $d_a < d_{\min}$. This result may be produced by one of these two methods:

- adding two points close to each other (*i.e.* at a distance $d_a < d_{\min}$ from each other) and being in the same configuration, separated by at most $2d$,
- adding a point close to a covering point (*i.e.* at a distance $d_a < d_{\min}$ from it) , in which case they are separated by a distance $2d + c$ with $c \leq 5(3n + C)^2$.

We thus obtain using the second case:

$$\frac{d_a}{d_{\min}} \leq \frac{2d + c}{3d} = \frac{2}{3} + \frac{c}{3d}.$$

After setting $d = \frac{5(3n+C)^2}{3(\varepsilon - \frac{2}{3})}$ we get:

$$\frac{d_a}{d_{\min}} \leq \frac{2}{3} + \frac{15(3n + C)^2(\varepsilon - \frac{2}{3})}{15(3n + C)^2},$$

and after simplification:

$$\frac{d_a}{d_{\min}} < \frac{2}{3} + \left(\varepsilon - \frac{2}{3}\right) = \varepsilon.$$

Now let $d_{\max}$ be the maximum separation distance for instance M. Since $d_a \geq \varepsilon d_{\max}$, then $d_{\max} < d_{\min}$.
Suppose that S is a positive instance of $(3, B_2)$-SAT. The construction ensures that $d_{\max} \geq d_{\min}$ and we have just shown that the separation distance found by the algorithm is $d_a \geq d_{\min}$. Similarly, if S is a negative instance, the separation distance found by the algorithm is $d_a \leq d_{\max} < d_{\min}$.
Consequently, this approximation algorithm is a polynomial time algorithm for $(3, B_2)$-SAT which is impossible unless P=NP.

□

Theorem 4.4. *There is no polynomial $\left(\frac{2}{3} + \varepsilon\right)$-approximation algorithm for the max-pLHD-CP for $k = 2$ and norm $\mathcal{L}_\infty$ for any $\varepsilon > 0$.*

Proof. The consequence of Theorems 3.1 and 4.3. □

4.3 Construction

After studying guarantees of performance for the completion problem, we study the maximin LHD construction, concentrating on norm $\mathcal{L}_2$. We start by giving two upper bounds, before presenting an approximation algorithm and prove its approximation using the bounds we designed [49].

4.3.1 Bounds

We give two upper bounds, both based upon the idea of representing an LHD by non-overlapping hyperspheres whose centers are the points of the LHD. We then compare the volume of these hyperspheres with the volume of a hypercube that contains them (see Fig. 4.2). The first bound is valid for all LHD of size large enough compared to its dimension, while the second is valid for LHDs of all sizes and dimensions.

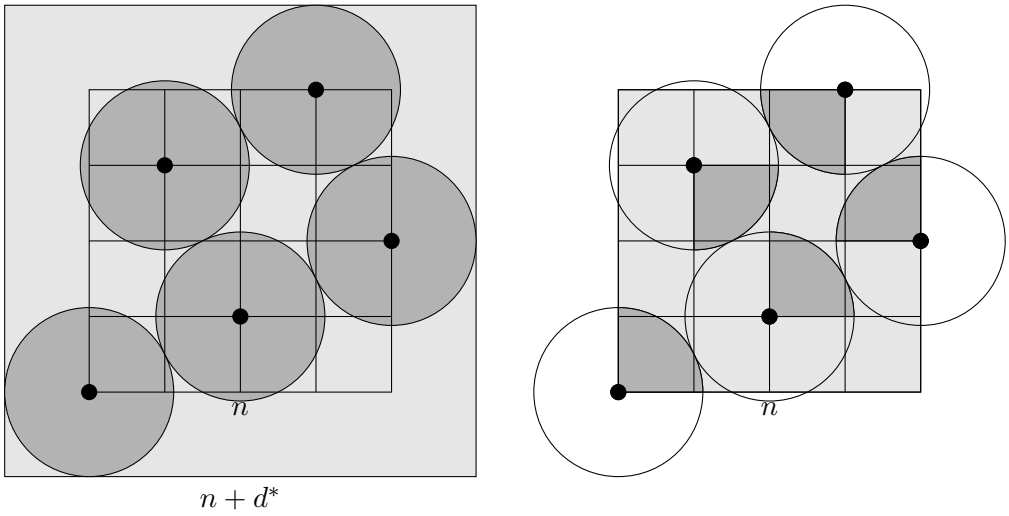

Figure 4.2: Illustration of the construction of the bounds for $n = 5$, $k = 2$, for large LHD on the left (Section 4.3.1), for any size LHD on the right (Section 4.3.1). The bounds are obtained by observing that the dark gray area is smaller than the light gray area.

Bound for large-size LHD.

Let L be a maximin LHD of size n and dimension k, with a separation distance d^*. We consider n hyperspheres of dimension k, with radius $\frac{d^*}{2}$ and centered on the points of L, as shown in Fig. 4.2. We construct a hypercube of size $n + d^*$ and dimension k, which contains the hyperspheres. We compare the sum of the volumes of these hyperspheres with the volume of this hypercube. In Fig. 4.2, on the left, the light gray color indicates the volume of the hypercube and the dark gray color is attributed to the volume of the spheres. Thus we have the following inequality:

$$(n + d^*)^k \geq nC(k) \left(\frac{d^*}{2}\right)^k,$$

where $C(k)$ is the volume of the unit sphere of dimension k. After simplification and taking the k^{th} root we obtain:

$$n + d^* \geq \frac{d^*}{2} \sqrt[k]{nC(k)}.$$

Dividing both sides of the equation above by $\sqrt[k]{C(k)} - \frac{2}{\sqrt[k]{n}}$, we get the bound we wanted:

$$d^* \leq \frac{2n^{\frac{k-1}{k}}}{\sqrt[k]{C(k)} - \frac{2}{\sqrt[k]{n}}} \underset{n\to\infty}{\sim} \frac{2n^{\frac{k-1}{k}}}{\sqrt[k]{C(k)}}. \tag{4.1}$$

The consequence of this division is that this bound is only valid for high values of n, such that $n > \frac{2^k}{C(k)}$. This is due to the fact that when n grows and k is fixed, n grows faster than d^*. On the other hand, when k grows and n is fixed, the volume of the outer hypercube grows faster than the total volume of the hyperspheres. The volume of an hypersphere is $C(k) = \frac{\pi^{\frac{k}{2}}}{\Gamma(\frac{k}{2}+1)}$ [46].

Upper bound for LHD of any size.

We make the same assumptions as in the previous subsection. This time, however, we consider a smaller hypercube, of size n and dimension k. We represent this on the right of Fig. 4.2. The light gray shading represents the volume of the hypercube, and the dark gray

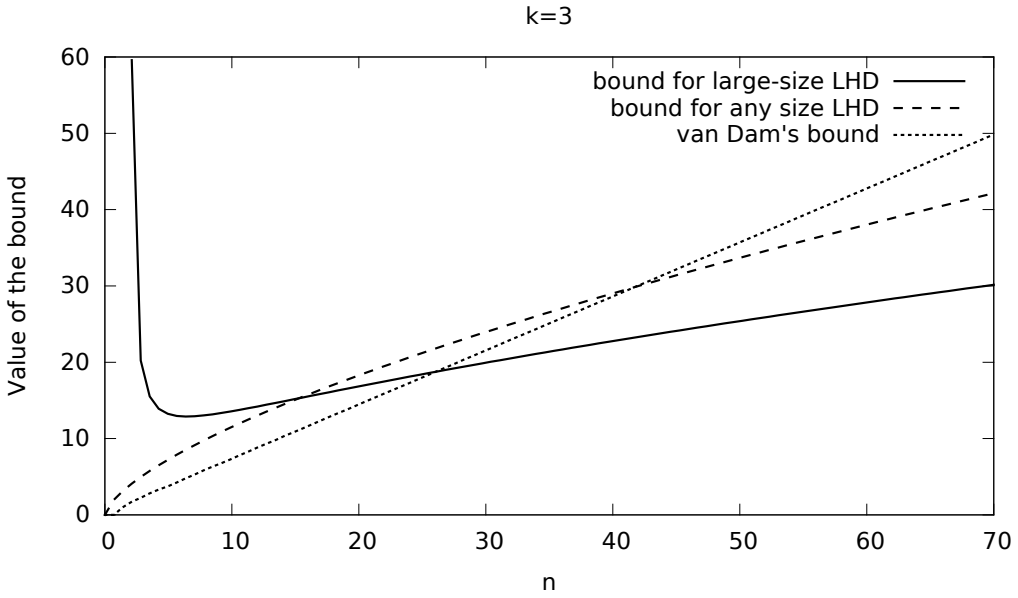

Figure 4.3: Values of the bounds for $k = 3$

shading represents the part of the volume of the spheres that is always inside the hypercube. The worst bounding arises when exactly $\frac{1}{2^k}$ of the hypersphere volume is contained inside the hypercube. This phenomenon occurs when a sphere is centered in a corner of the hypercube (the example on the right of Fig. 4.2 illustrates the worse bounding). We then compare the volume of the parts of hyperspheres lying inside the hypercube with the volume of the hypercube itself. We get the following inequality:

$$n^k \geq n\frac{1}{2^k}C(k)\left(\frac{d^*}{2}\right)^k,$$

which leads to

$$n \geq \frac{d^*}{4}\sqrt[k]{nC(k)}.$$

The bound is thus

$$d^* \leq \frac{4n^{\frac{k-1}{k}}}{\sqrt[k]{C(k)}}. \tag{4.2}$$

Comparison of the bounds.

In [52], the following upper bound was proved:

$$d^* \leq \sqrt{\left\lfloor \frac{n(n+1)k}{6} \right\rfloor}. \tag{4.3}$$

It was obtained using the average distance between points of an LHD. We compare this bound, which we refer to as *van Dam's bound*, with our volume-based bounds.
We observe in Figs. 4.3 and 4.4 that regardless of the dimension, the bound for a large-size LHD is better for high values of n. Van Dam's bound is lower for small values of n with regard to k. Depending on the dimension k, the bound for an LHD of any size can be better than the two other bounds for intermediate values of n. This is explained by the fact that van Dam's bound is of $\mathcal{O}(n)$, so while it is tight for low values of n, it will become worse than the other two which are of $\mathcal{O}(n^{\frac{k-1}{k}})$. The bound for LHDs of any size is obviously worse than

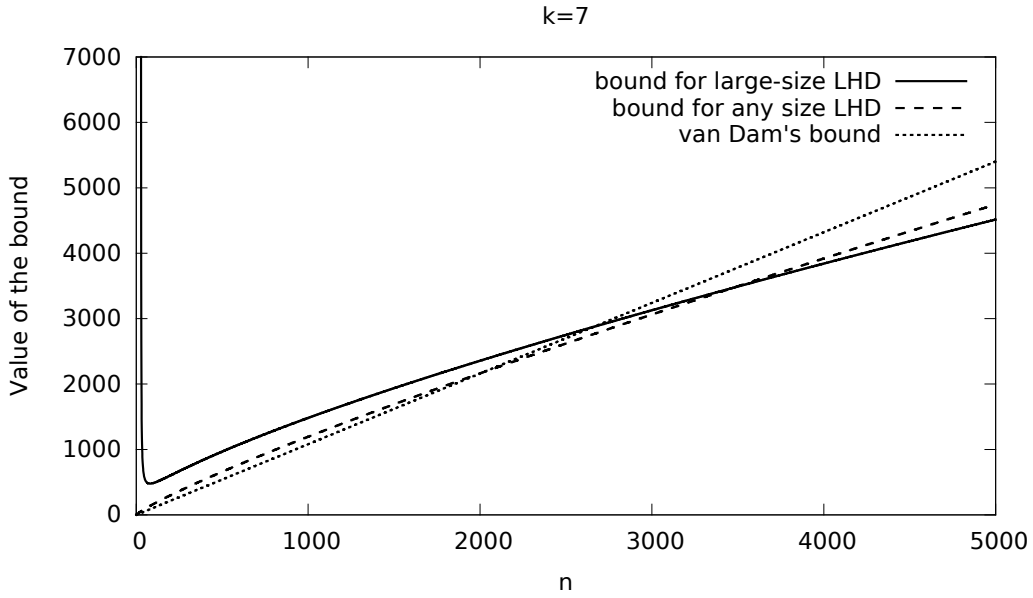

Figure 4.4: Values of the bounds for $k = 7$

the bound for large-size LHD when n takes high values. However, it outperforms the large-size LHD bound when the latter's bounding hypercube contains a lot of space unoccupied by hyperspheres (see Fig. 4.2). This occurs for intermediate values of n if $k \geq 7$.

After showing these upper bounds, we describe an approximation algorithm. We start by giving an algorithm for specific cases, prove it and give its approximation ratio. We then extend it to all cases, and compare it with the best heuristic algorithms to produce maximin LHDs described in the literature.

4.3.2 IES algorithm

We address a sub-problem of the maximin LHD problem in which $n = b^k$, where b and k are natural numbers. Algorithm 1, that we call Inflate, Expand and Stack (IES), produces solutions whose quality can be guaranteed (Subsection 4.3.4). It will be a starting point to design algorithms solving the maximin LHD problem for any k and n presented in Section 4.4.

Description of the IES algorithm

Simply put, the algorithm recursively constructs a smaller LHD of one dimension less, and uses it as a core layer which will be duplicated and stacked in order to obtain the final LHD. The base case for this recursion is the two-dimensional LHD. In this case, we use a regular grid. We rotate the grid counterclockwise around its center with angle $\arctan \frac{1}{b}$ in order to have an LHD (see Fig. 4.5). This operation is handled in lines 2–5 of Algorithm 1. The existence of such a regular grid is ensured by our assumption $n = b^2$.
For the recursive case, we use b layers of size b^{k-1}. Each of these layers will be constructed recursively, using an LHD of size b^{k-1} and dimension $k - 1$.
To construct the first layer, we start by taking an LHD of size b^{k-1} and dimension $k - 1$, obtained recursively. We then add to each point a coordinate whose value is duplicated from the last coordinate of that point. We call this phase the **inflate phase** (line 9 of Algorithm 1), as we add one dimension.
Next, we expand each layer by multiplying each coordinate but the last by b. We call this the **expand phase** (line 12) as the size of layers increases which expands distance between

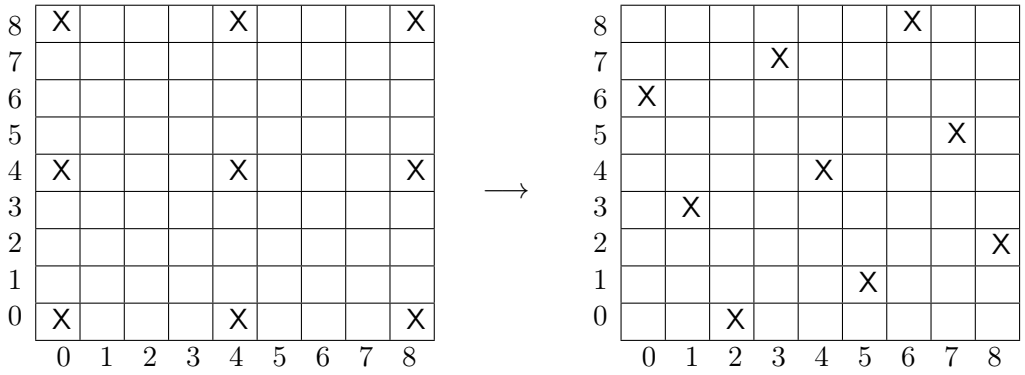

Figure 4.5: Algorithm for $k = 2$ and $n = 3^2 = 9$ applied to the grid on the left produces the grid on the right. The grid is rotated counterclockwise around $(4, 4)$ with angle $\arctan \frac{1}{3}$.

points.
Finally, we stack the layers by copying the first layer and adding an offset to each coordinate. For the $k - 1$ first coordinates, the offset is the index of the layer, with the layer's indexing starting with zero. For the last coordinate, the offset will be b^{k-1} times the index of the layer. We call this the **stack phase** (line 14). As the reader might already observe, layers are piled up in such a way that a separation distance is preserved between the points of consecutive layers.
The algorithm is illustrated using the example in three-dimensional space, $k = 3$, $b = 2$, $n = b^3 = 8$, shown in Fig. 4.6. To be able to represent a three-dimensional LHD on a two-dimensional surface, we use the first two coordinates of a point to find its position on the surface. The third coordinate is explicitly written. It may be seen as "the height" of the point in the cube. The objects in bold typeface highlight the transformations used by the algorithm.
Let us take the point $(2, 3)$. We first give the two-dimensional point a third dimension (inflate phase). This point then becomes $(2, 3, 3)$. The expand phase places the point on a bigger LHD. In our example, the point coordinates become $(2 \times 2, 3 \times 2, 3) = (4, 6, 3)$. We then use this point multiple times during the stack phase. We then obtain in our example two points, $(4 + 0, 6 + 0, 3 + 0 \times 4) = (4, 6, 3)$ and $(4 + 1, 6 + 1, 3 + 1 \times 4) = (5, 7, 7)$, which correspond to 3 and 7 in the upper right-hand corner of Fig. 4.6.

Complexity of the algorithm

We note $C(n, k)$ the time-complexity of the algorithm in function of n and k. For $k = 2$, we set b points with a constant number of arithmetic operations. Therefore, as $n = b^2$

$$C(b^2, 2) = \mathcal{O}(b^2).$$

For the general case, $k > 2$, we run the algorithm for IES$(b^{k-1}, k - 1)$ and then have a constant number of operations for all k coordinates of all $n = b^k$ points. Thus $C(b^k, k) = C(b^{k-1}, k - 1) + \mathcal{O}(kb^k)$ which gives $C(b^k, k) = \mathcal{O}(k^2 b^k)$. Put differently:

$$C(n, k) = \mathcal{O}(nk^2). \tag{4.4}$$

4.3.3 Separation distance

The IES algorithm returns an LHD whose separation distance (for norm $\mathcal{L}_2$) $d_{\min}$ can be written as a function of n and k, bearing in mind that $n = b^k$.

$$d_{\min} = \sqrt{n^{\frac{2(k-1)}{k}} + k - 1}. \tag{4.5}$$

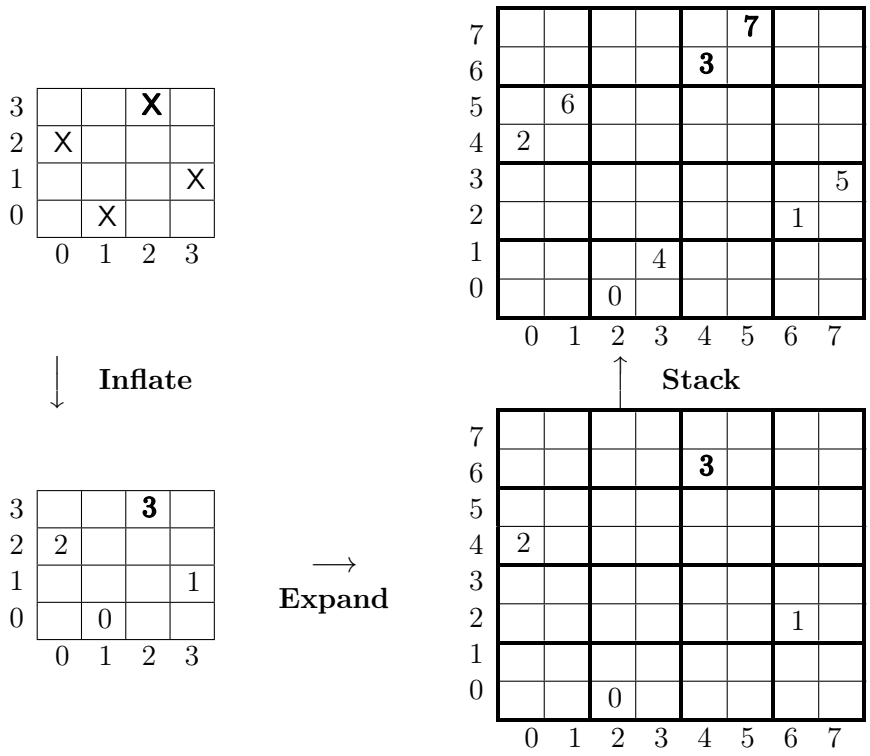

Figure 4.6: Algorithm in three dimensions for $n = 2^3 = 8$, $b = 2$. Point $(2, 3)$ is gradually transformed into $(2, 3, 3)$ by inflation, to $(4, 6, 3)$ by expansion and to $(4, 6, 3)$ and $(5, 7, 7)$ by stacking.

