REPUBLIQUE ALGERIENNE DEMOGRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE BLIDA 1
FACULTÉ DES SCIENCES
DEPARTEMENT DE MATHÉMATIQUES

Mémoire de Master
Spécialité : Recherche Opérationnelle

Présenté par
Fodil Myriam
Ouchene Imane
Sur le thème :

Approches métaheuristiques pour l'optimisation d'un Hypercube Latin

Soutenu publiquement, devant le jury composé de :

Mr. BOUDJEMAA Redouane	Professeur	Univ. Blida1	Président
Mme. ARRACHE Saida	MAA	Univ. Blida1	Examinatrice
Mme. ADLI Feriel	MAA	Univ. Blida1	Promotrice

Table des matières

Table des figures

Remerciements

Avant tout propos, nous remercions « **Dieu** » le tout puissant qui nous a donné la sagesse et la santé pour faire ce modeste travail.

C'est avec un grand plaisir que nous exprimons notre gratitude et nos sincères remerciements à notre promotrice M^{me} **ADLI Feriel** pour son orientation et son encadrement dans l'élaboration de ce mémoire de fin d'étude.

Nous adressons nos remerciements aussi M^{r} **BOUDJEMAA Redouane** Professeur à l'université Saad Dahlab Blida 1 d'avoir accepter de présider ce jury de mémoire

Nous remercions également M^{me} **ARRACHE Saida** Maitre Assistant classe A à l'université Saad Dahlab Blida 1 d'avoir accepter de juger notre mémoire.

Toutes nos reconnaissances sont adressées à tous les enseignants qui nous ont suivi infatigablement durant tout notre cursus universitaire.

Nous tenons à exprimer tout au fond de nos cœurs les reconnaissances à nos familles qui nous ont offert toujours un appui sûr par leurs soutiens et leurs encouragements, et bien sur nos amis, nos camarades qui nous ont soutenus de près ou de loin.

Résumé

La puissance croissante des ordinateurs a permis aux techniques inventées pour la conception et l'analyse des simulations d'être appliquées à un large éventail de problèmes. En général, lorsque les simulations prennent du temps, un modèle de substitution est construit pour remplacer le code informatique. La toute première étape d'une modélisation réussie est la planification de la configuration d'entrée qui sera utilisée pour exercer le code de simulation. Parmi les stratégies inventées pour les expériences informatiques, les plans d'hypercubes latins sont devenus particulièrement populaires. Dans notre mémoire, on s'intéresse à l'optimisation d'un hypercube latin selon les deux critères d'optimalité les plus utilisés en littérature : le maximin distance, où il s'agit de maximiser la distance minimale entre chaque paire de points de la conception en utilisant un algorithme dit maximin, et le critère de l'énergie potentielle, basé sur l'analogie physique de la minimisation des forces entre particules chargées en utilisant un algorithme génétique.

Mots clés : Plans d'expériences, hypercube latin, maximin distance, algorithme génétique.

Abstract

The increasing power of computers has enabled techniques invented for the design and analysis of simulations to be applied to a wide range of problems. Typically, when simulations are time-consuming, a surrogate model is built to replace the computer code. The very first step in successful modeling is planning the input configuration that will be used to exercise the simulation code. Among the strategies invented for computer experiments, Latin hypercube designs have become particularly popular. In our dissertation, we focus on optimizing a Latin hypercube according to the two optimality criteria most widely used in the literature : the maximin distance criterion, where the aim is to maximize the minimum distance between each pair of points in the design using a so-called maximin algorithm, and the potential energy criterion, based on the physical analogy of minimizing forces between charged particles using a genetic algorithm.

Key words : Design of experiments, Latin hypercube, maximin distance, genetic algorithm.

Introduction

De nombreux phénomènes physiques sont représentés par des équations déterministes qui conduisent, lors de la phase de modélisation numérique, à l'obtention de codes de calcul. Ces codes sont souvent coûteux en temps de calcul et prennent en compte un grand nombre de variables d'entrée (paramètres physiques, paramètres physico-chimiques, etc.). Il est important de comprendre l'influence de ces variables d'entrée sur la réponse du code. Pour cela, il convient de réaliser des études qui consistent à identifier quelles sont les variables d'entrée qui contribuent le plus à la variabilité de la réponse du modèle ([3]). Malheureusement, ces études nécessitent souvent un grand nombre de simulations. Face à la complexité de certains codes et au grand nombre de variables d'entrée, il est alors nécessaire d'approximer le code et de le remplacer par un métamodèle que l'on construit à partir d'un jeu de simulations initiales du code ([8],[11]). La manière la plus courante de définir un méta-modèle est de le désigner comme un modèle du modèle. Ce métamodèle est évidemment souhaité le plus fidèle possible au code dans le domaine de variation des variables influentes.

De récents travaux ([1]) ont développé une méthodologie permettant de construire le métamodèle dans le cas d'un grand nombre de variables d'entrée (20 variables d'entrée par exemple). Ce métamodèle est construit à partir d'un nombre initial de simulations du code que l'on nomme base d'apprentissage. C'est ainsi que se pose le problème du choix de la localisation des points de la base d'apprentissage afin d'améliorer la qualité de prédiction du métamodèle. Ces points peuvent être générés aléatoirement ou fixés de manière déterministe dans l'espace de variation des entrées suivant différents plans d'expériences. Un plan couramment utilisé en simulation numérique est l'hypercube latin noté LH pour Latin Hypercube ([26]).

Parmi cette vaste famille de plans aléatoires, des LH particuliers existent et ont pour objectif d'occuper au mieux l'espace des variables d'entrée et d'améliorer ainsi la qualité de

prédiction du métamodèle. Parmi ceux-là, on retrouve les LH optimisés. Le choix du critère d'optimisation et la mise au point de l'algorithme d'optimisation associé constituent un élément clef pour les LH optimisés. L'optimisation de ce type de plan entraîne une implémentation robuste, qui améliore considérablement les propriétés de remplissage du LH. L'objectif de notre travail et de trouver le meilleur LH adéquat selon deux critères les plus utilisés dans la littérature, le critère maximin distance en utilisant un algorithme dit maximin distance et le critère énergie potentielle en utilisant l'algorithme génétique.

Notre mémoire est répartit comme suit :

• Le premier chapitre est consacré à rappeler la définition de l'optimisation combinatoire, les généralités du problème combinatoire et les différentes méthodes de résolution.

• Le deuxième chapitre est consacré aux généralités sur les algorithmes génétiques.

• Le troisième chapitre est consacré à définir le plan hypercube latin, la problématique et les critères de remplissage d'espace.

• Le quatrième chapitre est consacré à la présentation des résultats de l'optimisation de l'hypercube latin selon le critère maximin distance par un algorithme maximin et le critère énergie potentielle par l'algorithme génétique.

Chapitre 1

Optimisation combinatoire

1.1 Introduction :

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Elle consiste à comprendre un problème réel et pouvoir le transformer en un modèle mathématique à fin d'extraire ses propriétés structurelles et le résoudre. Bien que les problèmes d'optimisation combinatoire sont souvent faciles à définir, ils sont généralement difficiles à résoudre. Les différentes méthodes de résolution développées en recherche opérationnelle (RO) peuvent être regroupées en deux catégories principales : les méthodes exactes qui garantissent l'optimalité de la solution et les méthodes approchées qui perdent en complétude pour gagner en efficacité.
Résoudre un problème d'optimisation consiste à trouver une solution optimale, c'est-à-dire trouver une solution réalisable qui minimise ou maximise, selon le contexte, la fonction objectif.

1.2 Problème d'optimisation combinatoire (POC) :

L'optimisation combinatoire est un outil indispensable combinant diverses techniques des mathématiques discrètes afin de résoudre des problèmes de la vie réelle [5]. D'une manière simple, résoudre un problème d'optimisation combinatoire consiste à trouver l'optimum d'une fonction, parmi un nombre fini de choix.[4]. Il s'agit, en général, de maximiser (problème de maximisation) ou de minimiser (problème de minimisation) une fonction objectif sous certaines contraintes. Il consiste à trouver un minimum (resp maximum) s^* d'une application

f, le plus souvent à valeurs entières ou réelles sur l'ensemble fini S

$$\text{Trouver } s^* \in S$$
$$f(s^*) = Min_{s \in S} f(s)$$

resp :

$$\text{Trouver } s^* \in S$$
$$f(s^*) = Max_{s \in S} f(s)$$

Clairement, la notion de l'optimisation combinatoire est une branche de la programmation mathématique qui recouvre les méthodes qui servent à déterminer l'optimum parmi plusieurs solutions réalisables d'une fonction sous des contraintes données.

1.2.1 Domaines d'application :

Les problèmes d'optimisation combinatoire représentent un outil très puissant d'aide à la décision dans les institutions économiques, sociales, industrielles, ... et trouvent leur application dans pas mal de domaines entre autres :
- Services publics : hôpitaux, transport public, informatique
- Industrie : automobile, aviation, énergie, télécom, production
- Finances : gestion de portefeuille
- Militaire : gestion des ressources, logistique
- Économie : planification de projets, organisation d'activités, affectation de tâches, problèmes de transport.
- Informatique : conception de programmes, implémentation de systèmes informatiques.
- Sociologie : modélisation et étude de phénomènes sociaux.

1.2.2 Exemples de problèmes d'optimisation combinatoire :

Les problèmes d'optimisation combinatoire se caractérisent en général par le fait qu'ils peuvent être modélisés par des formalismes mathématiques qui représentent en quelques sortes des méta-problèmes dont la résolution implique la résolution de toute une classe de problèmes, parmi ces modèles on trouve :
- Problème du voyageur de commerce

- Problème du sac-à-dos
- Problème du plus court chemin
- Problème d'arbre couvrant minimal
- Problème de coloration de graphes

- Le problème du voyageur de commerce consiste, étant donné un ensemble de villes séparées par des distances données, à trouver le plus court chemin qui relie toutes les villes. C'est un problème NP-complet.

- Le problème du sac à dos Étant donné plusieurs objets possédant chacun un poids et une valeur, la question est de trouver quels objets faut-il mettre dans un sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac.

- Le problème du plus court chemin consiste à trouver dans un graphe donné un chemin d'un sommet à un autre de façon à ce que la somme des poids des arcs de ce chemin soit minimale.

- Le problème de l'arbre couvrant minimal est un des plus vieux problèmes en théorie des graphes. La problématique se pose comme suit : étant donné un graphe avec un nombre de sommets et un nombre d'arêtes ayant des poids de valeurs dans l'ensemble des entiers relatifs, l'arbre couvrant minimal consiste à trouver l'ensemble des arêtes permettant de rejoindre tous les sommets sans former de cycle, et ce, avec un coût minimal.

- Le problème de la coloration de graphe consiste à attribuer une couleur dans un graphe donné à chacun de ses sommets de manière à ce que deux sommets reliés par une arête soient de couleur différente.

Le but est de trouver une solution optimale dans un temps d'exécution raisonnable. Cependant, la réalisation de cet objectif est loin d'être accomplie en raison de plusieurs problèmes qui deviennent de plus en plus complexes.

Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement pénibles à résoudre [28]. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent pas encore de solutions algorithmiques efficaces et acceptables pour toutes les données [10]. Outre la classification selon le nombre de

critère à optimiser, les méthodes de l'optimisation combinatoire peuvent être classées aussi en méthodes exactes et méthodes approchées. (voir section 1.5)

1.3 Complexité et classes de complexité :

Résoudre automatiquement un problème d'optimisation combinatoire nécessite sa définition, sa modélisation et sa formulation mathématique puis le choix d'une méthode appropriée qui peut se traduire en un algorithme informatique. L'exécution de cet algorithme exige la réquisition de deux ressources majeures de l'ordinateur, en l'occurrence : l'espace mémoire nécessaire pour stocker les données du problème et le temps processeur nécessaire pour effectuer les opérations de la méthode en question.

1.3.1 Complexité :

L'expression "complexité d'un problème" fait référence à une estimation du nombre d'opérations élémentaires nécessaires pour résoudre les différentes instances de ce problème. Cette estimation est donnée en termes d'ordre de grandeur par rapport à la taille de chaque instance. Il s'agit là d'une estimation dans le pire des cas dans le sens où la complexité d'un problème est définie en considérant son instance la plus délicate. Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution [29].

1.3.2 Classes de complexité :

Les classes de complexité ont été introduites pour **les problèmes de décision**. Un problème de décision est un problème posant une question dont la réponse est «oui» ou «non». Pour ces problèmes, on définit notamment les deux classes P et NP :

Classe P (Polynomial time) : qui contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale (majorée par un polynôme). Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre « efficacement ».

Classe NP (Nondeterministic Polynomial time) : elle contient l'ensemble des problèmes polynomiaux non déterministes, i.e., pouvant être résolus par un algorithme de complexité polynomiale pour une machine non déterministe (que l'on peut voir comme une machine capable d'exécuter en parallèle un nombre fini d'alternatives). Intuitivement, cela signifie que la résolution des problèmes NP peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas, mais l'examen de chaque cas doit pouvoir être fait en temps polynomial.

Problèmes NP-complets : Un problème de décision A est NP-complet s'il satisfait les deux conditions suivantes : $A \in NP$, et tout problème NP se réduit à A en temps polynomial. L'une des questions ouvertes les plus fondamentales en informatique théorique est vraisemblablement la question si "$P = NP$?" . Ceci revient à trouver un algorithme polynomial pouvant résoudre un problème NP-complet. Trouver un tel algorithme, pour un seul problème appartenant à la classe NP-complet, signifierait que tous les problèmes de cette classe pourraient être résolus en temps polynomial et en conséquence, que $P = NP$. Cependant, il est commun de penser que $P \neq NP$, mais aucune preuve n'a encore été trouvée jusqu'à aujourd'hui.
Il est important de préciser que tous les problèmes d'optimisation ne peuvent pas être classés comme des problèmes NP-complets, puisqu'ils ne sont pas tous des problèmes de décision, même si pour chaque problème d'optimisation on peut définir un problème de décision qui a une complexité équivalente. Voir la figure (1.1)

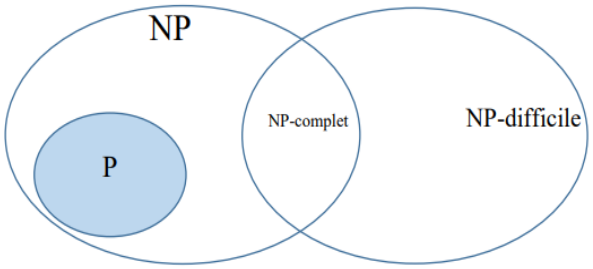

FIGURE 1.1 – Les classes de complexité

Problèmes NP-difficiles : Un problème A quelconque (de décision ou non) est

NP-difficile si et seulement si il existe un problème NP-complet A' qui est réductible à lui polynomialement. La définition d'un problème NP-difficile est donc moins étroite que celle de la NP-complétude [28]. De cette définition on peut observer que pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé à lui est NP-complet. De cette façon un grand nombre de problèmes d'optimisation ont été prouvés NP-difficiles. C'est notamment le cas des problèmes du Voyageur de Commerce, de Partitionnement de Graphes.

1.4 Modélisation :

Le verbe « modéliser » signifie « Procéder à une modélisation. » c'est-à-dire « Établissement de modèles, notamment des modèles utilisés en automatique, en informatique, en recherche opérationnelle et en économie. » Les mathématiques consistent d'abord en un langage, qui permet de transcrire des problèmes de nature quantitative : c'est la modélisation. Une fois cette transcription faite, des outils sont disponibles pour résoudre ces problèmes, partiellement ou complètement. On ramène ensuite la solution dans son contexte d'origine.
En pratique, il est rare que le problème proposé se traduise par l'optimisation d'une certaine fonction, parce qu'on ne peut optimiser qu'un seul critère à la fois, alors que les entreprises veulent agir sur plusieurs : augmenter la production, réduire les coûts, etc. L'étape de modélisation permet la recherche de "solutions acceptables", c'est à dire pour lesquelles un certain nombre de contraintes seront respectées.

Réaliser une modélisation signifie avant tout chercher à comprendre ce qui se passe, ne pas se contenter d'une solution empirique.
Modéliser un processus, c'est le décrire de manière scientifique, quantitative, par exemple en termes d'équations (physiques, chimiques, etc). Cela permet d'en étudier l'évolution, d'en simuler des variantes, en modifiant certains paramètres.