Algorithm 1 IES(n,k)

Input: Size n and dimension k respecting $n = b^k$, $b \in \mathbb{N}$, $k \geq 2$

Output: An LHD with separation distance $d_{\min} = \sqrt{n^{\frac{2(k-1)}{k}} + k - 1}$

$X \leftarrow \varnothing$
if $k = 2$ **then**
  for $i \in [\![0, b-1]\!]$ **do**
    for $j \in [\![0, b-1]\!]$ **do**
      $X \leftarrow X \cup \{(b(i+1) - (j+1), i + bj)\}$
else
  $Y \leftarrow$IES(b^{k-1},$k-1$)
  $Layer \leftarrow \varnothing$
  for $x_i \in Y$ **do**
    $Layer \leftarrow Layer \cup \{(x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(k-1)}, x_i^{(k-1)})\}$
  $ExpLayer \leftarrow \varnothing$
  for $x_i \in Layer$ **do**
    $ExpLayer \leftarrow ExpLayer \cup \{(bx_i,^{(0)}, bx_i^{(1)}, \ldots, bx_i^{(k-1)}, x_i^{(k)})\}$
  for $x_i \in ExpLayer$ **do**
    for $j \in [\![0, b-1]\!]$ **do**
      $X \leftarrow X \cup \{(x_i^{(0)} + j, x_i^{(1)} + j, \ldots, x_i^{(k-1)} + j, x_i^{(k)} + jb^{k-1})\}$
return X

We prove this fact by induction in the rest of this section.
The property "IES(b^k,k) builds an LHD with separation distance $d_{\min}$ given by Eq. (4.5)" is expressed by a predicate noted as $H(k)$. To start the induction, we need the following lemma:

Lemma 4.1. *IES(b^2,2) returns an LHD with separation distance $d_{\min} = \sqrt{b^2+1}$.*

Proof. Let $k = 2$. We verify that the result obtained is an LHD of size n in two dimensions and with a separation distance $d_{\min} = \sqrt{b^2+1}$. Let X be the output of IES(b^2,2). We consider two different points of X, x_1 and x_2. We express x_1 and x_2 as a function of variables i_1 and j_1 as stated in line 5 of Algorithm 1:

$$x_1 = (b(i_1+1) - (j_1+1), i_1 + bj_1),$$

$$x_2 = (b(i_2+1) - (j_2+1), i_2 + bj_2),$$

where i_1, i_2 and j_1, j_2 correspond to the loop controlling variables i and j in Algorithm 1 (lines 3 and 4).
As $x_1 \neq x_2$, we have $i_1 \neq i_2$ or $j_1 \neq j_2$. Additionally, we have $i_1, i_2 < b$ and $j_1, j_2 < b$. Noting the coordinate index as a superscript we can write $x_1^{(1)} \neq x_2^{(1)}$ and $x_1^{(2)} \neq x_2^{(2)}$. Moreover, our design X has b^2 points and is thus an LHD. We still have to prove that $\delta(x_1, x_2) \geq \sqrt{b^2+1}$. Expressing the squared Euclidean distance $\mathcal{L}_2^2$ between x_1 and x_2 as a function of their coordinates, we get:

$$\begin{aligned}
\delta(x_1,x_2)^2 &= (b(i_1-i_2)+j_2-j_1)^2 + (i_1-i_2+b(j_1-j_2))^2, \\
\delta(x_1,x_2)^2 &\geq (b(i_1-i_2))^2 + (j_2-j_1)^2 + (i_1-i_2)^2 + (b(j_1-j_2))^2, \\
\delta(x_1,x_2)^2 &\geq (b^2+1)\left((i_1-i_2)^2 + (j_1-j_2)^2\right).
\end{aligned}$$

As $i_1 \neq i_2$ or $j_1 \neq j_2$, we have $(i_1-i_2)^2 + (j_1-j_2)^2 \geq 1$ which leads to:

$$\delta(x_1,x_2)^2 \geq b^2+1,$$

$$\delta(x_1,x_2) \geq \sqrt{b^2+1}.$$

There are two particular, mutually exclusive situations, the first one for $i_1 = i_2$ and $j_1 = j_2 \pm 1$ and the second one for $j_1 = j_2$ and $i_1 = i_2 \pm 1$, where $(i_1-i_2)^2 + (j_1-j_2)^2 = 1$ is satisfied, which leads to:

$$\delta(x_1,x_2) = \sqrt{b^2+1}.$$

□

The inductive step is made for $k > 2$. According to the inductive hypothesis, $H(k-1)$ holds. Let design X be the output of IES(b^k,k). First we check that X is an LHD of size b^k.

Lemma 4.2. *If $H(k-1)$ holds, then design X is an LHD of size b^k.*

Proof. According to the construction of X we add b points for each point of Y = IES(b^{k-1},$k-1$), as stated in lines 14–16 of Algorithm 1. Design Y has b^{k-1} points as $H(k-1)$ is satisfied. Thus design X has $bb^{k-1} = b^k$ points.
We also verify that for each coordinate, each of its possible values is only present in a single point. Let x_1 and x_2 be two different points of design X, as noted above. We observe their k^{th} coordinate $x_1^{(k)}$ and $x_2^{(k)}$, respectively. If we note

$$i_1 = x_1^{(k)} \bmod b^{k-1}, \quad i_2 = x_2^{(k)} \bmod b^{k-1}, \quad \text{and } j_1 = \left\lfloor \frac{x_1^{(k)}}{b^{k-1}} \right\rfloor, \quad j_2 = \left\lfloor \frac{x_2^{(k)}}{b^{k-1}} \right\rfloor,$$

we will observe that i_1 and i_2 are the indices of the points of IES(b^{k-1},$k-1$) used to construct x_1 and x_2 (lines 9, 12 and 14), and that j_1 and j_2 are the values of the loop controlling variable j of the algorithm, and thus the layer numbers of x_1 and x_2, respectively (lines 15 and 16). In the example given in Fig. 4.6, i corresponds to the vertical position of the point in both the 4×4 grids. This is also the position on the boldly typed 2×2 parts of the 8×8 grids. In the same example, j corresponds to the vertical or horizontal position of the points in the 2×2 grid which are inside one square of the boldly typed grid in the last phase. Resuming our example, for point **3** in the upper right-hand corner of Fig. 4.6, we have $i = 3$ and $j = 0$. Let y_1 and y_2 be the points of Y = IES(b^{k-1},$k-1$) used to construct x_1 and x_2. For the first $k-1$ coordinates, we see that the m^{th} coordinate, $y_1^{(m)} = \left\lfloor \frac{x_1^{(m)}}{b} \right\rfloor$ and $j_1 = x_1^{(m)} \bmod b$ (respectively $y_2^{(m)} = \left\lfloor \frac{x_2^{(m)}}{b} \right\rfloor$ and $j_2 = x_2^{(m)} \bmod b$) .
Additionally, we have $j_1 \neq j_2$, or $i_1 \neq i_2$, because we put $x_1 \neq x_2$. Moreover, if $i_1 \neq i_2$, we have, thanks to the induction hypothesis, $y_1^{(m)} \neq y_2^{(m)}$. Thus either $\left\lfloor \frac{x_1^{(m)}}{b} \right\rfloor \neq \left\lfloor \frac{x_2^{(m)}}{b} \right\rfloor$ or $x_1^{(m)} \bmod b \neq x_2^{(m)} \bmod b$, which leads to: $x_1^{(m)} \neq x_2^{(m)}$.
We still have to prove that the k^{th} coordinates of x_1 and x_2 are different. As above, we note that j_1 and i_1 (respectively j_2 and i_2) are the quotient and the remainder of the Euclidean division of $x_1^{(k)}$ (respectively $x_2^{(k)}$) by b^{k-1}.
As $j_1 \neq j_2$ or $i_1 \neq i_2$, we have $x_1^{(k)} \neq x_2^{(k)}$. Thus X is an LHD of size b^k. □

To complete the induction, we need to show that if $H(k-1)$ holds, then the separation distance of the LHD X is:

$$d_{\min} = \sqrt{b^{2(k-1)} + k - 1}. \tag{4.6}$$

Eq. (4.6) is, indeed, Eq. (4.5) with n replaced by b^k.
In order to prove this equation, we show that the distance between two different points x_1 and x_2 is at least $d_{\min}$. We investigate several cases, depending on the relative position of the points in the layers, and treated in three separated lemmas. We can have either two points of the same layer (Lemma 4.3), or two points of adjacent layers (Lemma 4.4), or two points of non-adjacent layers (Lemma 4.5). If the two points are in adjacent layers (Lemma 4.4), we have multiple cases again, where either the two points are constructed starting from the same point of the LHD of dimension $k-1$, or they are not. In the latter case, we have two additional sub-cases depending on the distance between the two points of the LHD of dimension $k-1$ we used to construct our two original points. The above-mentioned cases to be treated, together with the numbers of the auxiliary theorems which covers them, are illustrated in Fig. 4.7. We remind the reader that j_1 and j_2 represent the same j as in Algorithm 1, and thus the layer numbers of x_1 and x_2 (lines 15 and 16).
We start by treating the case where the points x_1 and x_2 are in the same layer:

Lemma 4.3. *If $H(k-1)$ holds and x_1 and x_2 are in the same layer, then $\delta(x_1 - x_2) > d_{\min}$.*

Proof. In this case, x_1 and x_2 are from the same LHD Y = IES(b^{k-1},$k-1$) of dimension $k-1$, which means that $j_1 = j_2$. Let y_1 and y_2 be the points of Y used to construct x_1 and x_2:

$$\delta(x_1, x_2)^2 = b^2 \delta(y_1, y_2)^2 + (x_1^{(k)} - x_2^{(k)})^2.$$

Since $x_1^k \neq x_2^k$, we have $(x_1^k - x_2^k)^2 \geq 1$. We then obtain:

$$\delta(x_1, x_2)^2 \geq b^2 d_{\min,k-1}^2 + 1.$$

As $H(k-1)$ is valid, we have

$$d_{\min,k-1}^2 = b^{2(k-2)} + k - 2.$$

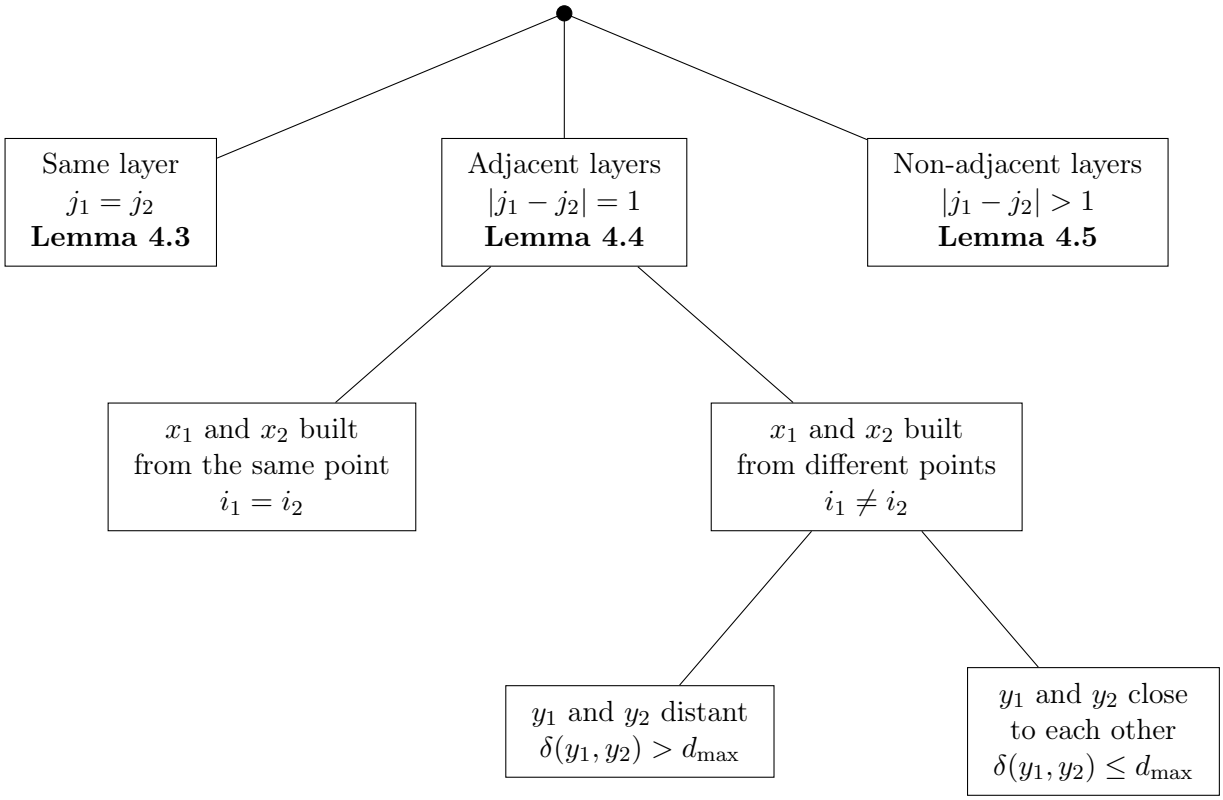

Figure 4.7: Sub-cases treated in Lemmas 4.3, 4.4 and 4.5 Lemma 4.6

Combining the two previous formulæ, we get:

$$\delta(x_1, x_2)^2 \geq b^{2(k-1)} + b^2(k-2) + 1,$$

$$\delta(x_1, x_2)^2 > b^{2(k-1)} + k - 1 = d_{\min}^2.$$

We conclude that $H(k)$ holds in this case. □

We now treat the case where the points x_1 and x_2 are in two adjacent layers:

Lemma 4.4. *If $H(k-1)$ holds and x_1 and x_2 are in adjacent layers, then $\delta(x_1, x_2) \geq d_{\min}$.*

Proof. In this case, x_1 and x_2 are in two adjacent layers, which means that $|j_1 - j_2| = 1$. Let i_1, i_2 be such that

$$i_1 = x_1^{(k)} \bmod b^{k-1} \text{ and } i_2 = x_2^{(k)} \bmod b^{k-1}.$$

In this situation, i_1 and i_2 are the indices of the points y_1 and y_2 of IES(b^{k-1},$k-1$) used to construct x_1 and x_2 (lines 9, 12 and 14 of Algorithm 1). Two cases occur: either $i_1 = i_2$ or $i_1 \neq i_2$.

- Case $i_1 = i_2$:
  Each coordinate of x_1 and x_2 differs by 1, except the last one, which differs by b^{k-1}:

  $$\delta(x_1, x_2)^2 = b^{2(k-1)} + k - 1 = d_{\min}^2.$$

  This is the case where the equality holds.

For the second case, $i_1 \neq i_2$, we distinguish two sub-cases: either y_1 and y_2 are far enough from each other (more than a certain distance $d_{\max}$ still to be defined) to get the result we want directly, or they are closer than this distance and we will have to use the last coordinate of x_1 and x_2 to prove that they are still farther than $d_{\min}$.

- Case $i_1 \neq i_2$:

  For the first $k-1$ coordinates, we have:

  $$b(y_1 - y_2) = (x_1 - x_2)_{k-1} \pm \mathbb{1}_{k-1},$$

  where $\mathbb{1}_{k-1}$ is the vector of size $k-1$ with all elements equal to 1. Using the triangle inequality and noting as $\delta(x_1, x_2)_{k-1}$ the distance between x_1 and x_2 computed with the $k-1$ first coordinates, we get:

  $$b\delta(y_1, y_2) < \delta(x_1, x_2)_{k-1} + \sqrt{k-1},$$

  which leads to:

  $$\delta(x_1, x_2)_{k-1} > b\delta(y_1, y_2) - \sqrt{k-1}.$$

  Let

  $$d_{\max} = \sqrt{b^{2(k-2)} + \frac{k-1}{b^2}} + \sqrt{\frac{k-1}{b^2}}. \tag{4.7}$$

  – If the points are too far apart, $\delta(y_1, y_2) > d_{\max}$:

  $$\delta(x_1, x_2)_{k-1} > bd_{\max} - \sqrt{k-1}.$$

  The function of b and k describing $d_{\max}$ is given by Eq. (4.7). We insert it in the inequality above, obtaining:

  $$\begin{aligned} \delta(x_1, x_2)_{k-1} &> b\left(\sqrt{b^{2(k-2)} + \frac{k-1}{b^2}} + \sqrt{\frac{k-1}{b^2}}\right) - \sqrt{k-1}, \\ \delta(x_1, x_2)_{k-1} &> \sqrt{b^{2(k-1)} + k - 1} + \sqrt{k-1} - \sqrt{k-1}. \end{aligned}$$

  This allows us to bound $\delta(x_1, x_2)$:

  $$\delta(x_1, x_2) > \delta(x_1, x_2)_{k-1} > \sqrt{b^{2(k-1)} + k - 1} = d_{\min}.$$

  – If the points are too close to each other, $\delta(y_1, y_2) \leq d_{\max}$, we have $|y_1^{(k-1)} - y_2^{(k-1)}| < d_{\max}$. Consequently:

  $$|x_1^{(k)} - x_2^{(k)}| = b^{k-1} - |y_1^{(k-1)} - y_2^{(k-1)}| > b^{k-1} - d_{\max}.$$

  For the first $k-1$ coordinates, $\delta(x_1, x_2)_{k-1} > b\delta(y_1, y_2) - \sqrt{k-1}$. Using $H(k-1)$, we have:

  $$\delta(x_1, x_2)_{k-1} > bd_{\min,k-1} - \sqrt{k-1}.$$

  We rewrite the Euclidean distance:

  $$\delta(x_1, x_2)^2 = \delta(x_1, x_2)_{k-1}^2 + |x_1^{(k)} - x_2^{(k)}|^2.$$

  Putting together the two formulæ above, we write:

  $$\delta(x_1, x_2)^2 > \left(bd_{\min,k-1} - \sqrt{k-1}\right)^2 + \left(b^{k-1} - d_{\max}\right)^2.$$

We show that the right-hand side of the above equation is greater than $d_{\min}^2$ if $b \geq 3$ or ($b = 2$ and $k \geq 6$).
Let $A = \left(bd_{\min,k-1} - \sqrt{k-1}\right)^2 + \left(b^{k-1} - d_{\max}\right)^2 - d_{\min}^2$.
We will show that $A > 0$:

$$\begin{aligned} A = \quad & \left(b\sqrt{b^{2(k-2)} + k - 2} - \sqrt{k-1}\right)^2 + \\ & \left(b^{k-1} - \left(\sqrt{b^{2(k-2)} + \tfrac{k-1}{b^2}} + \sqrt{\tfrac{k-1}{b^2}}\right)\right)^2 - \quad b^{2(k-1)} - k + 1, \end{aligned}$$

which gives after developing:

$$\begin{aligned} A &= b^2(b^{2(k-2)} + k - 2) + k - 1 - 2\sqrt{(k-1)(b^{2(k-2)} + k - 2)} + b^{2(k-1)} \\ &+ b^{2(k-2)} + \frac{k-1}{b^2} + \frac{k-1}{b^2} + 2\sqrt{\frac{k-1}{b^2}(b^{2(k-2)} + \frac{k-1}{b^2})} \\ &- 2b^{k-1}\sqrt{b^{2(k-2)} + \frac{k-1}{b^2}} - 2b^{k-1}\sqrt{\frac{k-1}{b^2}} - b^{2(k-1)} - k + 1. \end{aligned}$$

After the simplification, the suppression of positive terms and the use of the inequality $\sqrt{x+y} < \sqrt{x} + \sqrt{y}$, we get:

$$\begin{aligned} A &> b^{2(k-1)} - 2\sqrt{k-1}b^{k-2} - 2(k-1) + b^{2k-4} - 2b^{2k-3} - 2\sqrt{k-1}b^{k-2} \\ &- 2\sqrt{k-1}b^{k-2}, \\ A &> b^{2k-4}(b^2 - 2b + 1) - 6\sqrt{k-1}b^{k-2} - 2(k-1), \end{aligned}$$

which gives:

$$A > b^{2k-4}\left((b-1)^2 - \frac{6\sqrt{k-1}}{b^{k-2}} - \frac{2k-2}{b^{2k-4}}\right).$$

Let $f(b,k) = (b-1)^2 - \frac{6\sqrt{k-1}}{b^{k-2}} - \frac{2k-2}{b^{2k-4}}$. Substituting $b = 3, k = 3$ and $b = 2, k = 6$ in this formula we obtain that $f(3,3) > 0$ and $f(2,6) > 0$. We will show that f is increasing with respect to k and b for $b \geq 2$ and $k \geq 3$. We will use this property of f to conclude that $f > 0$ for $(b,k) \in [\![6, +\infty[\![\times [\![2, +\infty[\![\cup [\![3, +\infty[\![\times [\![3, +\infty[\![$, and that we then have $A > 0$.
We observe that f is increasing with respect to b as the positive terms are increasing and the negative terms are decreasing (when $b \geq 1$). Let $f_1(b,k) = -\frac{6\sqrt{k-1}}{b^{k-2}}$ and $f_2(b,k) = -\frac{2k-2}{b^{2k-4}}$. We note that

$$\frac{\partial f}{\partial k} = \frac{\partial f_1}{\partial k} + \frac{\partial f_2}{\partial k},$$

* $\frac{\partial f_1}{\partial k}(b,k) = \frac{\ln(b)\sqrt{k-1} - \frac{1}{2\sqrt{k-1}}}{b^{k-2}} > 0$ for $b \geq 2$ and $k \geq 3$,
* $\frac{\partial f_2}{\partial k}(b,k) = \frac{2(2k-2)\ln(b) - 2}{b^{2k-4}} > 0$ for $b \geq 2$ and $k \geq 3$.

Thus f is increasing with respect to k and b. As $A > f(b,k)$ and $f(b,k) > 0$ when $b \geq 3$ as well as when $b = 2$ and $k \geq 6$, we have $A > 0$ in these cases.
Thus

$$\delta(x_1, x_2)^2 > d_{\min}^2.$$

We checked that the algorithm returns an LHD of dimension k and size b^k with a separation distance $d_{\min}$ for the cases $(b,k) = (2,3), (2,4)$ and $(2,5)$.

□

We finish by examining the case where the points x_1 and x_2 are in two different and non-adjacent layers:

Lemma 4.5. *If $H(k-1)$ holds and x_1 and x_2 are on two different, non-adjacent layers, then $|x_1 - x_2| > d_{\min}$.*

Proof. The proof in this case is trivial as x_1 and x_2 are separated by at least one layer of size b^{k-1} and thus are separated by more than $d_{\min}$.

□

We summarize the three previous lemmas:

Lemma 4.6. *If $H(k-1)$ holds, then the separation distance of the LHD X is:*

$$d_{\min} = \sqrt{b^{2(k-1)} + k - 1}.$$

Proof. This proof is immediate as the veracity of Lemmas 4.3, 4.4, and 4.5 ensures that the distance between any pair of points is at least equal to the one given by Eq. (4.6).

□

These lemmas allow us to state the following theorem:

Theorem 4.5. *IES(b^k,k) returns an LHD with separation distance*

$$d_{\min} = \sqrt{n^{\frac{2(k-1)}{k}} + k - 1}.$$

Proof. Proof of Theorem 4.5 Assuming $b \geq 2$, we proved that $H(2)$ holds (Lemma 4.1), and that if $H(k-1)$ holds, then $H(k)$ holds too (Lemmas 4.2 and 4.6). Thus, by induction, $H(k)$ holds for every $k \geq 2$ which concludes the proof. □

4.3.4 Approximation ratio

We use the upper bounds presented above to calculate approximation ratios for the IES algorithm. As this algorithm is only defined for values of n and k such that $n = b^k$, with $b \in \mathbb{N}$, the approximation ratio is only computed for such values of n and k.

Using the bound for large-size LHD.

We suppose $n = b^k$, with $b \geq 1$ and $k > 2$. As we address the quality of IES for large instances we also put $n > \frac{2^k}{C(k)}$. Our algorithm returns an LHD of size n with a separation distance $d_{\min}$ given by Eq. (4.5). Let $\rho = \frac{d_{\min}}{d^*}$. With the bound of d^* from Eq. (4.1) we obtain:

$$\rho \geq \frac{\sqrt{n^{\frac{2(k-1)}{k}} + k - 1}}{\frac{2n^{\frac{k-1}{k}}}{\sqrt[k]{C(k)} - \frac{2}{\sqrt[k]{n}}}} = \frac{1}{2}\left(\sqrt[k]{C(k)} - \frac{2}{\sqrt[k]{n}}\right)\sqrt{1 + \frac{k-1}{n^{\frac{2(k-1)}{k}}}} \underset{n\to\infty}{\longrightarrow} \frac{\sqrt[k]{C(k)}}{2}. \tag{4.8}$$

We want to find an equivalent of $\sqrt[k]{C(k)} = \sqrt[k]{\frac{\pi^{\frac{1}{2}}}{\Gamma(\frac{k}{2}+1)}}$. We know that

$$\Gamma(x) = x^{x-\frac{1}{2}}\,\mathrm{e}^{-x}\,\sqrt{2\pi}\left(1 + \mathcal{O}\left(\frac{1}{x}\right)\right),$$

thus

$$\sqrt[k]{\Gamma\left(\frac{k}{2}+1\right)} = \left(\frac{k}{2}+1\right)^{\frac{1}{2}+\frac{1}{2k}} \mathrm{e}^{-(\frac{1}{2}+\frac{1}{k})} \sqrt[2k]{2\pi} \sqrt[k]{1+\mathcal{O}\left(\frac{1}{\frac{k}{2}+1}\right)},$$

which gives

$$\sqrt[k]{\Gamma\left(\frac{k}{2}+1\right)} = \left(\frac{k}{2}+1\right)^{\frac{1}{2}} \mathrm{e}^{-\frac{1}{2}} \left(\frac{k}{2}+1\right)^{\frac{1}{2k}} \mathrm{e}^{-\frac{1}{k}} \sqrt[2k]{2\pi} \sqrt[k]{1+\mathcal{O}\left(\frac{1}{\frac{k}{2}+1}\right)}.$$

We note that

$$\left(\frac{k}{2}+1\right)^{\frac{1}{2k}} \mathrm{e}^{-\frac{1}{k}} \sqrt[2k]{2\pi} \sqrt[k]{1+\mathcal{O}\left(\frac{1}{\frac{k}{2}+1}\right)} \underset{k\to\infty}{\longrightarrow} 1.$$

Thus:

$$\sqrt[k]{\Gamma\left(\frac{k}{2}+1\right)} \sim \sqrt{\frac{k}{2\,\mathrm{e}}},$$

which finally gives

$$\sqrt[k]{C(k)} \sim \sqrt{\frac{2\pi\,\mathrm{e}}{k}}. \tag{4.9}$$

This allows us to obtain using Eq. (4.8):

$$\rho \geq f \sim \sqrt{\frac{\pi\,\mathrm{e}}{2k}}.$$

, where f is the expression described in Eq. (4.8). The approximation (4.9) will also be useful when computing the approximation ratio for LHDs of any size.

Using the bound for LHD of any size.

We suppose $n = b^k$, with $b \geq 1$ and $k > 2$. The separation distance $d_{\min}$ from Eq. (4.5) together with the bound of d^* from Eq. (4.2) yield:

$$\rho \geq \frac{\sqrt{n^{\frac{2(k-1)}{k}}+k-1}}{\frac{4n^{\frac{k-1}{k}}}{\sqrt[k]{C(k)}}} = \frac{1}{4}\sqrt[k]{C(k)}\sqrt{1+\frac{k-1}{n^{\frac{2(k-1)}{k}}}} > \frac{\sqrt[k]{C(k)}}{4} \sim \sqrt{\frac{\pi\,\mathrm{e}}{8k}}. \tag{4.10}$$

Using van Dam's bound.

The separation distance $d_{\min}$ given by Eq. (4.5) together with van Dam's bound, Eq. (4.3), give us:

$$\rho \geq \sqrt{\frac{n^{\frac{2(k-1)}{k}}+k-1}{\left\lfloor\frac{n(n+1)k}{6}\right\rfloor}} \geq \sqrt{\frac{6}{k}\frac{n}{n+1}\frac{1}{\sqrt[k]{n^2}}+\frac{6(k-1)}{n(n+1)k}}. \tag{4.11}$$

4.3.5 Comparison of the approximation ratios

The applicability of the three approximation ratios is conditional on the size n and dimension k of the maximin LHD considered. As shown in Fig. 4.8, the first approximation ratio, Eq. (4.8), is the best for high values of n, the last one, Eq. (4.11), is the best for low values of n. The second one, Eq. (4.10), can be better than the others for medium values of n, depending on the dimension k.

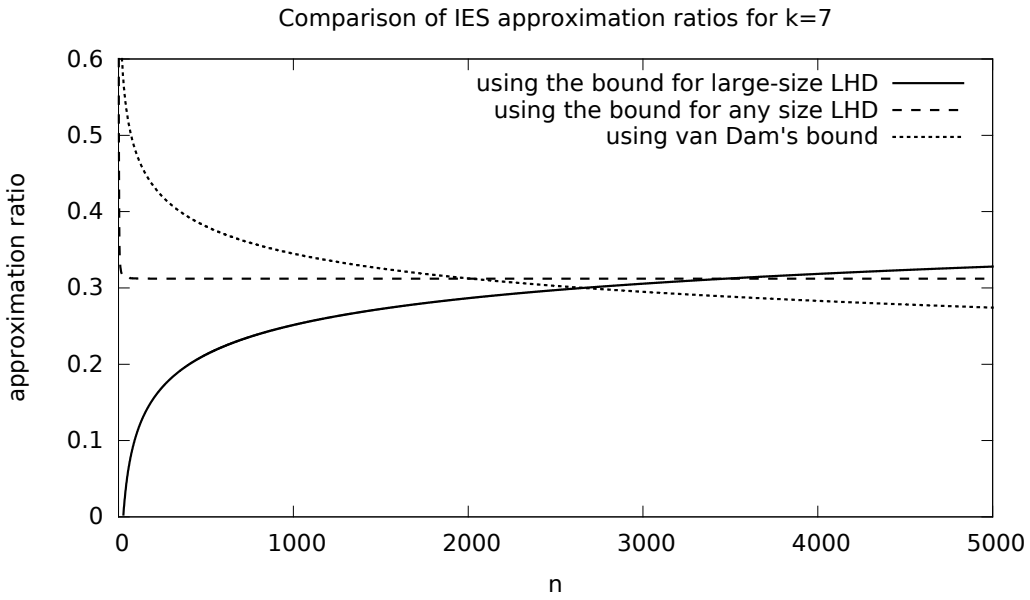

Figure 4.8: Comparison of the approximation ratios

4.4 Generalized algorithms

IES is designed for $n = b^k$, where b is a natural number, which limits its practical application. To overcome this restriction we have designed algorithms that find a satisfactory LHD for any n and k. We take as a starting point the IES algorithm.

4.4.1 Fixed-layer extension

One extension to the IES is to build an LHD using the IES algorithm for $b = \lfloor \sqrt[k]{n} \rfloor$, and continue to add layers until reaching at least n points. The LHD obtained may have more than n points. Some points are to be removed afterwards, to obtain the desired number of points.

We first count the number of layers we need, by dividing n by the size of the layers. We then have $\lceil \frac{n}{b^{k-1}} \rceil$ layers. We modify the expand phase so that we multiply the coordinates of each point by the number of layers.

Once we have finished the three IES phases, we may end up with too many points. We therefore remove points from the last layer to have n points. We start by deleting the last constructed point and decrement the coordinates of each point greater than the corresponding coordinates of the previously deleted point. We repeat this two-phase (removal-adjustment) operation until the number of points n is reached. As a result we get an LHD of n points with the same separation distance as the one obtained with the IES algorithm. The algorithm design implies that its approximation ratio may be given. We name this extension IES-FL (Fixed Layer).

The first part of this algorithm, before we start removing points, has the same complexity as the IES algorithm (see Eq. (4.4)). The number of points to be removed is at most equal to the layer size, *i.e.* $n^{\frac{k-1}{k}}$. To remove a point, we have to check each coordinate of each other point. We thus have to execute $\mathcal{O}(nk)$ operations for each point to be deleted. The part of the algorithm that "wipes out" surplus points costs $\mathcal{O}(kn^{1+\frac{k-1}{k}})$ operations. The total complexity of IES-FL is therefore

$$C_{\mathrm{FL}}(n,k) = \mathcal{O}(k^2 n^{1+\frac{k-1}{k}}).$$

As the IES-FL algorithm preserves the separation distance of the IES algorithm, we can calculate the ratio between the two separation distances in the worst case which happens when the size n is "almost" equal b^k, $n = b^k - 1$.
On the one hand, an LHD of size $n = b^k - 1$ produced by the IES-FL algorithm has a separation distance which is, according to Eq. (4.5):

$$d_{\text{IES}} = \sqrt{(b-1)^{k-1} + k - 1}.$$

On the other hand, the IES algorithm builds an LHD of size $n+1$ whose separation distance d_{IES} is explicitly given by Eq. (4.5). Therefore, if ρ_{IES} is an approximation ratio for the IES algorithm,

$$\rho_{\text{IES-FL}} = \frac{d_{\text{IES-FL}}}{d_{\text{IES}}} \rho_{\text{IES}}$$

is an approximation ratio for the IES-FL algorithm. The fact that the IES-FL ratio is lower than the IES ratio is not astonishing:

$$\frac{d_{\text{IES-FL}}}{d_{\text{IES}}} = \frac{\sqrt{(b-1)^{2(k-1)} + k - 1}}{\sqrt{b^{2(k-1)} + k - 1}} \geq \frac{(b-1)^{k-1}}{b^{k-1}}. \tag{4.12}$$

With the approximation ratios from Subsection 4.3.4 (Eqs. (4.8), (4.10), and (4.11)), we give three approximation ratios for this algorithm.

- For large-size LHD ($n > \frac{2^k}{C(k)}$):
  With ρ_{IES} given by Eq. (4.1) and with Eq. (4.12) we obtain:

$$\rho_{\text{IES-FL}} \geq \frac{1}{2} \frac{(b-1)^{k-1}}{b^{k-1}} \left(\sqrt[k]{C(k)} - \frac{2}{\sqrt[k]{n}} \right) \sqrt{1 + \frac{k-1}{n^{\frac{2(k-1)}{k}}}}.$$

- Similarly, with ρ_{IES} for an LHD of any size from Eq. (4.10) we get:

$$\rho_{\text{IES-FL}} \geq \frac{(b-1)^{k-1}}{4b^{k-1}} \sqrt[k]{C(k)} \sqrt{1 + \frac{k-1}{n^{\frac{2(k-1)}{k}}}} > \frac{\sqrt[k]{C(k)}}{4}.$$

- Finally, with ρ_{IES} according to van Dam's bound given by Eq. (4.3), we have:

$$\rho_{\text{IES-FL}} \geq \frac{(b-1)^{k-1}}{b^{k-1}} \sqrt{\frac{6}{k} \frac{n}{n+1} \frac{1}{\sqrt[k]{n^2}} + \frac{6(k-1)}{n(n+1)k}}.$$

4.4.2 Adapted Layers

The IES algorithm may be subject to another modification enabling it to treat instances with any values of n and k. In this case, we change the size of the layers, increasing the separation distance. We call this algorithm IES-AL (Adapted Layers) because the idea is to change the size of the layers rather than simply adding more fixed-size layers. To achieve this goal, we take a bigger LHD of dimension $k-1$ as our core layer. Once a certain threshold is reached, we add a new layer instead of increasing the current layer size.
Let $b = \lfloor \sqrt[k]{n} \rfloor$. Either $n \leq b(b+1)^{k-1}$, in which case we use b layers, or $n > b(b+1)^{k-1}$, in which case we use $b+1$ layers.
In the first case, $n \leq b(b+1)^{k-1}$, we have b layers, some of them sized $\lceil \frac{n}{b} \rceil$ and some of them sized $\lceil \frac{n}{b} \rceil - 1$. We construct the layers using an LHD of dimension $k-1$ and size $\lceil \frac{n}{b} \rceil$ with the same layer construction procedure as the IES algorithm. For the layers smaller in size, we remove the last constructed point of the layer using the same procedure as for IES-FL.

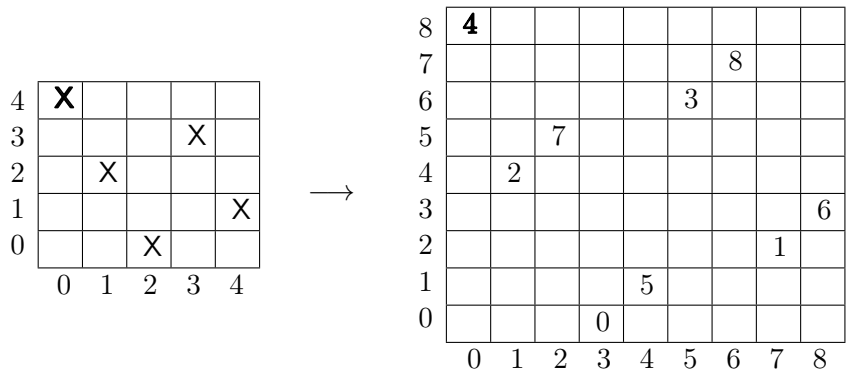

Figure 4.9: IES-AL in three dimensions for $n = 9$. The bolded point becomes only one point in two dimensions, the last layer is incomplete.