Modéliser un problème quelconque c'est l'écrire sous un aspect mathématique, les étapes de cette modélisation sont les suivantes :

1. • Identification des variables de décision.
2. • La description de la méthode qui permet d'évoluer l'état concerné, étant donné une configuration des variables de décision.

3. • La description mathématique des circonstances, ou contraintes précisant les valeurs que les variables de décision peuvent prendre.
4. • La fonction objectif.

1.5 Méthodes de résolution :

Résoudre un POC nécessite l'étude des trois points suivants :

• La définition de l'ensemble des solutions réalisables,
• L'expression de l'objectif à optimiser,
• Le choix de la méthode d'optimisation (exacte ou approchée) à utiliser.

Historiquement, c'est la recherche opérationnelle qui a commencé à élaborer des modèles d'optimisation programmables. Depuis, d'autres techniques, développées dans le cadre de la résolution de problèmes en intelligence artificielle, sont venues s'y joindre. Le choix d'une méthode de résolution répond toujours à une certaine problématique que l'on peut tenter de caractériser en quatre questions-clés :

• L'existence d'un modèle d'optimisation,
• L'exactitude des solutions,
• Le mode de résolution,
• Le coût de la méthode.

Ces questions ne sont pas totalement indépendantes.
Dans une démarche basée sur un modèle d'optimisation, l'ensemble des connaissances permettant de fournir une solution idéale est intégrée dans un modèle programmable dont l'exécution ne demande aucune intervention - ou presque - d'un décideur. Mais certaines connaissances sont difficiles à modéliser, à représenter ou à exploiter (surtout dans un milieu aléatoire où les données sont incertaines, imprécises...), il est impératif, dans ce genre de cas, de faire appel à des approches basées sur l'aide à la décision qui sont plus réalistes et plus souples.
Le coût, dans le choix d'une méthode de résolution d'un problème d'optimisation combinatoire, est évidemment très important. Malgré que la puissance des ordinateurs croît de jour en jour et la limite de la taille des problèmes combinatoires recule également mais l'utilisation de certaines méthodes dites exactes reste toujours d'un coût très élevé. Dans le cas d'une résolution interactive, le nombre de décisions prises par le décideur, et/ou la remise en question de ces décisions est également un facteur qui conditionne ce coût sans parler de

celui causé par la méthode de résolution elle-même.

Comme il est indiqué précédemment, les problèmes appartenant à la classe P ont des algorithmes efficaces et sont de complexité polynomiale tel que la méthode du chemin critique, la méthode PERT dans l'ordonnancement de projets... Pour les autres problèmes, il est peu réaliste de trouver des algorithmes de ce type. Alors, on fait appel à une famille de méthodes qui sont exactes ou approchées.(figure 1.2)

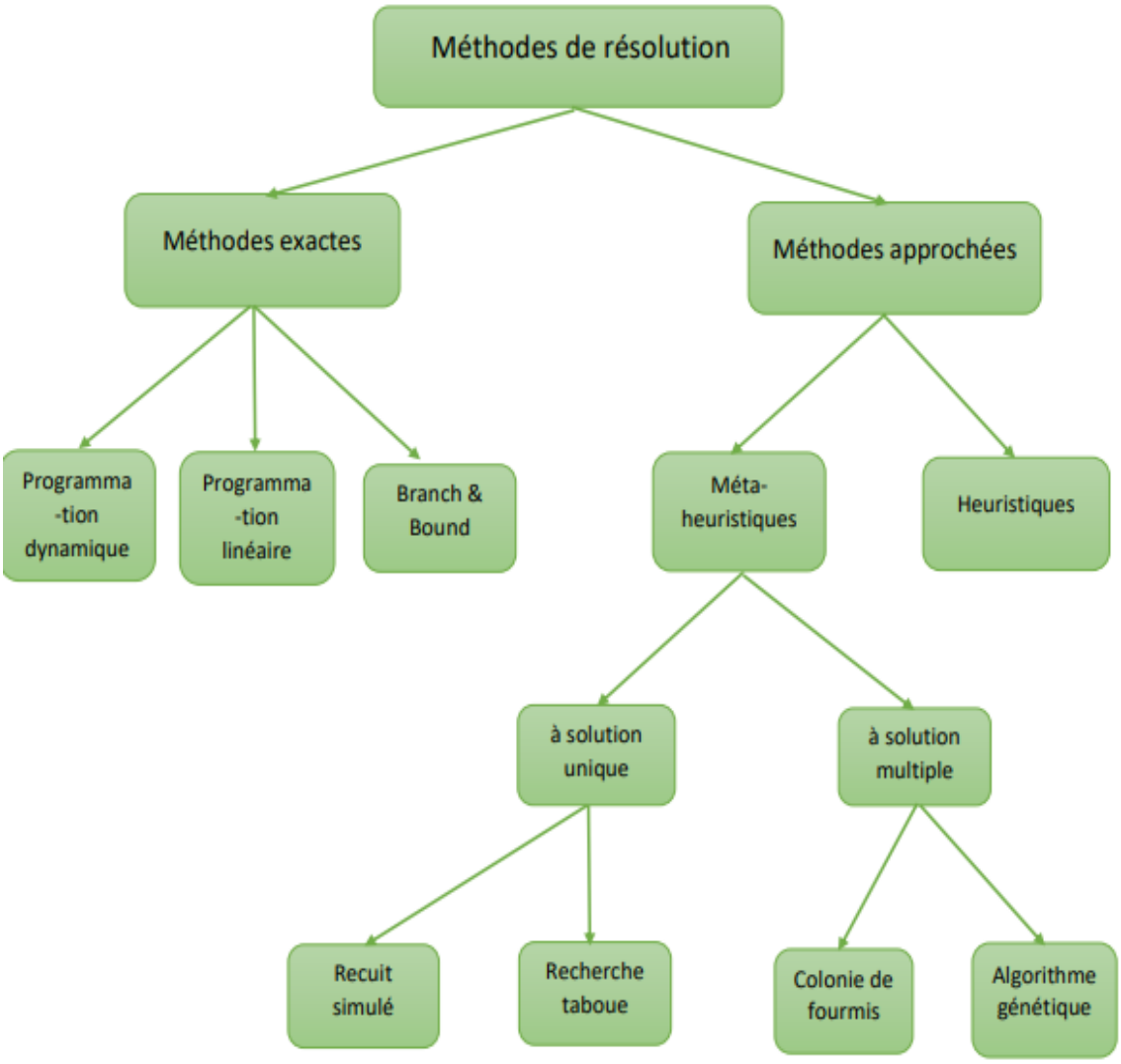

FIGURE 1.2 – Méthodes de résolution des problèmes d'optimisation combinatoire

1.5.1 Méthodes exactes :

Le principe des méthodes exactes consiste à rechercher, souvent de manière implicite, une solution, la meilleure solution ou l'ensemble des solutions d'un problème. L'optimisation exacte concerne toutes les méthodes permettant d'obtenir un résultat dont on sait qu'il est optimal à un problème précis. Cela va des méthodes du simplex aux méthodes du Lagrangien en passant par la programmation dynamique. On peut classer les méthodes exactes en trois grandes classes :

- La programmation dynamique,
- La programmation linéaire continue ou en nombres entiers,
- Les méthodes de recherche arborescente (Branch and Bound)

a-La programmation dynamique : La programmation dynamique consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires. Une fois tous les sous-problèmes résolus, la solution optimale pour le problème global est construite.

b-La programmation linéaire : En programmation linéaire, les problèmes sont modélisés à l'aide de variables de décision, d'une fonction objectif linéaire et de contraintes linéaires. Les variables de décision représentent les quantités à déterminer, tandis que la fonction objectif exprime l'objectif à atteindre (maximisation ou minimisation) en fonction des variables. Les contraintes linéaires décrivent les limites et les conditions auxquelles les variables doivent satisfaire.
Une fois le problème modélisé, des algorithmes spécifiques tels que la méthode du simplexe ou des méthodes basées sur la programmation linéaire en nombres entiers (PLNE) peuvent être utilisés pour trouver la meilleure solution possible. Le simplexe est l'algorithme le plus couramment utilisé pour résoudre des problèmes de programmation linéaire.

c-La méthode Branch and Bound : Branch and Bound est une méthode de résolution de classe de problèmes d'optimisation globale utilisant la stratégie de diviser un ensemble donné en plusieurs sous ensembles de plus en plus petits et ceci en se basant sur deux concepts, le branchement qui consiste à partitionner ou diviser l'espace des solutions en sous problèmes, pour les optimiser chacun individuellement, et l'évaluation qui consiste à déterminer l'optimum global. Le temps nécessaire pour trouver la solution optimale d'un

problème d'optimisation combinatoire augmente généralement avec la taille du problème. De façon pratique, seuls les problèmes de petite ou moyenne taille peuvent être résolus de façon optimale par des algorithmes exacts.