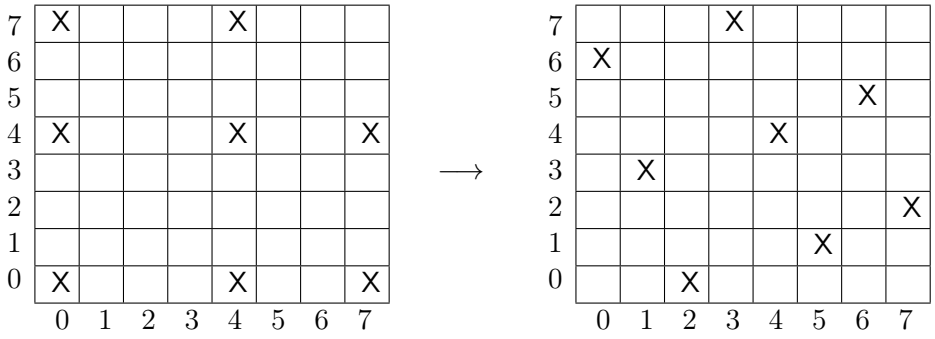

Figure 4.10: IES-AL in two dimensions for $n = 8$. We remove the last constructed point from the result of IES for $n = 9$.

Figure 4.9 shows the construction explained above, for the particular example where $k = 3$ and $n = 9$. Here, the layer's size is 5. When we construct the final result, we need to remove one point. The boldly printed point is alone in the final result, as the other one had to be deleted. Note that the other points found their place thanks to the "classical" IES manipulation.

In the second case, $n > b(b+1)^{k-1}$, we have b layers of size $(b+1)^{k-1}$, and we build the last layer using the first $n-(b+1)^k$ points of an LHD of dimension $k-1$ and size $(b+1)^k$. Once again, we remove the points of the last layer using the same procedure as for IES-FL. For the two-dimensional case, we start with a rotated grid of size b^2 using the same procedure as for IES and we add points to one line and then to one column until we have n points (see Fig. 4.10).

In order to retain the same approximation ratio as IES-FL, we build two LHDs, using both extensions, and we take the best one, according to the separation distance produced. However, in practice, we only need IES-AL, as it is almost always better (see Section 4.4.3). Moreover, the worst case of the approximation ratio occurs for values of n such as $n = b^k - 1$. In this case, IES-FL performs badly, as the separation distance will only increase for $n + 1$, while the separation distance for the second extension, IES-AL, is much better. It should be obvious that this algorithm possesses the same time complexity as the former.

4.4.3 Computational results

We present the results we obtained with the different IES algorithms, as summarized in Table 4.2, and compare them with those obtained with the Simulated Annealing algorithm [44]. We discuss the choice of this algorithm in the following subsection.

Algorithm	Values of n and k	Time complexity
IES	$n = b^k$, $b \in \mathbb{N}$	$\mathcal{O}(nk^2)$
IES-FL	Any n and k	$\mathcal{O}(k^2 n^{1+\frac{k-1}{k}})$
IES-AL		

Table 4.2: Summary of IES algorithms

Algorithm used to solve the LHD-CP

Multiple algorithms have been used to solve the maximin LHD problem. Most of these algorithms are metaheuristic algorithms. Simulated Annealing [37], Iterated Local Search [22] and Genetic Algorithms [6] are the most commonly used. It has been shown in [44] that with an appropriate set of parameters, the Simulated Annealing scheme performs better than the other algorithms. We used the SA version proposed in that paper as it allows us to exceed the highscores listed in the literature and the dedicated website (`spacefilling.nl`). Its numerical complexity on each iteration step is of $\mathcal{O}(n^2k^2)$ which makes it well adapted to treat LHDs of large size. For this reason, we compare our results with the results of this algorithm.

Experimental settings

The heuristic algorithm with which we decided to compare our results is Simulated Annealing (SA) adapted to the maximin LHD problem, with the same parameters as in [44]. All experiments were conducted on an i7 4765T 2.00GHz CPU. Each experiment involving SA was carried out for 10 minutes on one core, and was repeated 50 times, which was enough to produce very narrow confidence intervals on a 0.05 confidence level. The performance measure chosen is the mean squared separation distance in the LHD obtained.

Results of IES extensions

The time consumption of both IES-based algorithms is very low, taking at most half a second for the highest values of n and k we used. Thanks to the short execution times we were able to run the algorithm for all values of n within a certain range for a given k. Figures 4.11 and 4.12 show the separation distance of the LHDs obtained by the IES-FL and IES-AL algorithms for $k = 3$ and $k = 5$ for every n from 2 to 9000.
As expected, the IES-FL separation distance only grows when we reach a value of n where $n = b^k$. As for the IES-AL algorithm, we observe that the separation distance also increases step by step, just like the IES-FL algorithm, but these steps are much shorter. This is due to the fact that we use the same core layer for several LHDs consecutive in size. Additionally, the core layer in use may also have the same separation distance. We also see in Fig. 4.12 that, in certain cases, the distance decreases and oscillates slightly. The reason for this phenomenon is that when we use a bigger LHD as a core layer, one of its points will only be used in one layer, to the effect that two points may be brought close together. As we periodically take larger core layers, this behavior may occur several times in a row, creating oscillations.

4.4.4 Comparison with Simulated Annealing

Figures 4.13 and 4.14 show the result of the IES-AL algorithm and the SA algorithm for $k = 3$ and $k = 5$. We observe that SA gives better results for low values of n, and worse results for high values of n. However, it remains unclear which algorithm is better for intermediate values of n, due to the steps and oscillations of the IES-AL results as discussed in Subsection 4.4.3. To offer a better comparison between IES-AL and SA, we give two envelopes, noted

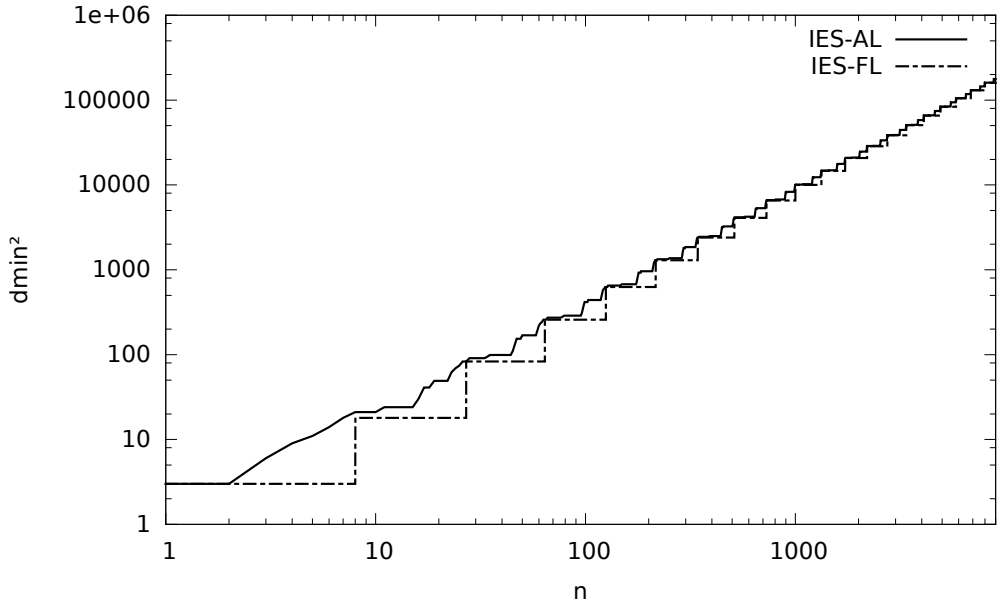

Figure 4.11: The squared separation distance of the LHDs obtained as results of the IES-FL and IES-AL algorithms for $k = 3$, in function of their size.

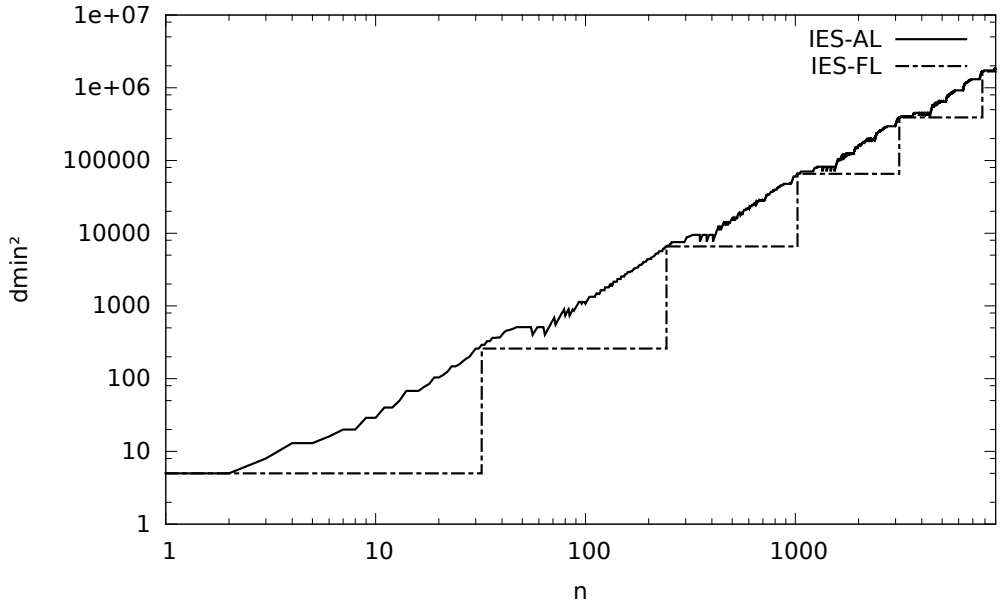

Figure 4.12: The squared separation distance of the LHDs obtained as results of the IES-FL and IES-AL algorithms for $k = 5$, in function of their size.

$d^2_{\min}$ and $cd^2_{\min}$. The first represents an upper bound for the result of IES-AL, obtained using $d^2_{\min,\text{IES}} = n^{\frac{2(k-1)}{k}} + k - 1$. The second represents a lower bound, obtained by comparing the results of the SA algorithm with $d^2_{\min} = c_k d_{\min,\text{IES}}$, where c_k is the worse ratio between the actual results of the IES-AL algorithm and $d^2_{\min,\text{IES}}$, for a fixed k.

Figure 4.15 shows the range where our algorithm beats the Simulated Annealing algorithm, with n on the y axis and k on the x axis. For all points (n, k) in the light gray area, the IES-AL algorithm outperforms SA. For all points in the dark gray area, SA produces better results.

The lower and upper area boundaries were obtained by comparing the results of the SA algorithm with respectively the upper bound $d^2_{\min}$ and the lower bound $cd^2_{\min}$ we have just described. For the central region, IES-AL may or may not be better than the SA algorithm, depending on the value of n. However, since the IES-AL algorithm is fast, it should always be possible to run it alongside the SA algorithm.

We made an attempt to improve the annealing descent by taking an IES-AL solution as an SA starting point. These experiments proved to be unsuccessful. The results were either worse than results obtained with a random starting point (when SA is better) or the same as the IES-AL algorithm (when IES-AL is more efficient). This behavior may be explained by the very regular structure of the LHDs produced by IES. A limited number of iterations will produce a poor solution. A lot of mutations have to be accepted before breaking the regularity, which is necessary to obtain better solutions. This phenomenon compromises the interest of using IES-AL to set a starting point.

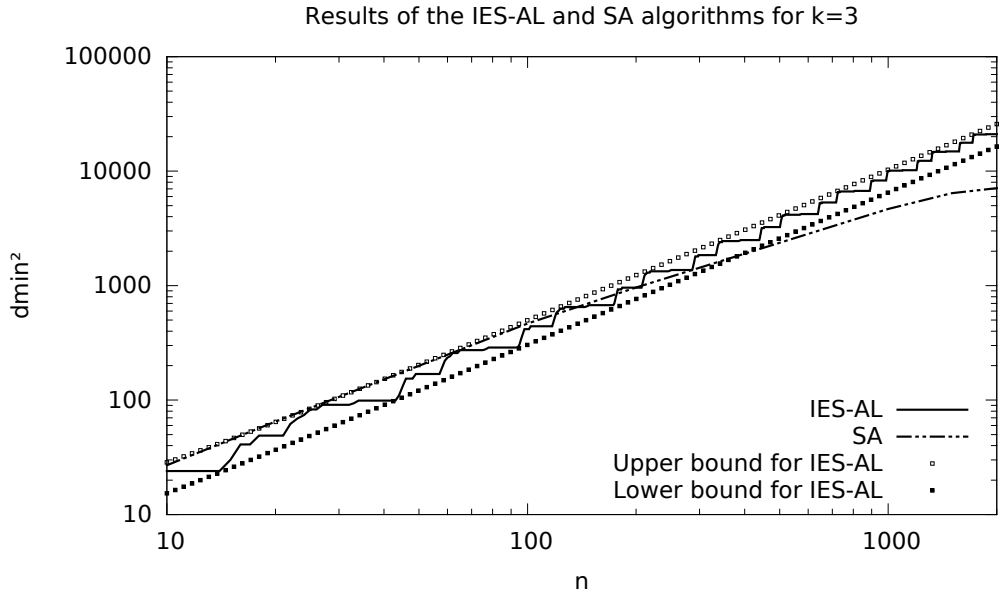

Figure 4.13: Comparison between SA and IES-AL for $k = 3$. The squared separation distance is given in function of the size of the LHD.

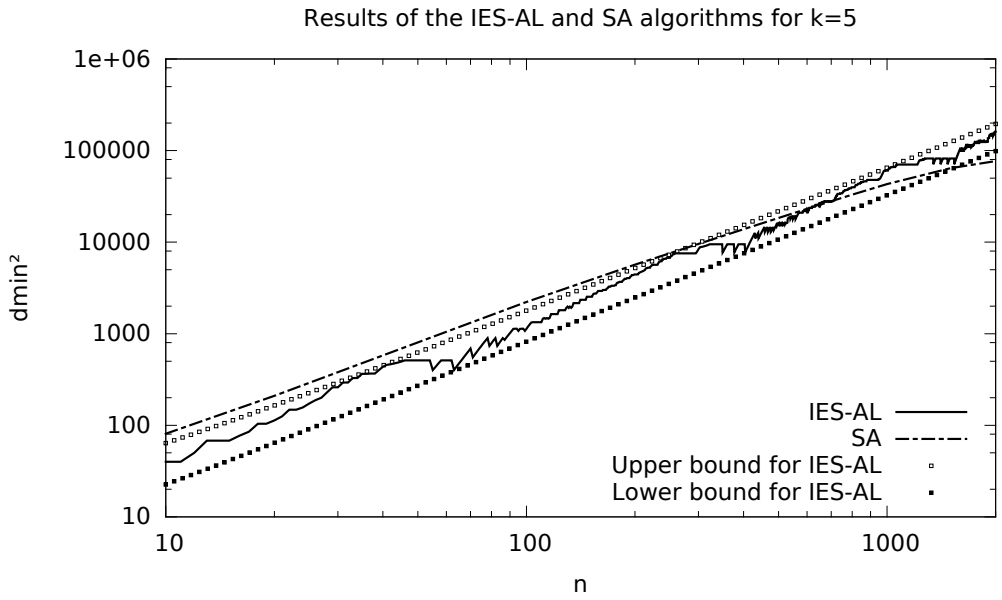

Figure 4.14: Comparison between SA and IES-AL for $k = 5$. The squared separation distance is given in function of the size of the LHD.

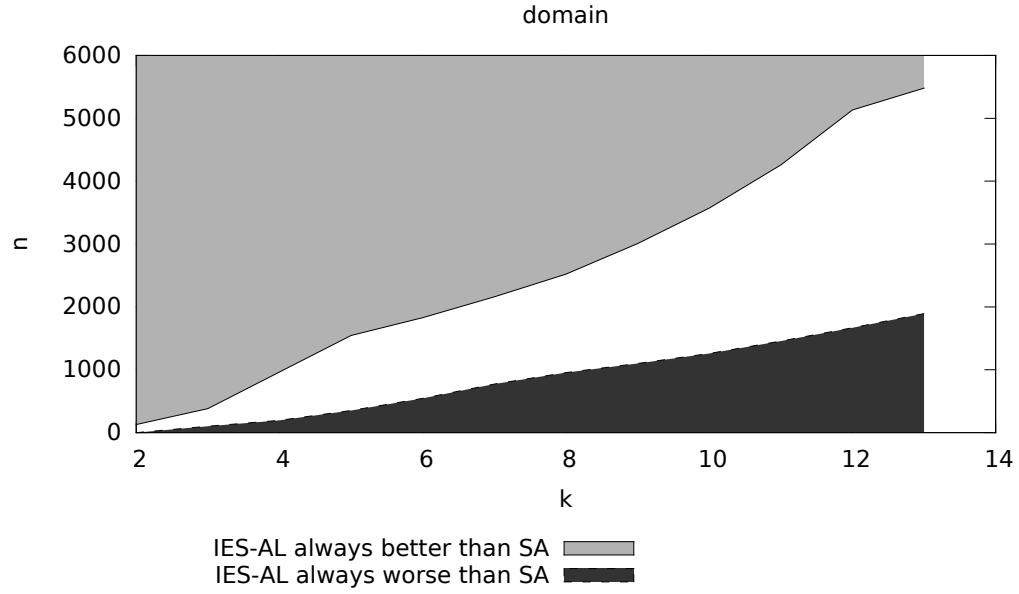

Figure 4.15: Comparison between SA and IES-AL

4.5 Conclusion

We searched for guarantees of performance for both the pLHD-CP and the LHD-CP. We gave inapproximation results for the former, finding that no approximation algorithm exists for $k \geq 3$ if P $\neq$ NP, a result which holds for every norm. We also give an upper bound for any approximation ratio in the two-dimensional space for norm $\mathcal{L}_\infty$. On the contrary, we designed an approximation algorithm for the LHD-CP and proved its approximation ratio using two new upper bounds we constructed. We compared this algorithm with the best metaheuristic algorithm found in the literature, Simulated Annealing. In the next chapter,

we try to get the best possible numerical results for the construction and completion problems to the detriment of theoretical performance bounds.

Chapter 5

Heuristic algorithms

After finding inapproximation results for the pLHD-CP and an approximation algorithm for the LHD-CP, we are interested in methods to produce LHDs or to complete pLHDs, so that they can actually be used for sampling. As the LHD-CP is a subproblem of the pLHD-CP, algorithms for the latter can also be used for the former. However, algorithms specific to the LHD-CP should be more effective. As we defined the pLHD-CP, no algorithms for it exists in the literature.
We start by giving an overview of the numerous algorithms used to construct LHDs. Among these algorithms, we concentrate on the Simulated Annealing metaheuristic, and we follow with improvements we made to Simulated Annealing algorithm (SA) for the the LHD-CP with a new mutation and a new evaluation function, allowing the algorithm to outperform its previous adaptations found in the literature [8]. The last section treats an adaptation of SA to the pLHD-CP [24].
As in practice the most used norm is the Euclidean ($\mathcal{L}_2$) norm, we only consider this norm in this chapter.