1.5.2 Méthodes approchées :

Contrairement aux méthodes exactes, les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution (de qualité raisonnable) en un temps de calcul aussi faible que possible. L'avantage principale de ces méthodes est qu'elles peuvent s'appliquer à n'importe quel problème facile ou difficile.

a-Heuristiques :

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial, au moins, une solution réalisable rapide, pas nécessairement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement, une heuristique est conçue pour un problème particulier en s'appuyant sur sa structure propre sans offrir aucune garantie quant à la qualité de la solution calculée.

b-Métaheuristiques :

Face aux difficultés rencontrées par les heuristiques pour avoir une solution réalisable de bonne qualité pour des problèmes d'optimisation difficiles, les méta-heuristiques ont fait leur apparition. Ces algorithmes permettent généralement d'obtenir une solution de très bonne qualité pour des problèmes issus des domaines de la recherche opérationnelle dont on ne connait pas de méthodes efficaces pour les traiter ou bien quand la résolution du problème nécessite un temps élevé ou une grande mémoire de stockage.
Une méta-heuristique peut être adaptée pour différents types de problèmes, tandis qu'une heuristique est utilisée à un problème donné (i.e. : une méta-heuristique combine plusieurs approches heuristiques).
On peut partager les métaheuristiques en deux grands types : les métaheuristiques **à solution unique** (i.e. ; évoluant avec une seule solution) comme le recuit simulé, la recherche taboue..., et celles **à solution multiple** ou population de solutions comme les colonies de fourmis et les algorithmes génétiques... Les méthodes d'optimisation à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions.

L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité.

Le recuit simulé : la méthode s'inspire du processus physique. Ce processus utilise une métallurgie pour améliorer la qualité d'un solide et cherche un état d'énergie minimale qui correspond à une structure stable ou solide. En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve son équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

La recherche taboue : très utilisée pour de nombreux problèmes, est encore une amélioration de la descente locale. Elle converge vite vers une bonne solution. Le principe est similaire à partir d'une solution, appelée solution courante, l'algorithme explore d'autres solutions en appliquant un opérateur de voisinage. La sélection d'une solution à partir de la solution courante doit être la meilleure parmi son voisinage. À chaque choix d'une solution parmi un voisinage, la solution choisie est stockée dans une liste taboue. Cette liste contient donc un certain nombre de solutions choisies précédemment. Le temps que la solution reste dans la liste dépend de la longueur maximale de cette liste que l'on aura fixée au préalable. À chaque sélection, la nouvelle solution choisie ne doit pas appartenir à cette liste taboue. Contrairement à une simple descente locale, la solution sélectionnée peut être de moins bonne qualité que la solution courante, ce qui permet d'éviter d'être rapidement bloqué sur une solution et de continuer à en explorer d'autres. Après un certain nombre d'itérations, la meilleure solution trouvée, parmi toutes celles qui ont été explorées, est retournée par l'algorithme. Le nombre d'itérations maximal doit lui aussi être défini préalablement.

Algorithme de colonies de fourmis : Initialement proposé par [24]. dans les années 1990 1,2, pour la recherche de chemins optimaux dans un graphe, le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture. L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes et plusieurs algorithmes ont vu le jour, s'inspirant de divers aspects du comportement des fourmis.

Algorithme génétique : Les algorithmes génétiques (AG) sont inspirés de la génétique classique. De manière générale, un algorithme génétique est constitué d'une population P de solutions appelées individus, dont l'adaptation à leur entourage est mesurée grâce à une fonction d'aptitude qui retourne une valeur réelle, appelée fitness qui est l'équivalent de la fonction objectif dans la recherche opérationnelle. Le principe général d'un AG consiste à simuler l'évolution d'une population d'individus en utilisant des opérateurs évolutionnaires jusqu'à la satisfaction d'un critère d'arrêt.

Chapitre 2

Algorithmes génétiques

2.1 Introduction

Trouver la solution optimale d'un problème dans un espace complexe implique un compromis entre deux objectifs l'exploitation des meilleures solutions et l'exploration robuste de l'espace de recherche. Les méthodes de type grimpeur procèdent itérativement en tentant, à chaque pas, de trouver localement une solution intermédiaire meilleure que la solution courante ; ce genre de méthode est pénalisée par son incapacité à traiter des problèmes représentant des reliefs de solutions multimodales (système possédant plusieurs optimaux locaux). Un algorithme génétique (AG) est une technique d'optimisation stochastique fondée sur les mécanismes de la sélection naturelle et de la génétique et elle est employée en informatique pour chercher les solutions approximatives à des problèmes d'optimisation. Les algorithmes génétiques sont une classe particulière des algorithmes évolutionnaires qui emploient des techniques inspirées de la biologie évolutionnaire telle que la mutation, la sélection, et la recombinaison (ou le croisement).
Les algorithmes génétiques (AG) représentent une stratégie de recherche réalisant un compromis équilibré entre l'exploration de l'espace de recherche et l'exploitation des meilleures solutions. Des analyses théoriques ont montré que les algorithmes génétiques gèrent ce compromis de façon optimale.

2.2 Définition :

L'optimisation par algorithme génétique prend son origine dans les mécanismes de la sélection naturelle et la génétique de l'évolution. Cette méthode a été mise en euvre par [24] dans les années 70. Comme son nom l'indique, elle est basée sur la traduction mathématique des phénomènes naturels qui sont la reproduction des espèces, la survie et l'adaptation des individus. Cette traduction est exploitée pour la résolution de problèmes nécessitant l'optimisation d'une fonction ou d'un système dépendant de plusieurs paramètres et qui ont besoin d'être calculés pour un critère bien défini (maximisation, minimisation, ..).
Cette technique constitue une méthode d'optimisation robuste. L'AG peut résoudre, avec fiabilité, des fonctions représentant des reliefs de solution réputés très difficiles pour les méthodes d'optimisation classiques (Simplex, le plus fort gradient ...). Les fonctions réputées difficiles sont des fonctions qui présentent plusieurs optima lOcaux, des fonctions discontinues ou des fonctions à plusieurs dimensions Où les méthodes ordinaires ne peuvent pas prendre en compte l'effet d'interaction entre tous les paramètres.

2.3 Analogie avec la biologie :

Les algorithmes génétiques étant basés sur des phénomènes biologiques, il convient de rappeler au préalable quelques termes génétiques.
Les organismes vivants sont constitués de cellules, dont les noyaux comportent des chromosomes qui sont des chaînes d'ADN.
L'élément de base de ces chromosomes est un gène. Sur chacun de ces chromosomes, une suite de gènes constitue une chaîne qui code la fonctionnalité de l'organisme (la couleur des yeux, la taille,....) ; la position d'un gène sur le chromosome est son locus ; l'ensemble des gènes d'un individu est son génotype ; et l'aspect que lui donne le patrimoine génétique est son phénotype.

Avant d'expliquer en détails le fonctionnement d'un algorithme génétique, il convient de présenter quelques mots du vocabulaire relatif à la génétique. Ces mots sont souvent utilisés pour décrire un algorithme génétique [16] .

- **Gène :** Un gène est une suite de bases azotées (adénine (A), cytosine (C), guanine (G) et la thymine (T)) qui contient le code d'une protéine donnée. On appellera gène la suite de symboles qui codent la valeur d'une variable. Dans le cas général, un gène correspond à un

seul symbole (0 ou 1 dans le cas binaire).

• **Chromosome :** Un chromosome est constitué d'une séquence finie de gènes qui peuvent prendre des valeurs appelées allèles qui sont prises dans un alphabet qui doit être judicieusement choisi pour convenir au problème étudié.

• **Individu :** On appelle individu une des solutions potentielles. Dans la plupart des cas, un individu sera représenté par un seul chromosome. Dans ce cas, par abus de langage, on utilisera indifféremment individu et chromosome.

• **Population :** On appellera population l'ensemble des solutions potentielles qu'un AG utilise.

• **Génération :** On appelle génération l'ensemble des opérations qui permettent de passer d'une population Pi à une population Pj. Ces opérations sont généralement :
- la sélection des individus de la population courante,
- l'application des opérateurs génétiques,
- l'évaluation des individus de la nouvelle population.

• **Fitness :** La fonction fitness est la pièce maitresse dans le processus d'optimisation, c'est l'élément qui permet aux algorithmes génétiques de prendre en compte un problème donné.Pour que le processus d'optimisation puisse donner de bons résultats, il faut concevoir une fonction fitness permettant une évaluation pertinente des solutions d'un problème sous forme chiffrée. Cette fonction est déterminée en fonction du problème à résoudre et du codage choisi pour les chromosomes. Pour chaque chromosome, elle attribue une valeur numérique, qui est supposée proportionnelle à la qualité de l'individu en tant que solution. Le résultat renvoyé par la fonction d'évaluation va permettre de sélectionner ou de refuser un individu selon une stratégie de sélection [36].

Comme dans les systèmes biologiques soumis à des contraintes, les meilleurs individus de la population sont ceux qui ont une meilleure chance de se reproduire et de transmettre une partie de leur héritage génétique à la prochaine génération. Une nouvelle population, ou génération, est alors créée en combinant les gènes des parents. On s'attend à ce que certains individus de la nouvelle génération possèdent les meilleures caractéristiques de leurs deux pa-

rents, et donc qu'ils seront meilleurs et seront une meilleure solution au problème. Le nouveau groupe (la nouvelle génération) est alors soumis aux même critères de sélection, et par après génère ses propres rejetons. Ce processus est répété plusieurs fois, jusqu'à ce que tous les individus possèdent le même héritage génétique. Les membres de cette dernière génération, qui sont habituellement très différents de leurs ancêtres, possèdent de l'information génétique qui correspond à la meilleure solution au problème.

L'algorithme génétique de base comporte trois opérations simples qui ne sont pas plus compliquées que des opérations algébriques :

– Sélection

– Reproduction

– Mutation

2.4 Principes généraux des algorithmes génétiques :

Les opérations successives utilisées dans les algorithmes génétiques sont illustrées par la figure ci-dessous (figure 2.1) [15]

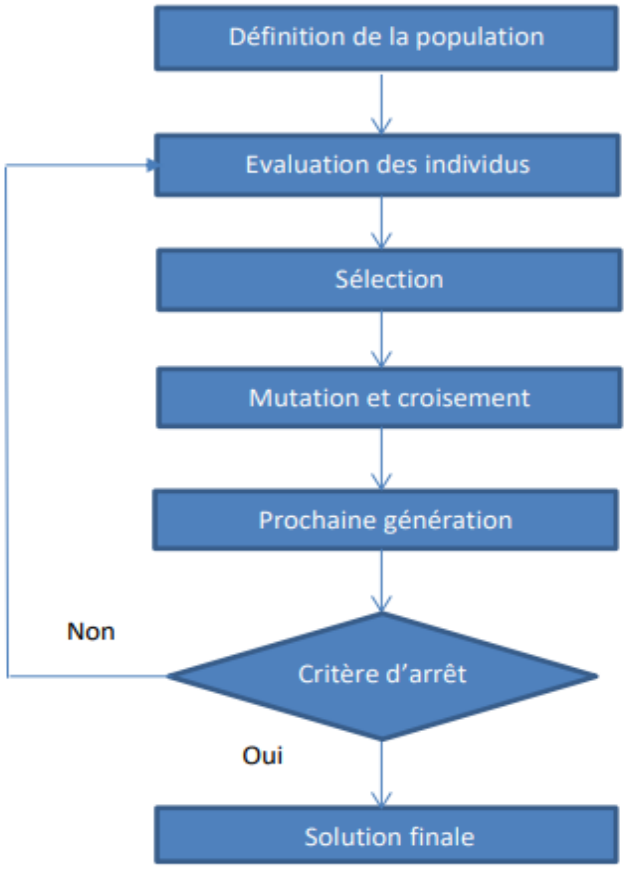

FIGURE 2.1 – Opérations successives utilisées dans les algorithmes génétiques.

Un algorithme génétique est constitué des étapes suivantes :

1. Tout d'abord une population d'individus est générée de façon aléatoire.

2. Une évaluation de l'ensemble des individus de la population initiale est procèdée, cette évaluation est utilisée comme un critère de sélection lors de l'étape de la sélection.

3. Un certain nombre d'individus est sélectionné dans la population, afin de produire une population intermédiaire qui va générer la nouvelle génération.

4. Un chromosome sélectionné aléatoirement afin d'appliquer l'opérateur de mutation.

5. Deux chromosomes parents $P1$ et $P2$ sélectionnés en fonction de leurs adaptations afin d'appliquer aléatoirement l'opérateur de croisement. Pour générer deux chromosomes enfants $E1$ et $E2$. On réitère les opérations de sélection, de mutation et de croisement afin de compléter la nouvelle population, ceci termine le processus d'élaboration d'une génération.

6. Réitérer les opérations précédentes à partir de la seconde étape jusqu'à la satisfaction du critère d'arrêt.

Codage :

C'est une modélisation d'une solution d'un problème quelconque en un chromosome. Le choix du codage est important, souvent il est délicat. En effet, le choix du type de codage ne peut pas être effectué de manière évidente. Le choix le plus adéquat consiste à choisir un codage qui semble le plus naturel en fonction du problème à traiter. L'objectif est d'abord de pouvoir coder n'importe quelle solution. Il est souhaitable, au-delà de cette exigence, d'imaginer soit un codage tel que toute chaîne de caractères représente bien une solution réalisable du problème, soit un codage qui facilite ensuite la conception du croisement de telle sorte que les « enfants » obtenus à partir de la recombinaison de leurs « parents » puissent être associés à des solutions réalisables, au moins pour un grand nombre d'entre eux [6],[9]. Parmi les codages les plus utilisés, on peut citer :

-Le codage binaire :

Ce type de codage est certainement le plus utilisé car il présente plusieurs avantages. Son principe est de coder la solution selon une chaîne de bits (les valeurs 0 ou 1). Le codage binaire a permis certes de résoudre beaucoup de problèmes, mais il s'est avéré obsolète pour certains problèmes d'optimisation numérique. En effet, il est plus pratique d'utiliser un codage réel des chromosomes.

-Le codage réel :

Une autre approche semblable est de coder les solutions en tant que suites de nombres entiers ou de nombres réels, où chaque position représente encore un certain aspect particulier de

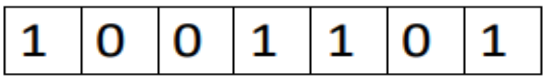

FIGURE 2.2 – Codage binaire d'un chromosome

la solution. Cette approche tient compte d'une plus grande précision et de complexité que le codage basé sur les nombres binaires seulement. Ce type de codage est intuitivement plus proche à l'environnement des problèmes à résoudre.

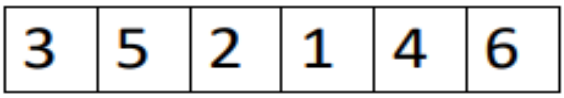

FIGURE 2.3 – Codage réel d'un chromosome

2.5 Fonctionnement des algorithmes génétiques :

Tout algorithme génétique passe par ces étapes : on commence par générer une population de façon aléatoire. Pour passer d'une génération à la suivante, les opérations suivantes sont répétées pour tous les éléments de la population. Des couples de parents sont sélectionnés en fonction de leur adaptation. L'opérateur de mutation est appliqué aux parents avec une probabilité p_m, L'opérateur de croisement leur est appliqué avec une probabilité p_c , et générer de couples d'enfants, et l'opérateur de remplacement est appliqué pour introduire les descendants. Ce cycle est répété jusqu'à ce qu'une solution satisfaisante soit trouvée.

2.5.1 Sélection :

La sélection permet d'identifier statistiquement les meilleurs individus de la population courante qui seront autorisés à se reproduire, cette opération est fondée sur la performance des individus, estimée à l'aide de la fonction fitness.
On trouve essentiellement quatre types de méthodes de sélection différentes :
• La méthode de la "loterie biaisée" (roulette wheel) de Goldberg,
• La méthode "élitiste",
• La sélection par tournois,

• La sélection universelle stochastique.