5.1 State of the art

In this section, we describe the algorithms used to produce LHDs reported in the literature. We start with exposing two components used in multiple heuristic algorithms, the evaluation function and the mutations, before formulating the algorithms themselves.

5.1.1 Algorithm components

A metaheuristic algorithm usually necessitates two components to be implemented for a particular problem: the evaluation function and the mutation. We present the evaluation functions and the mutation used in the literature to treat the LHD-CP.

Evaluation functions

Most algorithms need to evaluate multiple LHDs to progress. The most straightforward function one can think of is simply the separation distance. However, this function has a drawback. Indeed, two LHDs with the same separation distance have the same evaluation. As the separation distance often keeps the same value if the LHD is slightly modified, an algorithm using this evaluation cannot make a difference between two solutions. An example is illustrated in Fig. 5.1: the two LHD have a separation distance of $\sqrt{2}$, but the distribution of distances between points are the following: $\{\sqrt{2}, \sqrt{2}, \sqrt{2}, 2\sqrt{2}, 2\sqrt{2}, 3\sqrt{2}\}$ for the left LHD and $\{\sqrt{2}, \sqrt{5}, \sqrt{5}, \sqrt{5}, \sqrt{10}, \sqrt{15}\}$ for the right one.
To avoid this inconvenience, two other evaluation functions have been used. The authors of [1] made an analogy by representing points of a design with electrical charges, each exerting

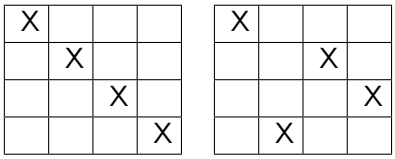

Figure 5.1: Two LHDs with an identical separation distance

a repulsive force on the others. The Audze-Eglais (AE) evaluation function is the potential energy of such a system:

$$\text{AE} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{\delta(p_i, p_j)^2}, \tag{5.1}$$

where $d(p_i, p_j)$ is the distance between p_i and p_j. While this evaluation function tends to increase the separation distance, a maximin LHD does not necessarily maximize the AE function. Conversely, an LHD maximizing AE is not always a maximin LHD. A more general evaluation function has been proposed in [37]:

$$\phi_p = \left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{\delta(p_i, p_j)^p} \right)^{\frac{1}{p}}, \tag{5.2}$$

where $p \geq 1$ is a chosen parameter and $d(p_i, p_j)$ is the distance between p_i and p_j. Contrary to the separation distance, ϕ_p takes every distance into account, and thus varies when the LHD is modified. The p factor makes the lowest distances have more weight in the evaluation function. This makes minimizing ϕ_p equivalent to maximizing the separation distance if p is high enough, while being better at guiding an algorithm.
Note that both evaluation functions can be used for various maximin designs and are not limited to maximin LHDs.

Mutations

Another important component of multiple algorithms is the mutation process. Most algorithm perform a local search and thus need to modify slightly an LHD to obtain another, similar one. Four mutations have been used for a great number of algorithms. They have been proposed by [37] and [27]. To explain these mutations, we use the notion of *critical point*:

Definition 5.1 Critical point**.** *A critical point is a point at distance equal to the separation distance to another point.*

The mutations mentioned above are as follows:

- m_1: Chose randomly a critical point and another point. Exchange a random number of coordinates. The evaluation function is computed once on the resulting LHD.
- m_2: Chose randomly a critical point and another point. Exchange a random coordinate. The evaluation function is computed once on the resulting LHD.
- m_3: Chose randomly a critical point and another point. Exchange the coordinate for which the resulting solution gives the best evaluation. The evaluation function is computed on each of the k possible resulting LHD.
- m_4: Chose randomly two points. Exchange a random number of coordinates. The evaluation function is computed once on the resulting LHD.

Note that m_3 requires multiple computations of the evaluation function, as every k coordinate change has to be evaluated. Therefore, to compare its performance with the performance of other mutations, the number of iterations of an algorithm should be divided by k.
After describing the core elements, we describe the algorithms currently used for constructing LHDs.

5.1.2 Heuristic algorithms

Numerous methods have been used to construct LHDs. Metaheuristic algorithms, such as Genetic Algorithms and Iterated Local Search, have been applied to the problem. Dedicated heuristic algorithms have also been designed. Furthermore, reducing the space of solutions has been tried, by considering particular LHDs such as symmetric LHDs only. We end this section with the description of the Simulated Annealing metaheuristic, which has been found to be the best performing algorithm.

Genetic Algorithms

Genetics algorithms are described in [25] and [21], and have been used to construct Latin Hypercubes in [6] and [35]. Genetic algorithms construct a population of solutions, applies mutations to each solution and crosses them to produce new solutions (*i.e.* construct a child solution from a certain number of parent solutions), and chooses the best-performing ones to obtain a new population.
Three essential components have to be designed: the mutation, the crossover process, and the evaluation function used to select the population. In [6], each solution is represented as k permutations of size n. The mutation used consists in exchanging two values of each permutation. Seeing the solution as a set of points, this mutation is equivalent to exchanging one coordinate of two points for each dimension. The evaluation function used is the Audze-Eglais function (Eq. (5.1)).
Two crossover processes have been found efficient.
The first crossover, called cycle crossover, consists in taking two parent solutions, and, for each permutation, applying the following procedure: Take the first number of the permutation of the first parent, and add that number in the first position of the child. Then, add to the child the number in the same position in the second parent, and add it in the position it belongs in the first parent. Repeat this process until the number in the second parent is already present in the child. Then, fill the remaining positions of the child with the same numbers as the second parent. Two child solutions can be produced this way by exchanging the two parents. This procedure is illustrated on Fig. 5.2.
The second crossover, called inversion, consists in taking one parent, selecting at random two positions in each permutation, and inverting the order of elements between these two positions.
While the two processes are efficient, using cycle crossover followed by inversion crossover has been found to be more efficient than one process alone.

Iterated Local Search

Iterated local search is a metaheuristic first proposed in [7]. It has been applied to the LHD-CP in [22]. The idea behind the algorithm is to perform a local search and apply a mutation until it does not improve the current solution anymore. At this point, a different mutation is applied, introducing a more significant change to the solution, before restarting the local search.
The local search uses a variation of the m_2 mutation: take a critical point, another point, and exchange one of their coordinates. All possible mutations are performed and the one resulting in the best solution is selected if it beats the initial solution.

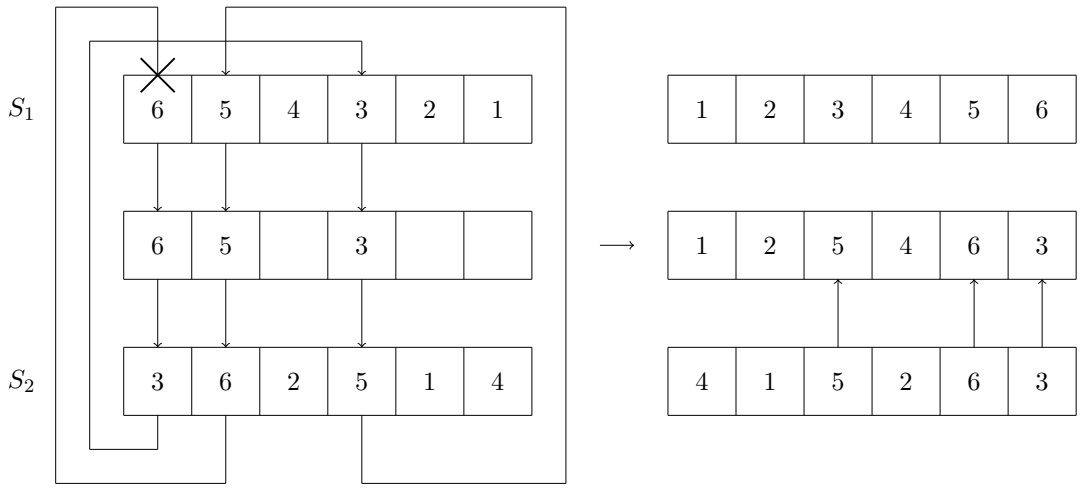

Figure 5.2: The cycle crossover process between two solutions S_1 and S_2. Two steps are applied to build the child solution. In the first step, we copy 6 from S_1 in the first position. At this position, S_2 contains 3, which is in fourth position in S_1, so we copy 3 in the fourth position. Likewise, S_2 contains 5 in fourth position, and 5 is in second position in S_1, so we copy 5 in the second position of the child. As the second position of S_2 contains 6 which has already been placed, we stop here. In the second step, we copy in the remaining empty positions of the child the corresponding numbers from S_2.

Once no improvements are found, the following mutation is applied:
Select randomly two points p_i and p_j, with $i \leq j + 2$. Select a dimension randomly. For all points p_l such that $i < l \leq j$, replace its coordinate on the selected dimension with the previous coordinate of p_{l-1}. Replace the coordinate of p_i on the selected dimension by the previous coordinate of point p_j. In other words, the coordinates of points p_l are exchanged in a circular way.
The algorithm stops when an arbitrary chosen number of local searches do not improve the solution.

Translational Propagation

The idea behind the Translational Propagation algorithm proposed in [53] is to take a small LHD, called a seed, and to build a larger LHD by copying this seed several times.
More precisely, to build an LHD of size n and dimension k with a seed of size s (and the same dimension k), the algorithm divides the target hypercube in b blocks, which are smaller hypercubes. The number of blocks have the following constraint: it has to be a power of k, and has to be greater or equal to $\frac{n}{s}$. The smallest number of blocks respecting these two constraints is selected. Each block is a hypercube of side length $n_s \sqrt[k]{b}$. The seed is then transformed in a pLHD of size $n_s \sqrt[k]{b}$ by multiplying the coordinate of each point by $\sqrt[k]{b}$. Each block is filled by a copy of the modified seed, which is shifted to respect the Latin constraint. As the resulting LHD may have more than n points, surplus points have to be removed until n points are left. To remove a point, we select the farthest point from the center of the block, remove it, and shift the coordinates of the other points to fill the gap: each point with one coordinate greater than the coordinate of the removed point on the same dimension gets this coordinate decreased by one.

Symmetric LHDs

Rather than designing a new algorithm to construct LHDs, we can reduce the space of solutions hoping to obtain better solutions. It has been observed that good quality solutions

have certain properties and focusing on LHD with these properties can lead to satisfactory solutions. Symmetrical LHDs and periodical LHDs have been used to this end.
A symmetrical LHD is an LHD such that for each point it contains, it also contains the image of the point with respect to the central symmetry. In other words, if a symmetrical LHD contains point $p = (p_1, p_2, ...p_k)$, it also contains point $p_s = (n+1-p_1, n+1-p_2, ..., n+1-p_3)$. They have been studied in [60]. To build them, a local search algorithm has been developed. The mutation used is the following: for each dimension, exchange the coordinates of two points on that dimension and exchange the coordinates of their two symmetric points. The algorithm starts from a random symmetric LHD and stops when the mutations do not improve the separation distance.

Periodic LHDs

In [51], 2-dimensional LHDs have been studied. They used an exponential-time algorithm to build optimal LHD for norm $\mathcal{L}_2$. They observed that the majority of optimal LHDs have periodic patterns. They used this fact to design an algorithm for this specific case, which explores every periodic pattern and return the LHD with the highest separation distance.

Simulated Annealing

The Simulated Annealing algorithm, introduced in [32], is a metaheuristic algorithm inspired by the physical process of annealing, in which metal is heated, and progressively cooled. During the metallurgical process, metal molecules organize themselves in a configuration reaching the global energy minimum. The basic step of the algorithm applied to a combinatorial problem is the following:

- Take the previous state, modify it, then evaluate it. This evaluation is also called energy.
- If the modification gives a better evaluation, accept it.
- If the evaluation is worse, accept it with a certain probability depending on the temperature and the difference between the previous and the new evaluation.

The acceptance probability is the following :

$$P(\Delta E) = \exp(-\frac{\Delta E}{kT}),$$

where ΔE is the difference between the two evaluations, k is a constant (usually chosen as $k = 1$) and T the temperature. The temperature varies through the execution of the algorithm, starting high and approaching zero at the end, which models a cooling process. The probability of accepting a worse solution is thus high at the beginning, and very low at the end. T is usually chosen to be either linear or exponentially decreasing.
This allows the algorithm to explore thoroughly the space of solutions without getting stuck in local minima at the start, and reaching a minimum at the end.
The Simulated Annealing algorithm has been applied to the LHD-CP for the first time by [37], using ϕ_p (Eq. (5.2)) as an evaluation function and m_2 as mutation.
Simulated Annealing has been found to be the best performing algorithm by [44], using m_3 as a mutation.

5.1.3 Conclusion

We gave an overview of the techniques used to produce maximin LHDs. We presented the most popular evaluation functions and mutations, and we described the heuristic algorithms: Genetic Algorithms, Iterated Local Search, Translational Propagation, periodic LHDs and

Points	p_1	p_2	p_3	p_4	p_5
dimension 1	0	1	2	3	4
dimension 2	1	2	0	4	3
dimension 3	**2**	1	4	**3**	0

Points	p_1	p_2	p_3	p_4	p_5
x	0	1	2	3	4
y	1	2	0	4	3
z	**3**	1	4	**2**	0

Table 5.1: Illustration of *1D–move* with an initial (left) and a following (right) solution

symmetric LHDs, ending with the most efficient one: Simulated Annealing. We continue our study with modifications we proposed to make the Simulated Annealing more efficient for constructing and completing LHDs [8] [24].

5.2 Simulated Annealing for the LHD-CP

We replaced two key elements in the Simulated Annealing algorithm for the LHD-CP: the mutation and the evaluation function [8]. We took advantage of the fact that a mutation should make the slightest possible modification to a solution. We studied the properties of distances obtained with the evaluation function ϕ_p in SA solutions. This analysis allowed us to establish a better evaluation function. We describe both new elements.

5.2.1 New mutation for the LHD-CP

We designed a new mutation for constructing LHDs. We start by giving the principle of the mutation which tries to preserve the Latin constraint. We evaluate the quality of this mutation to build LHDs taking the traditional evaluation function ϕ_p (Eq. (5.2)).

Principle of the mutation

We use m_2 as a basis to construct a more efficient mutation. Article [50] shows that for the Traveling Salesman Problem, the best mutations for SA are mutations which moves the smallest number of edges. We use this principle to create a mutation that makes the smallest modifications possible to an LHD. To clarify its principle, we define the notion of the neighborhood:

Definition 5.2 Neighbor of a point**.** *For a given instance of the LHD-CP of size n and dimension k, a point p_1 of a solution D is a neighbor of the point p_2 if and only if there is a dimension j for which coordinates of these two points are the closest possible. In other terms, $\exists j \in [|1;k|]$ such that $\left|p_1^{(j)} - p_2^{(j)}\right| = 1$.*

The new mutation *1D–move* consists in taking a critical point and taking one of its neighbors. Then we exchange the coordinates in one of the dimensions concerned by the neighborhood. This allows the mutation to have the least possible effect on the distances inside the initial solution.
We choose an instance with $n = 5$ and $k = 3$ to illustrate *1D–move*. Table 5.1 gives the coordinates of the points of a solution D. First, we choose a critical point: $d_{\min}$ is determined by points p_1 and p_2. We arbitrarily take p_1 for the example. Points p_2 (on dimensions 1, 2 and 3), p_3 (on dimension 2) and p_4 (on dimension 3) are neighbors of p_1. For the sake of the example, we shall choose p_4 as a neighbor. Then, we exchange the coordinates of p_1 and p_4 on dimension 3 because p_1 and p_4 are neighbors through dimension 3. The new solution is also given in Table 5.1.
After describing the mutation, we prove that it has a smaller effect on a solution than the other mutations already used.

Effect of the mutation

To prove that *1D–move* modifies less the LHD than m_3, we define a distance function δ between two LHD D_1 and D_2 of the same size n and dimension k:

Definition 5.3. *Let F be the set of mappings from points of D_1 to points of D_2. Then the distance between D_1 and D_2 is*

$$\delta(D_1, D_2) = \min_{f \in F} \sum_{i=1}^{n} d(p_i, f(p_i)).$$

The distance between two LHD is the sum of the distances between their points taken by pair for the mapping minimizing this sum.
Let us consider an LHD D of size n and dimension k, and the two mutations m_2 and *1D–move*. We want to prove that the changes due to these two mutations can be of different order of magnitude. Let us note:

$$m_2 : D \longrightarrow D' \quad \text{and} \quad \textit{1D–move} : D \longrightarrow D''.$$

We assume the two mutations translate the points p_1 and p_2 on a given dimension j. The objective is to find a configuration for which $\delta(D, D') \gg \delta(D, D'')$. We immediately see that $\delta(D, D'') = 2$: two points are translated by one unit on one dimension. This is obtained by the mapping associating each point of D to its corresponding point in D''. The only points having a non-zero distance are the two points modified by *1D–move*, which are each at a distance one from their original position in D.
Now we note $d_{\min}$ the separation distance of D. We suppose that the j^{th} coordinates of the points displaced by m_2 differ by $\lfloor \frac{d_{\min}}{2} \rfloor$. The distance between D and D' is then $\delta(D, D') = 2\lfloor \frac{d_{\min}}{2} \rfloor$. This is again obtained by the mapping associating each point of D to its corresponding point in D''. Another mapping would result in a greater or equal distance, since the distance between p_1 and any other point p is at least $d_{\min}$, and thus the distance between p'_1 and p it at least $d_{\min} - \lfloor \frac{d_{\min}}{2} \rfloor \geq \lfloor \frac{d_{\min}}{2} \rfloor$. The same reasoning holds for p_2.
By using the result of the IES algorithm for D and choosing $n = b^k$, we obtain that $\delta(D, D') = 2 \left\lfloor \frac{\sqrt{n^{\frac{2(k-1)}{k}}} + k - 1}{2} \right\rfloor$ and finally $\delta(D, D') = \mathcal{O}(n^{\frac{k-1}{k}})$.
In the case discussed, $\delta(D, D') \gg \delta(D, D'')$ which signifies that m_2 can modify the LHD much more than *1D–move*. This makes the local search performed with the latter smoother.

Performance Evaluation

We evaluate the performances of *1D–move* for SA with ϕ_p as an evaluation function and compare them with those of m_2 and m_3. We reproduced the experiments made in [44] keeping the same value of parameter p ($p = 10$) for instances with $k = 4$ and $n = 25$, $k = 9$ and $n = 10$, $k = 8$ and $n = 20$ to show its performance (Table 5.2). SA performs a linear thermal descent until temperature $T = 0$ is reached. The initial temperature is set thanks to a series of preliminary runs. We computed 100 effective runs and we present here the average within the 95% confidence interval. *1D–move* outperforms not only m_2 but m_3 as well.

Conclusion

We presented a new mutation for the LHD-CP, outperforming the previously used mutations. Our interest is now oriented towards the second essential element of Simulated Annealing, the evaluation function.