La méthode par roulette :

Est la plus célèbre des sélections stochastiques. Supposant un problème de maximisation avec uniquement des performances positives, elle consiste à donner à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance. Alors Un individu avec une forte évaluation a une grande chance d'être sélectionné alors qu'un individu avec un plus faible coût en a moins.

Sélection par élitisme :

Cette méthode consiste à sélectionner les N individus dont on a besoin pour la nouvelle génération P_i en prenant les n0 meilleurs individus de la population P après l'avoir triée de manière décroissante selon la fonction d'adaptation (fitness) de ses individus. Il est inutile de préciser que cette méthode est encore pire que celle de la loterie biaisée dans le sens où elle amènera à une convergence prématurée encore plus rapidement et surtout de manière encore plus sûre que la méthode de sélection de la loterie biaisée ; en effet, la pression de la sélection est trop forte, la variance nulle et la diversité inexistante.

Sélection par tournoi :

Cette méthode n'utilise que des comparaisons entre individus et ne nécessite pas le tri de la population. Elle possède un paramètre, taille du tournoi. Pour sélectionner un individu, on en tire t uniformément dans la population, et on sélectionne le meilleur de ces t individus.

Sélection par rang :

La sélection par rang trie d'abord la population par fitness. Chaque individu se voit associé un rang en fonction de sa position. La valeur du rang est calculée selon le total des adaptations des individus divisé par la taille de la population, autrement $rang = \sum f_i/n$, alors le plus mauvais individu aura le premier rang, le suivant le deuxième rang, ainsi de suite. La sélection par rang d'un individu est la même que par roulette, mais les proportions

sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. Avec cette méthode de sélection, tous les individus ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution.

2.5.2 Mutation :

C'est un processus où un changement mineur de code génétique est appliqué à un individu pour introduire de la diversité et ainsi d'éviter de tomber dans des optimums locaux. Différentes manières de mutation d'un chromosome sont aussi définies dans la littérature.

Mutation en codage binaire :

Dans un algorithme génétique simple, la mutation en codage binaire est la modification aléatoire occasionnelle (de faible probabilité) de la valeur d'un caractère de la chaîne.

Mutation en codage réel :

Pour le codage réel, les opérateurs de mutation les plus utilisés sont les suivants :

Mutation par inversion : Deux positions sont sélectionnées au hasard et tous les gènes situés entre ces positions sont inversés. La figure suivante montre un exemple de mutation par inversion.

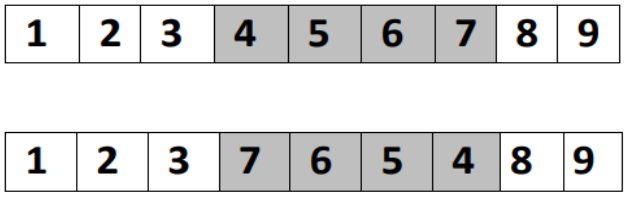

FIGURE 2.4 – Mutation par inversion

Mutation par insertion : Deux positions sont sélectionnées au hasard et le gène appartenant à l'une est inséré à l'autre position. Un exemple de mutation par insertion est illustré par la figure suivante :

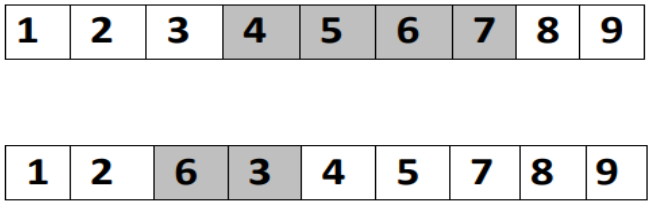

FIGURE 2.5 – Mutation par insertion

Mutation par déplacement : Une séquence est sélectionnée au hasard et déplacée vers une position elle même tirée au hasard. Un exemple de mutation par déplacement est illustré par la figure :

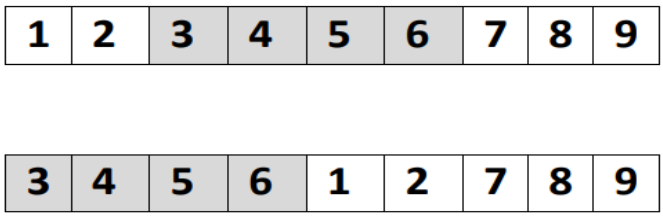

FIGURE 2.6 – Mutation par déplacement

Mutation par permutation : Deux positions sont sélectionnées au hasard et les gènes situés dans ces positions sont permutés. Comme il est illustré dans la figure :

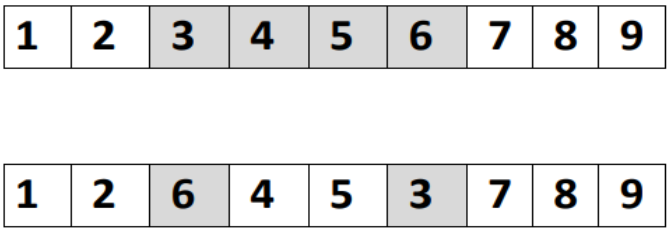

FIGURE 2.7 – Mutation par permutation

2.5.3 Croisement :

Cet opérateur permet la création de nouveaux individus selon un processus simple. Il permet donc l'échange d'information entre les chromosomes (individus). Lors de cette opération, 2 chromosomes s'échangent des parties de leurs chaînes pour donner de nouveaux chromosomes. Ces croisements peuvent être simples ou multiple. On distingue 2 types :

Le croisement binaire :

il correspond à un échange de gènes (bits) entre les parents.

Exemples :

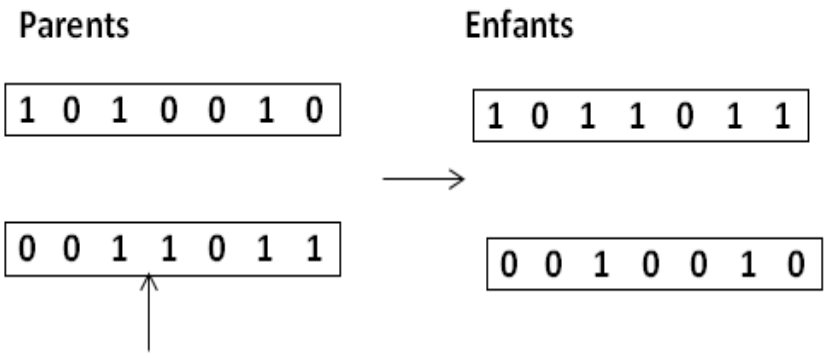

FIGURE 2.8 – Croisement avec un point

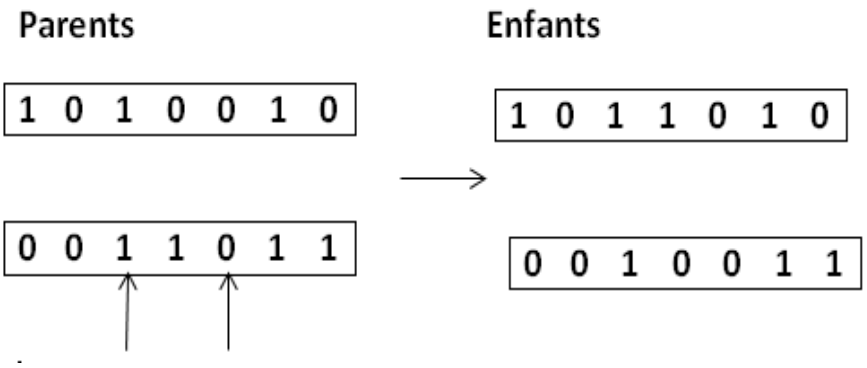

FIGURE 2.9 – Croisement avec deux points

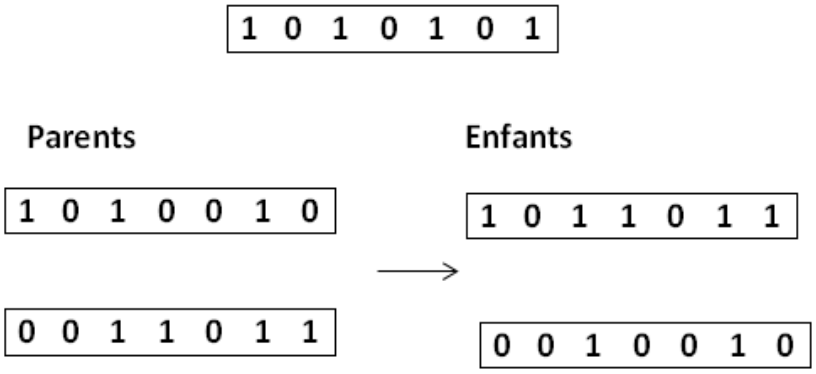

FIGURE 2.10 – Croisement uniforme

Le croisement réel :

Le croisement standard : Le croisement réel standard est très proche de celui décrit pour le codage binaire. Il ne se différencie du croisement binaire que par la nature des éléments qu'il altère. Bien évidement ce ne sont plus des bits qui sont échangés de part et d'autre du point de croisement mais des valeurs réelles.

Le croisement arithmétique (barycentrique) : Il consiste à choisir deux gènes $P_1(i)$ et $P_2(i)$ dans chacun des parents à la même position i, et à définir les gènes correspondants $E_1(i)$ et $E_2(i)$ chez les enfants par combinaison linéaire :

$$E_1(i) = \alpha P_1(i) + (1-\alpha)P_2(i)$$
$$E_2(i) = (1-\alpha)P_1(i) + \alpha P_2(i)$$

Où α est un coefficient de pondération aléatoire adapté au domaine de définition, il n'est pas nécessairement compris entre 0 et 1. Dans les applications il prendra des valeurs dans l'intervalle $[-0.5, 1.5]$.

2.5.4 Remplacement :

Cet opérateur est le plus simple, son travail est à introduire les descendants obtenus après application successive des opérateurs de sélection, de croisement et de mutation. Il y a 2 méthodes de remplacement :

Remplacement stationnaire : Les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives.

Remplacement élitiste : On garde au moins l'individu possédant les meilleurs performances d'une génération à la suivante.

2.6 Critères d'arrêt :

Le critère d'arrêt indique que la solution est suffisamment approchée de l'optimum. Plusieurs critères d'arrêt de l'algorithme génétique sont possibles. On peut arrêter l'algorithme après un nombre de générations suffisant pour que l'espace de recherche soit convenablement exploré, ce critère peut s'avérer coûteux en temps de calcul si le nombre d'individus à traiter dans chaque population est important. L'algorithme peut aussi être arrêté lorsque l'évolution de la population est ralentie. On peut aussi envisager d'arrêter l'algorithme lorsque la fonction d'adaptation d'un individu dépasse un seuil fixé au départ. Nous pouvons également faire des combinaisons des critères d'arrêt précédents.

2.7 Avantages et limites des algorithmes génétiques :

$*$ Un des grands avantages des algorithmes génétiques est qu'ils autorisent la prise en compte de plusieurs critères simultanément, et qu'ils parviennent à trouver de bonnes solutions sur des problèmes très complexes.

L'avantage principal des algorithmes génétiques par rapport aux autres techniques d'optimisation combinatoire consiste en une combinaison de l'exploration de l'espace de recherche, et de l'exploitation des meilleures solutions disponibles à un moment donné. Ils doivent simplement déterminer entre deux solutions quelle est la meilleure, afin d'opérer leurs sélections. Leur utilisation se développe dans des domaines aussi divers que l'économie, la bio informatique ou la vérification formelle.

∗ **Les inconvénients des algorithmes génétiques peuvent être résumés comme suit :**

- Coût d'exécution élevé : Les algorithmes génétiques nécessitent de nombreux calculs, en particulier lors de l'évaluation des individus. Cela peut être coûteux en termes de temps de calcul, en particulier pour des problèmes complexes avec de grandes populations.

- Difficulté d'ajustement des paramètres : Les algorithmes génétiques ont plusieurs paramètres à ajuster, tels que la taille de la population, le taux de mutation, le taux de croisement, etc. Trouver les valeurs optimales pour ces paramètres peut être délicat et nécessiter plusieurs essais.

- Sensibilité aux optima locaux : Les algorithmes génétiques peuvent converger rapidement vers un optimum local, ce qui signifie qu'ils peuvent trouver une solution qui est relativement bonne dans le voisinage de cette solution, mais pas nécessairement la meilleure solution globale. Cela peut être un problème si l'objectif est de trouver la meilleure solution possible.

- Caractère indéterministe : En raison de l'utilisation de facteurs aléatoires dans les opérateurs génétiques, un algorithme génétique peut se comporter différemment pour des paramètres et populations identiques. Cela signifie qu'il faut exécuter l'algorithme plusieurs fois et analyser statistiquement les résultats pour évaluer correctement sa performance.

Chapitre 3

Plans hypercubes latins

3.1 Introduction

La méthode des plans d'expériences, également connue sous le nom de conception expérimentale, est une méthode systématique utilisée en sciences et en ingénierie pour étudier les relations entre les variables d'un système et les résultats observés. Elle vise à optimiser les processus, à améliorer les produits et à résoudre les problèmes en identifiant les facteurs clés qui influencent un phénomène donné.
Les plans d'expériences sont souvent utilisés dans des domaines tels que l'ingénierie, la chimie, la biologie, la pharmacologie, l'agriculture et les sciences sociales. Ils peuvent être utilisés pour résoudre des problèmes variés, tels que l'optimisation de processus de fabrication, l'amélioration des performances des produits, l'identification des causes de défaillances, l'optimisation des paramètres de conception et la compréhension des relations entre les variables.

3.2 Plan d'expériences :

Les plans d'expériences impliquent la manipulation délibérée des variables d'entrée, appelées facteurs, afin de mesurer leur effet sur les variables de sortie, appelées réponses. Les facteurs peuvent être des paramètres physiques, chimiques, opérationnels ou d'autres variables contrôlables dans le système étudié.
La planification d'expériences est essentielle dans le domaine de la métamodélisation. Son objectif est de sélectionner des points dans l'espace des paramètres où la réponse d'intérêt

doit être évaluée. Ces points sélectionnés forment le plan d'expériences. Ensuite, la réponse d'intérêt est évaluée ou simulée pour chaque point, ce qui génère un ensemble d'observations. Le plan d'expériences et les observations sont les seules informations nécessaires pour construire un métamodèle. La qualité du métamodèle est fortement influencée par la définition du plan d'expériences. La littérature propose de nombreuses méthodes pour définir un plan d'expériences, en particulier dans le contexte de la construction d'un métamodèle.
L'objectif principal d'un plan d'expériences est de minimiser le nombre d'essais nécessaires pour obtenir des résultats significatifs. Il permet de prendre des décisions éclairées en fournissant des informations sur l'importance relative des différents facteurs et de leurs interactions.

3.3 Plan Hypercube Latin :

Les plans d'expériences classiques (factoriels, composite, Box-Benhken,...) ne sont pas appropriés lors de simulations numériques. En l'absence d'information sur le comportement de la réponse en fonction des entrées, des plans de type Hypercube Latin (LH pour *Latin Hypercube* en anglais) sont souvent privilégiés.
Contrairement aux plans d'expériences classiques, associés à des modèles polynomiaux du premier ou second degré, les plans développés pour LH sont des plans optimaux qui répondent à certains critères d'optimalité, tels que : le critère de remplissage de l'espace, le critère de la discrépance centrée , le critères d'entropie,..

Les plans d'expériences Hypercubes latins désignent des plans d'expériences exploratoires particulièrement adaptés à la métamodélisation [32]. Ce sont des plans pour lesquels le placement des points d'expériences est optimisé selon un critère géométrique ou statistique. L'optimisation est aussi dépendante du nombre d'expériences n et de la dimension p considérés. Les LH sont utilisés pour économiser le temps de traitement informatique lors de l'exécution de simulations. Des études ont montré qu'un LH bien exécuté peut réduire le temps de traitement jusqu'à 50 pour cent. Le LH est donc plus important lorsqu'on travaille avec des système d'exploitation et des logiciels lents

Cette technique permet ainsi : de s'assurer une bonne répartition spatiale des points du plan, d'avoir un plan optimal pour le modèle, de diminuer le temps de calcul de l'optimisation puisque le champ d'investigation est réduit à la classe du plan.

3.3.1 Histoire des hypercubes latins :

Les hypercubes latins s'inspirent du concept du "carré latin" des mathématiques combinatoires, où une matrice $n \times n$ est remplie de n objets différents (c'est-à-dire des nombres, des caractères, des mots, etc.), de telle sorte que chaque objet apparaît exactement une fois dans chaque ligne et exactement une fois dans chaque colonne. voir (figure 3.1) pour un exemple avec 4 objets.