Instance \ Mutation	m_2	m_3	*1D–move*
$k = 4, n = 25$	177.59 ± 0.29	177.67 ± 0.29	$\mathbf{180.51 \pm 0.27}$
$k = 9, n = 10$	156.24 ± 0.10	156.06 ± 0.08	$\mathbf{156.54 \pm 0.06}$
$k = 8, n = 20$	431.98 ± 0.61	433.72 ± 0.84	$\mathbf{436.20 \pm 0.56}$

Table 5.2: Performance of SA with different mutations

5.2.2 New evaluation function targeting Maximin

Presentation of a Maximin effect: narrowing the distribution of distances

We study the properties of distances obtained with the evaluation function ϕ_p in SA solutions. We represent all the distances between the points of an LHD in histograms and identify properties that will allow us to establish a better evaluation function. We note $M(k, n)$ the random variable representing any square distance in any solution of an instance with dimension k and size n. We note its mean $\overline{M}(k, n)$. As shown in [52], the following equality always holds for norm $\mathcal{L}_2$: $\overline{M}(k, n) = \frac{kn(n+1)}{6}$. From now on, we distinguish three cases relative to values taken by n and k, each case showing a different behavior of the distribution of distances;

- **Case** $n \leq k$ In this case (see the histogram in the top of Figure 5.3, $k = 50, n = 40$), the distances of potential solutions are concentrated around the mean. It is highly probable that two points taken at random will be neighbors. In our example with $k = 50$ and $n = 40$ in Figure 5.3, the statistical range of M relative to $\overline{M}$, $\frac{d_{\max} - d_{\min}}{\overline{M}} = \frac{340}{13667} = 2.5\%$, in fact, is narrow. The rationale for this behavior is that when the number of points is less than the number of dimensions, it happens, in absence of constraints, that all the points are equidistant. Since the Latin constraint has to be respected, the points cannot be exactly equidistant. The distances, however, do not differ significantly. We talk about unimodal distribution.

- **Case** $k \leq n \leq 2k$ In this case (the histogram in the center of Figure 5.3, $k = 30, n = 50$), distributions are concentrated around two peaks. The first peak is mainly around the average distance (actually, there is a little shift between the peak and the mean because both the peaks preserve $\overline{M}$) and the second peak is located around the doubled average distance. Much more distances are concerned by the first peak.

  We illustrate this phenomenon with the instance with $k = 30$ and $n = 50$ in Figure 5.3. We can explain this distribution shape by the fact that it is possible for this many points to be placed in an hyperoctahedron. In such a geometric object, each point is at the same distance from every other point but one, which is farther away. Thus, the distribution of distances shows two values, with the smaller being represented much more frequently. We name it bimodal distribution.

  In our example, $\overline{D}(30, 50) = 12500$. Concerning the highest peak, the statistical range remains small compared with the mean: the ratio is 7.8%, larger than in the first case for the whole distribution. There are only seven distances located in the interval $[13183; 24865]$

- **Case** $2k \leq n$ In this last case (the histogram in the bottom of Figure 5.3, $k = 10, n = 100$), distances are distributed more uniformly. There is neither a dense peak nor a sparse interval. We observe a decrease of occurrences with an increase in the value of the distance. It is an amodal distribution.

- **Observations and consequences** In the unimodal case, the only peak is naturally thin thanks to SA and particularly ϕ_p action. It would be pointless to try to narrow

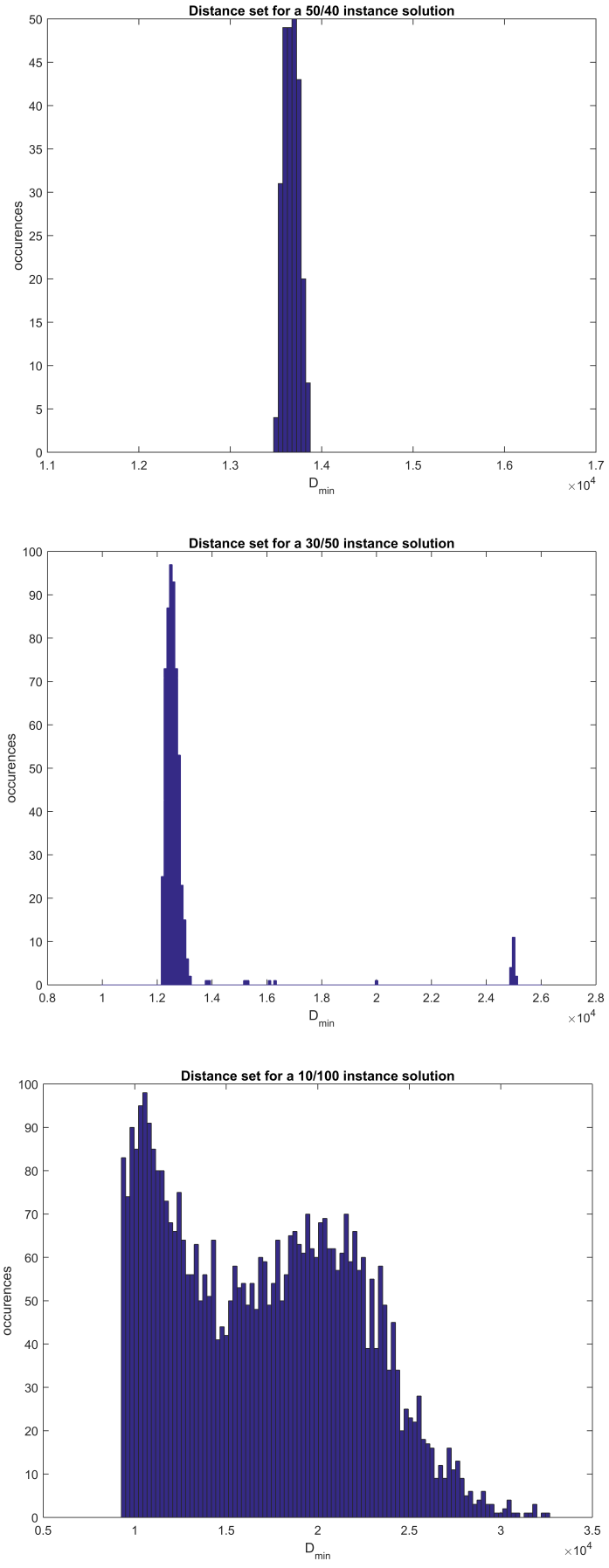

Figure 5.3: Histograms of distances for $(k = 50, n = 40)$, $(k = 30, n = 50)$ and $(k = 10, n = 100)$ solutions

Inst.	σ	ϕ_{10} & m_2	$\psi_{10,\sigma}$ & m_2	ϕ_{10} & *1D-move*	$\psi_{10,\sigma}$ & *1D-move*
$k=4, n=25$	70	177.59 ± 0.29	177.98 ± 0.71	180.51 ± 0.27	181.24 ± 0.23
$k=9, n=10$	20	156.24 ± 0.10	156.09 ± 0.06	156.54 ± 0.06	156.49 ± 0.10
$k=8, n=20$	65	431.98 ± 0.61	433.58 ± 0.70	436.20 ± 0.56	445.28 ± 0.45

Table 5.3: Performance of SA with different setups for evaluation function and mutation

it more. We note that for the two other cases ($k \leq n$), narrowing differences between distances lead to improve performance. We illustrate this on the instance with $k = 8$ and $n = 20$. We represent distance sets of several possible solutions and observe that the best solutions have the most narrow distributions. We compare two solutions in Figure 5.4 with $d_{\min} = 421$ and $d_{min} = 446$ which is the best solution found in [44]. Indeed, we note that $d_{\min} = 446$ has the narrowest peak. We formulate the hypothesis that this property may be beneficial for SA performance. We introduce below a new evaluation function taking into account this aspect.

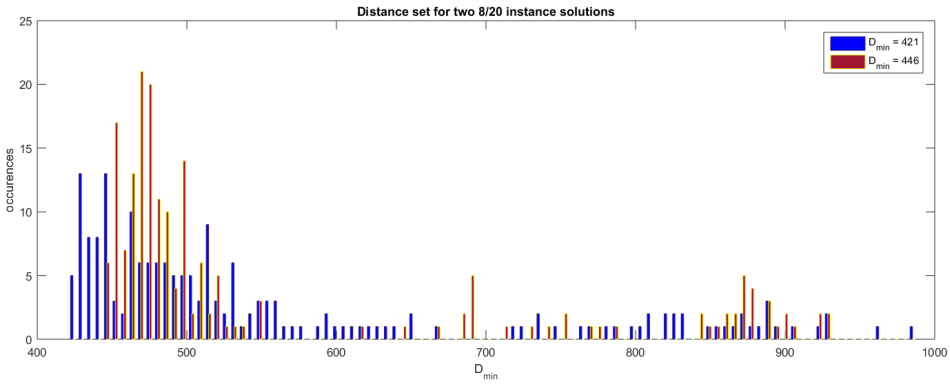

Figure 5.4: Distance sets of two 8/20 solutions

Definition of evaluation function ψ

We propose an evaluation function $\psi_{p,\sigma}$ to replace the usual function ϕ_p given by Eq. (5.2):

$$\psi_{p,\sigma} = \left(\sum_{i=1}^{\binom{n}{2}} w_i d_i^{-p} \right)^{\frac{1}{p}}, \text{ where } w_i = \frac{1}{\sqrt{\sum_{j=1}^{\binom{n}{2}} e^{-\frac{|d_j - d_i|^2}{\sigma^2}}}}. \tag{5.3}$$

The idea is to add weights $w_i \geq 1$ for each distance term d_i^{-p}. These weights determine if the distance is close to other ones. If a distance is far from the others, the weight will be high. Consequently, it forces the distances to be close to each other. However, $\psi_{p,\sigma}$ has two drawbacks: its complexity in $\mathcal{O}(n^4)$ and the need to find a good value for σ.
There are different ways to reduce this complexity. First, for instance, it is possible to consider only the differences which respect $|d_j - d_i|^2 \leq 5\sigma^2$. In this way, we avoid the calculations of terms that may be considered as negligible ($e^{-5} \ll 1$). Instead of summing up $\binom{n}{2}$ distances, we can randomly choose $\mathcal{O}(n)$ distances d_j.
The next section treats the second drawbacks by explaining the method to fix σ.

5.2.3 Tuning of parameter σ and its justification

Let us focus on the parameter σ: we would like to avoid tuning it with preliminary experiments for each instance. We will therefore search for a general expression depending on n

n \ k	3	4	5	6	7	8	9	10
3	6	7	8	12	13	14	18	19
4	6	12	14	20	21	26	28	33
5	11	15	24	27	32	40	43	50
6	14	22	32	40	47	54	62	68
7	17	28	40	52	62	72	81	91
8	21	42	50	66	80	91	103	116
9	22	42	61	82	95	114	128	144
10	27	50	82	95	113	**134**	158	175
11	30	55	82	111	**133**	157	184	211
12	36	63	94	142	158	**184**	213	243
13	41	70	**107**	143	184	**214**	246	279
14	42	78	**109**	**162**	220	**247**	282	318
15	48	89	**135**	**179**	228	**281**	**323**	363
16	50	94	154	**200**	**254**	**328**	364	**412**
17	56	102	163	221	*277*	343	413	462
18	57	114	**176**	**249**	**306**	**376**	469	515
19	62	**123**	**193**	**268**	**336**	408	491	576
20	66	**138**	**210**	**293**	**372**	**448**	528	645
21	69	149	232	**315**	**401**	482	570	**674**
22	82	**154**	**246**	347	433	525	623	**721**
23	82	**165**	260	364	**468**	566	667	*773*
24	83	**173**	**276**	**391**	506	609	**720**	837
25	89	183	294	419	**541**	657	*768*	**897**

Table 5.4: Highscores obtained with "all-purpose" tuning

and k. Looking at the definition of $\psi_{p,\sigma}$, this variable is introduced in order to regulate the order of magnitude of the exponential term. We see that σ should have approximately the same order of magnitude as the values taken by $|d_j - d_i|^2$.
This is why we try to give the expression of a linear function of k and n which is similar to typical values $|d_j - d_i|^2$. To establish it, we study the variance of $M(k,n)$: the tuning of σ is founded on Theorem 5.1.

Theorem 5.1. *$M(k,n) \sim \mathcal{N}\left(\frac{kn(n+1)}{6}, g(k,n)\right)$ with $g(k,n) \sim \frac{7kn^4}{180} + \mathcal{O}(n^3)$.*

Proof. Thanks to [52], we know that $\mathbb{E}(M(k,n)) = \frac{kn(n+1)}{6}$. We note (P_1, P_2) the random variable that gives any couple of points for an instance. The random variable $M(k,n)$ is a function of (P_1, P_2). For any $1 \leq j \leq k$, we note $Y(j) = (P_1(j) - P_2(j))^2$ and get $M(k,n) = \sum_{j=1}^{k} Y(j)$. As $Y(i)$ and $Y(j)$ are independent if $i \neq j$, we note $Y(i) = Y$ to keep the notation simple. If k is high enough, we apply the Central Limit Theorem: $M(k,n) \sim \mathcal{N}\left(\frac{kn(n+1)}{6}, k\mathbb{V}\text{ar}(Y)\right)$. We focus first on $\mathbb{E}(Y^2) = \mathbb{E}((P_1(j) - P_2(j))^4)$:

$$\mathbb{E}(Y^2) = \frac{\sum_{x=1}^{n}\sum_{y\neq x}(x-y)^4}{\frac{n(n-1)}{2}} = \frac{2\left(n\sum_{z=1}^{n-1} z^4 - \sum_{z=1}^{n-1} z^5\right)}{n(n-1)} = \frac{n^4}{15} + \mathcal{O}\left(n^3\right).$$

We thus deduce $\mathbb{V}\text{ar}(Y) = \mathbb{E}(Y^2) - \mathbb{E}(Y)^2 = \frac{n^4}{15} - \frac{n^4}{36} + \mathcal{O}\left(n^3\right) \sim \frac{7n^4}{180}$, and thus $g(k,n) = k\mathbb{V}\text{ar}(Y) \sim \frac{7n^4}{180}$. □

A good way to tune σ^2 is to express it as a linear function of the variance of the distances, *i.e.* the random variable $M(k,n)$. As we have just showed, this variance follows the function $g(k,n)$ above. We will therefore have the following expression for σ^2: ckn^4 where c is a

constant. We just need to determine the value of c. for this, we will proceed experimentally and give a different value for each different case.
In the amodal case $n \geq 2k$, we assume $\sigma^2 = ckn^4$. According to several experiments series, we identify a good compromise with $c = \frac{1}{300}$.
In the case unimodal case $n \leq k$, ψ does not bring more interesting results than ϕ. It is equivalent to assuming c to be very large ($c \to \infty$).
Finally, the bimodal case $k \leq n \leq 2k$ which is an intermediary of the two previous cases, can be tuned with $\sigma = 2ckn^4$. This proposition does not obviously represent the best tuning for all possible instances but gives an efficient and simple solution for the tuning of σ.
It is necessary to mention that the case $k \leq n \leq 2k$ is the case where tuning is essential: to be as efficient as possible, the value of σ has to be carefully selected.

5.2.4 Experimental results

Table 5.3 shows the impact of $\psi_{p,\sigma}$ on the SA performance with mutations m_2 and *1D-move*. We keep the same experimental setup as in Subsection 5.2.1: SA makes a thermal linear descent, the results presented come out from 100 runs and the average is within the 95% confidence interval. The evaluation function $\psi_{p,\sigma}$ brings better results than ϕ_p for the bimodal and amodal cases, in particular when paired with *1D–move*. In particular, the case where $k = 8$ and $n = 20$ shows a high increase in quality.
In Table 5.4 we update scores for the same instances as in [44]. The results were produced with 10^7 iterations and $p = 5$. Our function $\psi_{p,\sigma}$ is used when $k \leq n$, ϕ_p is used elsewhere. We note in bold type improved results and in italics results worse than [44]. For $4 \leq k \leq 8$, the use of *1D–move* and $\psi_{p,\sigma}$ allows us to exceed a large number of scores but this improvement is less significant for other values. For $k = 3$, we suppose that the new tools are not able to outperform previous results because the results are already optimal or very good. For $k = \{9, 10\}$, a credible hypothesis is that the value of $\frac{n}{k}$ is so close to 1 that the effect of $\psi_{p,\sigma}$ is weak. Generally, results could be better with a specifically adapted tuning. Here, we established temperature, p and σ by making compromises between all the instances. However, in a real-life case, when treating complex systems, we work on a defined instance with k and n fixed. In such circumstances, we naturally advice to customize the tuning of the different parameters by making preliminary experiments on this very instance. We expect that such an approach would produce results outperforming those in Table 5.4.

5.2.5 Conclusion

We presented the state of the art of the algorithms used to solve the LHD-CP, focusing at first on the Simulated Annealing metaheuristic and continuing with other heuristics applied to constructing LHDs. We then presented improvements we made to SA for LHD in the form of a new mutation and a new evaluation function, both allowing to overcome the best results obtained in the literature. We go on with the implementation of Simulated Annealing for the pLHD-CP.

5.3 SA for completion

As we did not find any algorithm with performance guarantees for the pLHD-CP, and even worse, found that it is inapproximable for most practical cases, we use a heuristic algorithm to treat it. As the best performing heuristic for the LHD-CP is the Simulated Annealing metaheuristic, we applied it to the pLHD-CP. While the evaluation function ϕ_p can be used without a change, mutations need to be adapted. We adapt and compare two mutations, whose results vary depending on the number of points already chosen in the pLHD, and combine them using a method inspired by bandits strategies.

5.3.1 Mutation targeting "relatively empty" hypercubes

The mutations conceived for the construction problem may be reused in the completion context. The single difference is that only the points which have not been fixed in advance may be involved in them. In the remainder we use the term **authorized points** to refer to these points.

The choice of an appropriate mutation is conditioned by the number of points set *a priori*. When the design is entirely empty, the completion becomes a construction and *1D-move* is a natural choice as it is the best mutation to this day, as we have seen in Section 5.2.1.

However, even though *1D-move* is efficient for the construction problem, it behaves very poorly for the completion. The behavior of this mutation when completing a Maximin LHD needs to be adapted to this case. Otherwise, unauthorized critical points may restrict the mutation freedom and, as a consequence, the algorithm may be stuck in a local minimum. As a matter of fact, in the extreme case, the mutation becomes totally blocked. We will now describe in detail a situation where the use of the *1D-move* mutation leads to oscillations. This will be illustrated in Fig. 5.5. If the minimal distance occurs once in the design and one of the points involved in it is fixed, one and only point can be chosen by the mutation. In Fig. 5.5 we use the following notation. Points U, A, and C stand for unauthorized, authorized, and critical points, respectively. Point U_C indicates a fixed point which is involved in the minimal distance which spans between it and C_A. Point C_A is the only authorized and critical point in our example.

Once the mutation has chosen a dimension (we fix our attention on the horizontal axis in Fig. 5.5), the sequence of points of type $UCAU$ on this dimension imposes the choice of the other point to exchange one coordinate. In our example, on the horizontal dimension this sequence is $U_C C_A A_1 U_1$. Mutation *1D-move* will exchange the coordinates of A_1 and C_A on this dimension (C_A becomes C_A''' while A_1 becomes A_1'''). If C_A''' remains the only critical point and the same dimension is chosen again, the sequence will be of $UACU$ type, $U_C A_1''' C_A''' U_1$ in our case, and C_A''' will exchange its coordinate with A_1''' going back to its initial position. If such a sequence exists on each dimension (as this is a case in our example in Fig. 5.5) and every exchange maintains C_A (or C_A', or C_A'', or C_A''') as the one and only authorized critical point, the algorithm will be stuck oscillating ($C_A \leftrightarrow C_A'''$, $C_A \leftrightarrow C_A''$, etc.) or turning around ($C_A \rightarrow C_A''' \rightarrow C_A'' \rightarrow C_A' \rightarrow C_A$). In our example, mutation *1D-move* cannot exchange one coordinate of the critical point with one coordinate of A_3 or A_4 as they are "too far" from C_A. Consequently, it will never produce a configuration with a point added either on position C_A^* or position C_A^{**}.

Now that we have seen that there are situations where 1D-move leads to oscillations, we will design a new mutation called *cmpl1D-move* that is based on 1D-move but that avoids such problem. The first major difference with the previous mutations is that it selects an **oriented** dimension, which means that we take into account a direction when looking for another point to exchange coordinates. Indeed, as seen in the example above, the selection of the neighbor in one dimension, performed by *1D-move*, may be constrained to only one possibility. The second major difference is that mutation *cmpl1D-move* does not limit itself to choose points which are neighbors: once the direction has been fixed, it looks for the point with the minimal distance on the dimension and direction selected. Thus, considering the direction in which we look for the authorized point beforehand ensures that, given a critical point and a dimension, *cmpl1D-move* will have two outcomes (in the two opposite directions, possibly not symmetrical one to another with reference to the critical point on the dimension selected). In the example given in Fig. 5.5, critical point C_A can be found in positions C_A^* or C_A^{**} after the mutation, if the *cmpl1D-move* selects respectively the horizontal axis leftward (for C_A^* obtained by the exchange with A_3 which becomes A_3^*), or the vertical axis downward (for C_A^{**} obtained by the exchange with A_4 which becomes A_4^{**}). In summary:

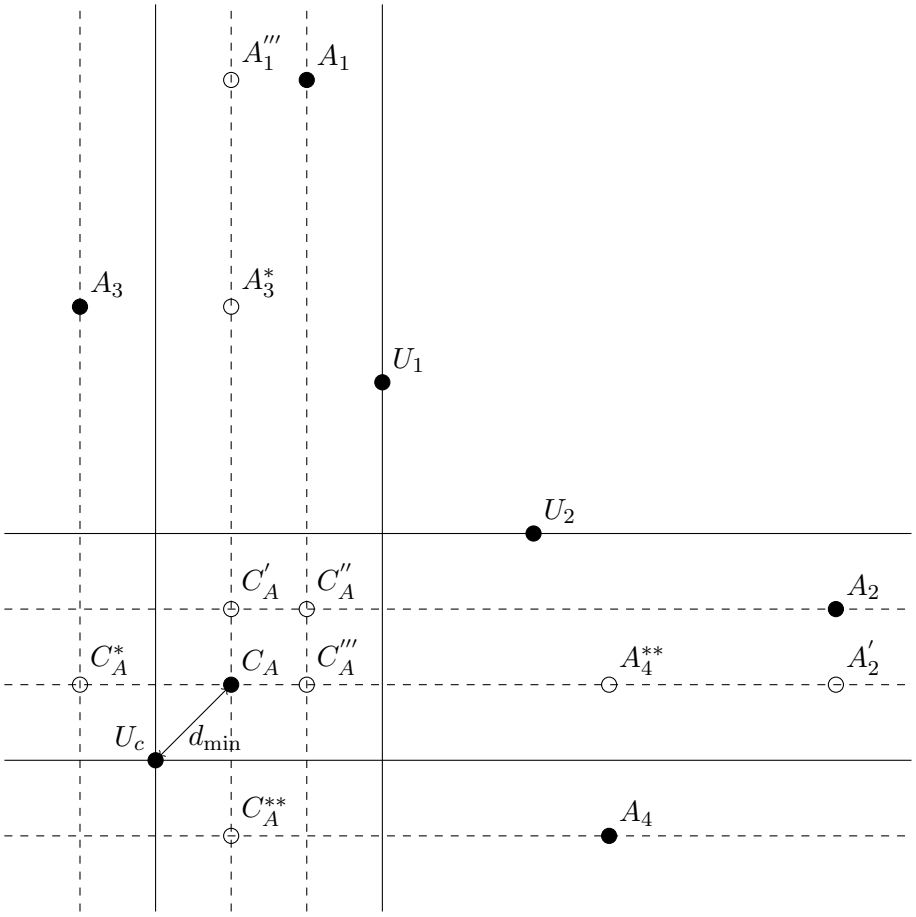

Figure 5.5: The points marked with solid black circles form the initial configuration for this example. The smallest distance between lines is equal to one. Point C_A is the one and only authorized critical point. It can exchange coordinates with points A_1 and A_2 as the coordinates of C_A and A_1 differ by one on the horizontal axis and those of C_A and A_2 differ by one on vertical axis. These coordinates of are represented by dotted lines. Unauthorized points U_c, U_1 and U_2 prevent from choosing the other coordinates, represented by solid lines. Mutation *1D-move* would make point C_A oscillate between three positions C_A', C_A'', and C_A''' as it cannot exchange the coordinates of C_A with either A_3 or A_4. By contrast, mutation *cmpl1D-move* can do these exchanges and insert a point on either position C_A^* or position C_A^{**}.

***cmpl1D-move*:**

1. Select one critical point $p^{(1)}$, one dimension j and the sign $+/-$ of the coordinate difference at random.
2. Find the authorized $p^{(2)}$ such that the coordinate difference $\left|p_j^{(1)} - p_j^{(2)}\right|$ is minimal.
3. Exchange the coordinates on the dimension j with respect to the chosen sign which determines the direction.
4. If there is no authorized point on the dimension and direction chosen, perform mutation *m2*.

As the restriction of the search space, induced by the fixed points, gets stronger when the number of these points increases, the mutation *cmpl1D-move* would not look for a solution sufficiently far from a current configuration. Its interest is therefore limited to instances in which a relatively small number of points is fixed in advance. At the contrary, m_2 performs best when a lot of points are fixed. We therefore try to combine both mutations to obtain one mutation adapted to every case.

5.3.2 Bandit-driven mutation

At this stage we conclude that there is not a single mutation which could cope with the entire spectrum of incomplete hypercubes, from "almost empty" to "almost filled up". We know, however, that *cmpl1D-move* works well for the first category of instances while *m2* is well suited to explore the search space of the second one. As we do not want to change the mutation depending on the instance, we would like to create a decision-making algorithm for choosing the mutation on-the-fly. In order to do this, we consider that at each iteration of SA, we have at our disposal several mutations and that choosing a mutation is a multi-armed bandit (MAB) problem as described in [2].
The MAB problem can be defined by several random variables X_i, where i is the index of a gambling machine whose output is given by X_i. At each step, a machine j is chosen and a reward which is the realization of X_j is given. The goal is to maximize the sum of the rewards. In our case, the choice of the mutation would correspond to the choice of the machine and the reward, the evaluation of the new individual, *i.e.* a configuration in the SA context.
However, in our problem, the choice of a mutation changes the state of the system. We make the hypothesis that this is equivalent to a modification of the random variables behind each machine. With this hypothesis, our problem can be seen as a classical variant of the MAB problem: the non-stationary multi-armed bandit (NSMAB) problem [3].
An algorithm has been proposed in [20] to solve this variant: the sliding-window UCB algorithm. The principle is to choose the next gambling machine based on a trade-off between the exploitation of machines that previously gave good rewards and the exploration of other machines. At each step, a score is computed for each machine and the machine with the highest score is selected. This score is the sum of an exploitation part and an exploration part. The exploitation is based on the average reward of the machine. To handle the non-stationary aspect of the problem, the average is computed only on the last results of the machine. The exploration part will grow logarithmically when the machine is not selected. Our own algorithm will be based on this one with an adaptation to the nature of our problem. In SA, the difference in energy led by mutations will decrease during the algorithm and therefore the rewards will decrease in the corresponding bandit problem. As a result, we will handle the exploration in a different way.
The idea of our algorithm is to compute the average reward for each mutation based on their last results as in the sliding-window UCB algorithm. However, we chose the mutation based

on a probability depending on this average instead of having a deterministic algorithm. This and the fact that the rewards decrease in our problem allow us to get ride of the exploration part. Indeed, less exploited machines will have older rewards in their corresponding stored window. Those rewards, according to the previous remark, will likely be higher than the reward recently given, thus the average of less exploited machines tend to increase over time (and *de facto* the probability of choosing those machines will increase).
Here is a formal description of our algorithm. Let us note W_i the set of size w of the last rewards x_i for mutation i. At each step, we select the next mutation according to the probability p_i computed as specified by:

$$p_i = \frac{\exp(\overline{X_i})}{\sum_j \exp(\overline{X_j})}, \quad \text{where} \quad \overline{X_i} = \frac{1}{w} \sum_{x_i \in W_i} \exp(x_i). \tag{5.4}$$

In the following, we apply this algorithm to the mutations *m2* and *cmpl1D-move* and name the resulting mutation Bandit Driven Mutation (BDM). We note that it can be applied to any number of mutations.

BDM:

1. Compute the probability of mutations *m2* and *cmpl1D-move* according to Eq. 5.4.
2. Randomly select the mutation to be used based on these probabilities.
3. Store the corresponding gain x_i.
4. Apply the chosen mutation.

5.3.3 Numerical experiments

We chose to base our experiments on the problem of dimension 4 and size 75 as was done in [44]. The experiments are performed for the hypercube taken from `spacefillingdesigns.nl`, with the distance expressed by the Euclidean norm $\mathcal{L}_2$. According to the site mentioned, this hypercube has $D_{\min}$ of 867 (we obtained, however, a hypercube with a score of 889 during our experiments).
Unless it is stated otherwise, the number of mutations of the annealing used for the experiments is 10^5. We remark that for mutation *m3*, experiments are realized with four times less mutations than for the others, because this mutation evaluates the $D_{\min}$ of the configuration once per dimension of the instance, thus we limited the total number of mutations performed in order to fairly compare the different types of mutations. Also, the temperature follows a linear cooling scheme that decreases the temperature every 100 mutations, from an initial acceptance probability of 0.4 down to 0.
Incomplete hypercube instances are obtained by removing points according to a uniform distribution. The results are obtained by averaging over 200 incomplete instances except for some points in Table 5.6 and Figure 5.6 where a significant difference could not be obtained and therefore the average over 2000 runs was used. The same set of 200 (respectively 2000) instances of hypercubes with fixed points randomly generated with 200 (respectively 2000) different seeds were used as starting hypercubes across all algorithms.
We will first compare the performance of existing mutations with the one we designed for the completion problem: *cmpl1D-move* (Section 5.3.1) in function of the number of deleted points in the hypercube. The results are presented in Table 5.5.
The results in bold type correspond to the best average for a given number of deleted points. The last line (75 deleted points) of this table corresponds to the construction problem.
We note that the mutation *1D-move* which gives the best results for the construction problem performs very poorly for the completion problem, which was explained in Section 5.3.1. The mutation we proposed: *cmpl1D-move* is successful as it obtains the best average for 35 to

Deleted points	mutations				
	m1	**m2**	**m3**	**1D-move**	**cmpl1D-move**
5	846±49	**864±14**	858± 32	331±54	853± 32
15	810±55	**854±20**	832± 42	173±30	710± 57
25	745±36	769± 28	**791±35**	137±28	732± 32
35	735±14	728± 14	752± 13	126±25	**767±15**
45	735±11	724± 10	745± 8	134±24	**780± 9**
55	740±10	722± 10	747± 8	173±31	**799± 7**
65	747±11	724± 10	753± 7	325±53	**818± 6**
75	751±12	730± 10	761± 8	**844± 5**	**844± 6**

Table 5.5: The mean score in function of the number of deleted points, for an instance of size 75, in four dimensions, with $D_{\min} = 867$

75 deleted points. However, it is interesting to observe that this is not the case for a smaller number of deleted points where mutations *m2* and *m3* produce better scores.

This suggests that the problem behaves very differently depending on the number of deleted points. This observation leads us to propose the mutation *BDM* that is able to pick up the best mutation depending on the instance.

We will now present in Table 5.6 and Figure 5.6 the results of the comparison of *BDM* with mutations *m2* and *cmpl1D-move* of which *BDM* is composed.

Deleted points	mutations		
	m2	**cmpl1D-move**	**BDM**
5	**864±11**	858±18	863± 12
15	**853±14**	709±41	830± 25
20	**819±21**	713±31	811± 24
25	766± 19	736±22	**794±20**
30	736± 11	750±14	**780±14**
35	727± 8	763±10	**773± 9**
40	725± 7	774± 7	**775± 6**
45	724± 6	**782± 6**	779± 5
55	723± 6	**798± 4**	791± 4
65	726± 6	**818± 3**	807± 4
75	731± 6	**843± 3**	830± 3

Table 5.6: Comparison of the mean score of *BDM* and its components: m_2 and *cmpl1D-move*

The size of the sliding window w discussed in Section 5.3.2 is arbitrarily set for our experiments to 100 (for a total of 10^5 mutation steps during an annealing process).

We see that for the extreme cases (a large or a small number of deleted points), *BDM* obtains scores close to those produced by the best-performing mutation. Our goal of having a mutation that can handle both cases has therefore been achieved.

On top of that, for intermediate values of deleted points, *BDM* reaches significantly better results than both other mutations. This shows that a dynamic choice of a mutation during a single run based on a bandit formula can not only perform as well as each individual mutation but also combine the advantages of the different mutations to obtain even a better result.

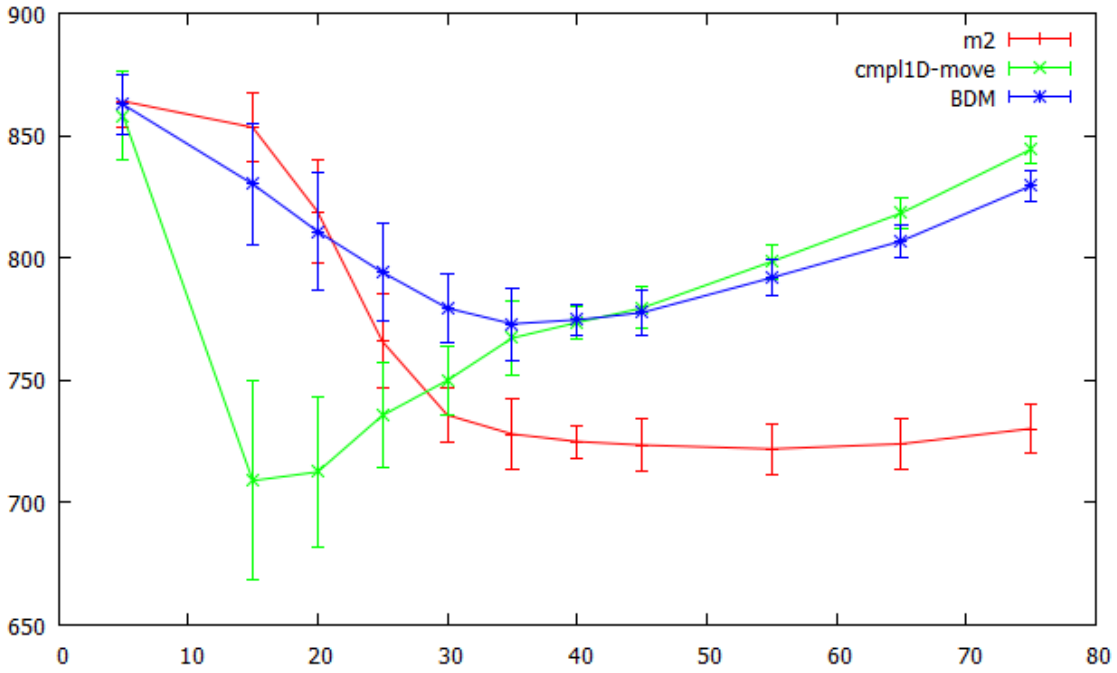

Figure 5.6: Mean $D_{\min}$ with confidence interval for mutations: *m2*, *cmpl1D-move*, and *BDM* in function of the number of deleted points

5.4 Conclusion

We presented a state of the art of the methods used to produce maximin LHDs. We improved the most efficient technique, the Simulated Annealing metaheuristic, by designing a new mutation and a new evaluation function, both performing better than their equivalent in the literature. This allowed us to beat several highscores for numerous instances. We applied this metaheuristic to the pLHD-CP, adapting the mutations used for the LHD-CP, and improved them by using a bandit method to choose a mutation.

Chapter 6

Conclusion

6.1 Summary of contributions

Two goals motivated this study: on the one hand, designing algorithms for constructing LHDs and improving existing ones, and on the other hand, finding the complexity class to which the LHD-CP belongs. The first goal has been met, and significant advances have been made toward the second one.
After giving a state of the art of metamodeling we focused on a sampling method, maximin Latin hypercube designs, which have the required qualities for metamodeling but are difficult to construct. We generalize the problem of constructing maximin LHDs (LHD-CP) by defining a new problem: the maximin completion of partial LHDs (pLHD-CP).
We studied the complexity of the pLHD-CP and proved that it is NP-complete in dimensions $k \geq 3$ for norm $\mathcal{L}_1$, a result we extended to all norms. We also proved that it is NP-complete in the two-dimensional space for norm $\mathcal{L}_1$, norms $\mathcal{L}_p$ with $p \in \mathbb{N}$, and norm $\mathcal{L}_\infty$.
We then searched for guarantees of performance for algorithms concerning both problems and found that the pLHD-CP is inapproximable for all norms in dimensions $k \geq 3$. We also gave an upper bound for an approximation ratio in dimension $k = 2$ for norm $\mathcal{L}_\infty$. We designed IES, an approximation algorithm for the LHD-CP which is, to the best of our knowledge, the first approximation algorithm to solve this problem. We proved its approximation ratio by introducing two new upper bounds for the LHD-CP.
We finished by a study of heuristic algorithms for both problems. We started by describing the state of the art of algorithms used to solve the LHD-CP and we proposed two modifications for the best performing algorithm, the Simulated Annealing algorithm. The first is a new mutation which makes the local search smoother by reducing the effect of the mutation over the examined solutions. The second is an evaluation function that takes advantage of the shape of the distribution of distances we observed in good LHDs. These two modifications allowed us to produce better LHDs than those found in the literature for numerous instances of the LHD-CP.
We also adapted the Simulated Annealing algorithm to the pLHD-CP by modifying the two best-performing mutations for the LHD-CP. We observed that the best performing mutation depends on the instances of the problem and we designed a super-heuristic inspired by bandit methods able to choose the best mutation depending on the situation and even outperforming each individual mutation in some cases.

6.2 Directions for further research

The complexity of the LHD-CP remains an open problem.
Proving its supposed NP-completeness by finding a reduction to another problem is challenging considering the small amount of information which defines each instance. The instance

is made up of only the two positive natural numbers which are the LHD size and dimension. Proving that the construction problem is in P (if this is the case) would require two elements. The first would be tight upper bound. While we did find an upper bound of the right order, described in Section 4.3.1, it is not thight. The second element would be an optimal polynomial-time algorithm. While we designed a polynomial-time approximation algorithm, it is not an optimal algorithm.

While the pLHD-CP has been found to be NP-complete in most cases, in two dimensions it has only been proven NP-complete for the usual norms $\mathcal{L}_\infty$ and $\mathcal{L}_p$, with $p \in \mathbb{N}^*$. Furthermore, its approximability is also unknown on the plane. Proving its inapproximability for one norm would lead to proving its inapproximability and NP-completeness for all norms with the same technique we used for dimensions $k \geq 3$.

The approximation algorithm we designed, the Inflate, Expand and Stack algorithm (IES), has the advantage of being very fast. However, we did not successfully use it as a starting point for the heuristic algorithms we studied. Nonetheless, it could be used in this way in algorithms using different mutations and thus a different search space. Finding better upper bounds for the problem would also give a better, more accurate, approximation ratio for IES.

A work of considerable volume has been done in the operations research field to build maximin LHDs and thus numerous heuristic algorithms have been developed to this end. The mutation and evaluation function we designed for the Simulated Annealing algorithm may be used for other heuristic algorithms. Notably, the evaluation function we proposed is not specific to LHDs and can be used for other types of maximin designs.

The pLHD-CP, has been introduced in this thesis and has not been studied out of this scope. It would be interesting to see what other heuristic algorithms could be used to solve it. As we oberved, this problem behaves differently when the number of points already fixed is low or high, different algorithms, mutations or evaluation functions should be used for each individual case.

Bibliography

[1] P. Audze and V. Eglais. New approach for planning out of experiments. Problems of dynamics and strengths, 35:104–107, 1977.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. Machine Learning, 47(2–3):235–256, 2002.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. SIAM Journal on Computing, 32(1):48–77, 2002.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer Science & Business Media, 2012.

[5] D. Baer. Punktverteilungen in Würfeln beliebiger Dimension bezüglich der Maximum-norm. Wiss. Z. Pädagog. Hochsch. Erfurt/Mühlhausen, Mathematik-Naturwissenschaften. Reihe, 28:87–92, 1992.

[6] S. J. Bates, J. Sienz, and V. V. Toropov. Formulation of the optimal Latin hypercube design of experiments using a permutation genetic algorithm. In Proc. of AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, pages 19–22, 2004.

[7] J. Baxter. Local optima avoidance in depot location. Journal of the Operational Research Society, pages 815–819, 1981.

[8] P. Bergé, **K. Le Guiban**, A. Rimmel, and J. Tomasik. Search space exploration and an optimization criterion for hard design problems. In Proc. of GECCO (compagnon), pages 43–44, July 2016. Full version aviable on CoRR: `https://arxiv.org/abs/1608.07225`.

[9] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. Electronic Colloquium on Computational Complexity (ECCC), (049), 2003.

[10] A. W. Blom, S. Setoodeh, J. M. A. M. Hol, and Z. Gürdal. Design of variable-stiffness conical shells for maximum fundamental eigenfrequency. Computers & Structures, 86(9):870–878, 2008.

[11] G. E. P. Box and N. R. Draper. Empirical model-building and response surfaces, volume 424. Wiley New York, 1987.

[12] G. E. P. Box and S. J. Hunter. The 2 $k-p$ fractional factorial designs. Technometrics, 3(3):311–351, 1961.

[13] J. F. M. Burkert, F. Maugeri, and M. I. Rodrigues. Optimization of extracellular lipase production by geotrichum sp. using factorial design. Bioresource Technology, 91(1):77–84, 2004.

[14] C. J. Colbourn. The complexity of completing partial Latin squares. Discrete Applied Mathematics, 8(1):25–30, 1984.

[15] A. Darmann, U. Pferschy, J. Schauer, and G. J. Woeginger. Paths, trees and matchings under disjunctive constraints. Discrete Applied Mathematics, 159(16):1726–1735, 2011.

[16] N. Dyn, D. Levin, and S. Rippa. Numerical procedures for surface fitting of scattered data by radial functions. SIAM Journal on Scientific and Statistical Computing, 7(2):639–659, 1986.

[17] O. Ekren and B. Y. Ekren. Size optimization of a pv/wind hybrid energy conversion system with battery storage using response surface methodology. Applied Energy, 85(11):1086–1101, 2008.

[18] G. Gan and X. S. Lin. Efficient Greek calculation of variable annuity portfolios for dynamic hedging: A two-level metamodeling approach. North American Actuarial Journal, 21(2):161–177, 2017.

[19] M. R. Garey and D. S. Johnson. Computers and intractability. W.H. Freeman, New York, 1979.

[20] A. Garivier and E. Moulines. On upper-confidence bound policies for switching bandit problems. In Proc. of ALT, pages 174–188. Springer, 2011.

[21] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. Machine Learning, 3(2):95–99, 1988.

[22] A. Grosso, A. R. M. J. U. Jamali, and M. Locatelli. Finding maximin Latin hypercube designs by iterated local search heuristics. European Journal of Operational Research, 197(2):541–547, 2009.

[23] L. Gu. A comparison of polynomial based regression models in vehicle safety analysis. In ASME Design Engineering Technical Conferences, ASME Paper No.: DETC/DAC-21083, 2001.

[24] C. Hamelain, **K. Le Guiban**, A. Rimmel, and J. Tomasik. Bandits help simulated annealing to complete a maximin Latin hypercube design. 2017. Submission to CPAIOR in November 2017.

[25] J. H. Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, 1992.

[26] J. K. Hunter and B. Nachtergaele. Applied analysis. World Scientific Publishing Co Inc, 2001.

[27] B. Husslage, G. Rennen, E. R. Van Dam, and D. den Hertog. Space-filling Latin hypercube designs for computer experiments. Tilburg University, 2006.

[28] M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. Journal of Statistical Planning and Inference, 26(2):131–148, 1990.

[29] M. Hall Jr. Distinct representatives of subsets. Bulletin of the American Mathematical Society, 54(10):922–926, 1948.

[30] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. Inf. Process. Lett., 37(1):27–35, 1991.

[31] A. I. Khuri and S. Mukhopadhyay. Response surface methodology. Wiley Interdisciplinary Reviews: Computational Statistics, 2(2):128–149, 2010.

[32] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

[33] J. P. C. Kleijnen. Statistical tools for simulation practitioners. Marcel Dekker, Inc., 1986.

[34] J. P. C. Kleijnen. Kriging metamodeling in simulation: A review. European Journal of Operational Research, 192(3):707–716, 2009.

[35] M. Liefvendahl and R. Stocki. A study on algorithms for optimization of Latin hypercubes. Journal of Statistical Planning and Inference, 136(9):3231–3247, 2006.

[36] M. McKay, R. Beckman, and W. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics, 21(2):239–245, 1979.

[37] M. Morris and T. Mitchell. Exploratory designs for computational experiments. Journal of Statistical Planning and Inference, 43(3):381–402, 1995.

[38] T. Mukhopadhyay, T. K. Dey, R. Chowdhury, and A. Chakrabarti. Structural damage identification using response surface based multi-objective optimization: a comparative study. Arabian Journal for Science and Engineering, 40(4):1027–1044, 2015.

[39] M. A. Nik, K. Fayazbakhsh, D. Pasini, and L. Lessard. Surrogate-based multi-objective optimization of a composite laminate with curvilinear fibers. Composite Structures, 94(8):2306–2313, 2012.

[40] N. Oler. A finite packing problem. Canad. Math. Bull, 4(2), 1961.

[41] A. B. Owen. Orthogonal arrays for computer experiments, integration and visualization. Statistica Sinica, pages 439–452, 1992.

[42] Lei Peng, Li Liu, Teng Long, and Wu Yang. An efficient truss structure optimization framework based on cad/cae integration and sequential radial basis function metamodel. Structural and Multidisciplinary Optimization, 50(2):329–346, 2014.

[43] J. N. Reddy. An introduction to the finite element method, volume 2. McGraw-Hill New York, 1993.

[44] A. Rimmel and F. Teytaud. A survey of meta-heuristics used for computing maximin Latin hypercube. In Proc. of EvoCOP, pages 25–36. Springer, 2014.

[45] H. J. Ryser. A combinatorial theorem with an application to Latin rectangles. Proceedings of the American Mathematical Society, 2(4):550–552, 1951.

[46] D.M.Y. Sommerville. An Introduction to the Geometry of n Dimensions. Methuen & Co., London, 1929.

[47] G. Sun, X. Song, S. Baek, and Q. Li. Robust optimization of foam-filled thin-walled structure based on sequential Kriging metamodel. Structural and Multidisciplinary Optimization, 49(6):897–913, 2014.

[48] **K. Le Guiban**, A. Rimmel, M.-A. Weisser, and J. Tomasik. Completion of partial Latin Hypercube Designs: NP-completeness and inapproximability. Journal of Theoretical Computer Science, section A, 2017. submitted paper.

[49] **K. Le Guiban**, A. Rimmel, M.-A. Weisser, and J. Tomasik. The first approximation algorithm for the maximin Latin hypercube design problem. Operations Research, 2017. Accepted, in press. doi:10.1287/opre.2017.1665.

[50] P. Tian, J. Ma, and D.-M. Zhang. Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism. European Journal of Operational Research, 118(1):81–94, 1999.

[51] E. R. van Dam, B. Husslage, D. den Hertog, and H. Melissen. Maximin Latin hypercube designs in two dimensions. Operations Research, 55(1):158–169, February 2007.

[52] E. R. van Dam, G. Rennen, and B. Husslage. Bounds for maximin Latin hypercube designs. Operations Research, 57(3):595–608, 2009.

[53] F. A. C. Viana, G. Venter, and V. Balabanov. An algorithm for fast optimal Latin hypercube design of experiments. International Journal for Numerical Methods in Engineering, 82(2):135–156, 2010.

[54] G. Vicente, A. Coteron, M. Martinez, and J. Aracil. Application of the factorial design of experiments and response surface methodology to optimize biodiesel production. Industrial Crops and Products, 8(1):29–35, 1998.

[55] S. Volpi, M. Diez, N. J. Gaul, H. Song, U. Iemma, K. K. Choi, E. F. Campana, and F. Stern. Development and validation of a dynamic metamodel based on stochastic radial basis functions and uncertainty quantification. Structural and Multidisciplinary Optimization, 51(2):347–368, 2015.

[56] G. G. Wang. Adaptive response surface method using inherited Latin hypercube design points. Transactions-American Society of Mechanical Engineers Journal of Mechanical Design, 125(2):210–220, 2003.

[57] D. B. West et al. Introduction to graph theory, volume 2. Prentice Hall, Upper Saddle River, 2001.

[58] J. Wu and M. S. Hamada. Experiments: planning, analysis, and optimization, volume 552. John Wiley & Sons, 2011.

[59] Y. Yamini, A. Saleh, and M. Khajeh. Orthogonal array design for the optimization of supercritical carbon dioxide extraction of platinum (iv) and rhenium (vii) from a solid matrix using cyanex 301. Separation and Purification Technology, 61(1):109–114, 2008.

[60] K. Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. Journal of Statistical Planning and Inference, 90(1):145–159, 2000.

[61] H. Zhao, Z. Yue, Y. Liu, Z. Gao, and Y. Zhang. An efficient reliability method combining adaptive importance sampling and kriging metamodel. Applied Mathematical Modelling, 39(7):1853–1866, 2015.

[62] G. Zhu and H. Ju. Determination of naproxen with solid substrate room temperature phosphorimetry based on an orthogonal array design. Analytica Chimica Acta, 506(2):177–181, 2004.

Résumé de la thèse

Un hypercube latin (LHD) est un ensemble de n points en dimension k, à coordonnées entières, contenus dans un hypercube de taille n^k, et tel que les points ne partagent pas de coordonnées sur aucune dimension. Un LHD maximin est un LHD tel que la distance de séparation, c'est-à-dire la distance minimale entre deux points, est maximale. Les LHDs maximin sont particulièrement utilisés pour la construction de métamodèles en raison de leurs bonnes propriétés pour l'échantillonnage. Comme la plus grande partie des travaux concernant les LHD se sont concentrés sur leur construction par des algorithmes heuristiques, nous avons décidé de produire une étude détaillée du problème, et en particulier de sa complexité et de son approximabilité en plus des algorithmes heuristiques permettant de le résoudre en pratique.

Pour conduire cette étude, nous avons généralisé le problème de construction d'un LHD maximin en définissant le problème de compléter un hypercube latin entamé en respectant la contrainte maximin: étant donné un LHD partiel (un LHD auquel il manque des points), lui ajouter des points de manière à obtenir un LHD avec une distance de séparation maximum. Le sous-problème dans lequel le LHD partiel est vide correspond au problème de construction de LHD classique.

Dans le second chapitre, nous étudions les métamodèles et décrivons les différentes techniques utilisées, avec les régressions polynomiales, la méthode des surfaces de réponses, le krigeage et les fonctions de base radiales. Nous décrivons les différentes méthodes d'échantillonage utilisées pour les métamodèles étudiés, les plans factoriels fractionnaires, les réseaux orthogonaux et les hypercubes latins. Cela nous conduit à définir formellement les problèmes de construction d'hypercubes latins maximin et de complétion maximin d'hypercubes latins partiels, en tant que problèmes de décision et d'optimisation.

Dans le troisième chapitre, nous avons étudié la complexité du problème de complétion et avons prouvé qu'il est NP-complet pour toutes les normes en dimension $k \geq 3$, et pour les normes usuelles (*i.e.* les normes $\mathcal{L}_p$, avec $p \in \mathbb{N}$ et la norme $\mathcal{L}_\infty$) dans le plan, à l'aide de réduction polynomiales à partir des problèmes de complétion de carrés latins ainsi que du problème $(3, B_2)$-SAT qui est une variante de 3-SAT. N'ayant pas déterminé la complexité du problème de construction, nous avons cherché des garanties de performances pour les algorithmes résolvant les deux problèmes.

Dans le quatrième chapitre, nous avons prouvé que le problème de complétion n'est approximable pour aucune norme en dimensions $k \geq 3$. Nous avons également prouvé un résultat d'inapproximabilité plus faible pour la norme $\mathcal{L}_\infty$ en dimension $k = 2$. D'un autre côté, nous avons proposé le premier algorithme d'approximation pour le problème de construction, l'algorithme IES, dont nous avons calculé le rapport d'approximation grâce à deux bornes supérieures que nous avons établies.

En plus de l'aspect théorique de cette étude, nous avons travaillé dans le cinquième chapitre sur les algorithmes heuristiques, et en particulier sur la métaheuristique du recuit simulé. Nous avons proposé une nouvelle mutation pour le problème de construction, en prenant en compte le fait que le recuit simulé est plus performant lorsque la mutation ne change que légèrement la solution. Après avoir remarqué que la distribution des distances dans des bonnes solutions avait une forme particulière, nous avons développé une nouvelle fonction

d'évaluation prennant ce fait en compte.. Ces deux éléments ont permis d'améliorer les résultats rapportés dans la littérature. Nous avons adapté la méta heuristique du recuit simulé au problème de complétion, et nous avons observé que le comportement du problème de complétion change en fonction du nombre de points initialement présent dans le LHD partiel, faisant varier la mutation la plus adaptée au problème. Nous avons pris ce fait en compte et amélioré le recuit simulé en utilisant une méthode de bandit pour choisir la mutation la plus appropriée pendant le déroulement de l'algorithme, dépassant chaque mutation individuelle lorsque le nombre de points présents initialement est intermédiaire.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Hypercubes Latins maximin pour l'echantillonage de systèmes complexes

Mots clefs : Hypercube latins maximin, NP-complet, algorithme d'approximation, recuit simulé

Résumé :
Un hypercube latin (LHD) maximin est un ensemble de points contenus dans un hypercube tel que les points ne partagent de coordonnées sur aucune dimension et tel que la distance minimale entre deux points est maximale. Les LHDs maximin sont particulièrement utilisés pour la construction de métamodèles en raison de leurs bonnes propriétés pour l'échantillonnage. Comme la plus grande partie des travaux concernant les LHD se sont concentrés sur leur construction par des algorithmes heuristiques, nous avons décidé de produire une étude détaillée du problème, et en particulier de sa complexité et de son approximabilité en plus des algorithmes heuristiques permettant de le résoudre en pratique.
Nous avons généralisé le problème de construction d'un LHD maximin en définissant le problème de compléter un LHD entamé en respectant la contrainte maximin. Le sous-problème dans lequel le LHD partiel est vide correspond au problème de construction de LHD classique.
Nous avons étudié la complexité du problème de complétion et avons prouvé qu'il est NP-complet dans de nombreux cas. N'ayant pas déterminé la complexité du sous-problème, nous avons cherché des garanties de performances pour les algorithmes résolvant les deux problèmes.
D'un côté, nous avons prouvé que le problème de complétion n'est approximable pour aucune norme en dimensions $k \geq 3$. Nous avons également prouvé un résultat d'inapproximabilité plus faible pour la norme $\mathcal{L}_\infty$ en dimension $k = 2$. D'un autre côté, nous avons proposé un algorithme d'approximation pour le problème de construction, et avons calculé le rapport d'approximation grâce à deux bornes supérieures que nous avons établies.
En plus de l'aspect théorique de cette étude, nous avons travaillé sur les algorithmes heuristiques, et en particulier sur la méta-heuristique du recuit simulé. Nous avons proposé une nouvelle fonction d'évaluation pour le problème de construction et de nouvelles mutations pour les deux problèmes, permettant d'améliorer les résultats rapportés dans la littérature.

Title : Maximin Latin hypercubes for experimental design

Keywords : Latin Hypercube Design, NP-completeness, approximation algorithm, Simulated Annealing

Abstract : A maximin Latin Hypercube Design (LHD) is a set of point in a hypercube which do not share a coordinate on any dimension and such that the minimal distance between two points, is maximal. Maximin LHDs are widely used in metamodeling thanks to their good properties for sampling. As most work concerning LHDs focused on heuristic algorithms to produce them, we decided to make a detailed study of this problem, including its complexity, approximability, and the design of practical heuristic algorithms.
We generalized the maximin LHD construction problem by defining the problem of completing a partial LHD while respecting the maximin constraint. The subproblem where the partial LHD is initially empty corresponds to the classical LHD construction problem.
We studied the complexity of the completion problem and proved its NP-completeness for many cases. As we did not determine the complexity of the subproblem, we searched for performance guarantees of algorithms which may be designed for both problems.
On the one hand, we found that the completion problem is inapproximable for all norms in dimensions $k \geq 3$. We also gave a weaker inapproximation result for norm $\mathcal{L}_\infty$ in dimension $k = 2$. On the other hand, we designed an approximation algorithm for the construction problem which we proved using two new upper bounds we introduced.
Besides the theoretical aspect of this study, we worked on heuristic algorithms adapted for these problems, focusing on the Simulated Annealing metaheuristic. We proposed a new evaluation function for the construction problem and new mutations for both the construction and completion problems, improving the results found in the literature.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France