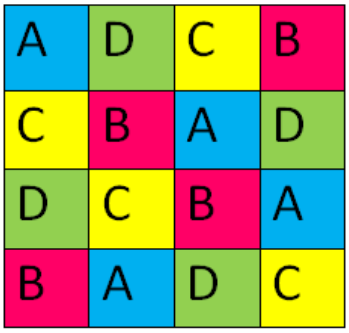

FIGURE 3.1 – Exemple de carré latin avec 4 caractères latins différents disposés de manière à ce que la lettre apparaît pas plus d'une fois dans chaque ligne ou colonne.

Le terme "latin" dans les carrés latins a été inspiré par les travaux du célèbre mathématicien Leonhard Euler, qui utilisait des caractères latins comme objets [37].
Comme pour les carrés latins, l'idée de base de LH pour un espace à deux dimensions et une taille n consiste à partitionner chaque dimension en n intervalles disjoints (niveaux) avec une probabilité marginale égale à $\frac{1}{n}$, puis partitionner au hasard une fois dans chaque intervalle pour s'assurer qu'il n'y a qu'un seul point à chaque niveau. (La figure 3.2) montre un exemple bidimensionnel avec 4 points uniformément répartis dans chaque dimension par un hypercube latin (LH).

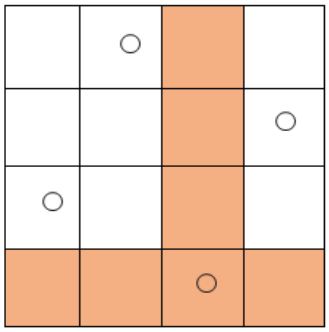

FIGURE 3.2 – Exemple bidimensionnel de LH avec 4 points. Il n'y a qu'un point dans chaque ligne et chaque colonne.

3.3.2 Améliorations fondées sur la notion d'optimisation :

Des améliorations méthodologiques telles que le LH orthogonal et le LH symétrique qui tentent de générer systématiquement des hypercubes latins de meilleure qualité, il y a eu un grand nombre d'études qui utilisent la théorie de l'optimisation pour améliorer la performance du LH [32],[38].

Fondamentalement, l'approche consiste à définir des critères secondaires (fonctions objectif), en plus d'être un hypercube latin, et à formuler et résoudre un problème d'optimisation pour atteindre des hypercubes latins optimaux.

Cela peut être très efficace, car dans toute variation de LH, il peut exister un grand nombre de configurations (arrangements de points) qui répondent aux critères de LH associés, mais qui se comportent mal en termes d'autres critères (par exemple, remplissage d'espace). (La figure 3.3) (a) illustre ce phénomène : les points du plan sont concentrés sur la diagonale du domaine. La répartition uniforme sur chaque axe (dimension) n'assure pas l'uniformité sur le domaine expérimental. Toutefois, pour une valeur fixée de n, il existe $(n!)^p$ hypercubes latins possibles.

Les algorithmes pour LH choisissent généralement au hasard l'une des nombreuses configurations possibles, tandis que l'optimisation permet de naviguer dans la myriade de choix et d'identifier celui qui est optimal en termes des critères optimaux.

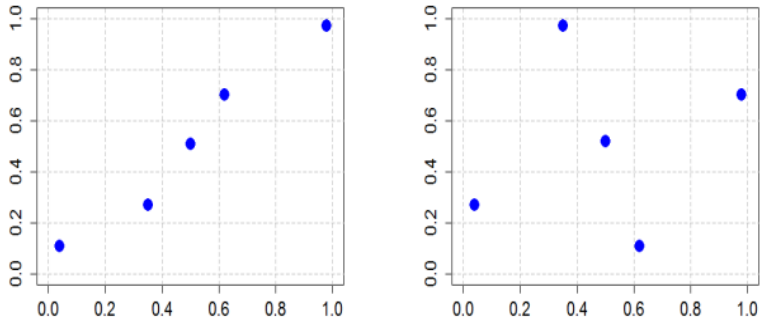

FIGURE 3.3 – (a)Un Hypercube Latin à 5 points en dimension 2.Le plan (a) optimisé.

3.3.3 Définition de l'hypercube latin :

Les hypercubes latins ont été le premier type de plan proposé spécifiquement pour les expériences numériques [26]. Un LH est une matrice de n lignes et p colonnes. (Figure3.4)

$$LH = \begin{pmatrix} x_{11} & x_{12} & ... & x_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ . & . & ... & . \\ . & . & ... & ... \\ x_{n1} & x_{n2} & ... & x_{np} \end{pmatrix}$$

telle que :

n : est le nombre de points pour chaque variable d'entrée.

p : est le nombre de dimensions.

avec $x_i = [x_{i1}, x_{i2}, ..., x_{in}]$ représente les niveaux. et les colonnes représentent les variables.

Chaque ligne représente un point de conception.

Chaque colonne représente une permutation des points.

Dans les hypercubes latins, les variables d'entrée sont présentées en n niveaux.

point	x1	x2	x3
1	1	1	2
2	2	4	3
3	4	2	4
4	3	3	1

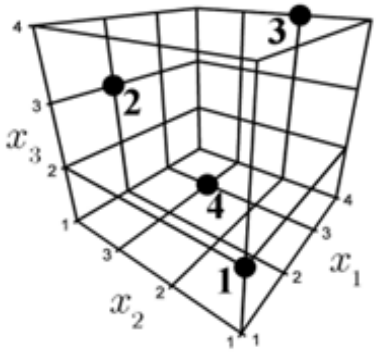

FIGURE 3.4 – Exemple de LH pour 4×3

Dans un LH, chaque valeur doit apparaître exactement une fois par ligne, une fois par colonne et une fois par plan. Cela signifie qu'aucune valeur ne peut se répéter dans une même ligne, colonne ou plan.
L'objectif dans un LH est de remplir la grille avec les valeurs de manière à respecter les contraintes de remplissage. Chaque combinaison unique de valeurs qui satisfait ces contraintes constitue une solution valide.
En pratique, un LH sélectionne n points parmi les n^p points de la grille de façon à ce que les n niveaux des variables d'entrée soient testés une fois par les simulations.

La complexité de la recherche d'un hypercube latin maximin optimal dépend de la fonction de distance et de la dimension. Pour la dimension 2 et les deux fonctions de distance l_1 et l_∞, un algorithme polynomial donnant l'hypercube latin maximin a été montré dans [35]. Pour d'autres dimensions et d'autres fonctions de distance, la complexité n'est pas connue mais on pense qu'elle est NP-hard [33].Dans [23], l'auteur affirme que l'optimisation d'un hypercube latin est un problème NP-complet. Elle peut être résolue à l'aide de techniques d'optimisation combinatoire telles que la recherche exhaustive, la recherche locale ou les algorithmes génétiques. Ces techniques peuvent être utilisées pour explorer l'espace des solutions possibles et trouver la disposition qui minimise ou maximise la fonction objectif tout en satisfaisant les contraintes.

3.4 Problème de conception d'hypercubes latins :

Les LH sont d'un grand intérêt dans le domaine de la métamodélisation et en tant qu'objet combinatoire. Les auteurs introduisent des plans LH optimisés selon un des critères de remplissage de l'espace comme [30] qui optimise les hypercubes latins selon l'entropie (équation (3.12)) en utilisant un algorithme d'échange, [27] selon le critère ϕ_p(équation (3.10)) en utilisant un algorithme de recuit simulé, [12] selon la discrépance centrée L2 (équation (3.13)) en utilisant un algorithme de seuil, [19] selon les critères d'entropie, de ϕ_p et de discrépance centrée L2 en utilisant l'algorithme ESE (Enhanced Stochastic Evolutionary) , [14] selon le critère de l'arbre couvrant de poids minimum (équation (3.11) en utilisant l'algorithme de Prim et enfin, en utilisant un algorithme génétique, [7] selon l'énergie potentielle (équation (3.9)).
Pour évaluer la bonne répartition spatiale d'une distribution de points, des critères numériques sont utilisés. Il sont répartis en deux grandes classes :
• les critères basés sur les distances entre les points du plan.
• les critères d'uniformité, qui quantifient l'écart entre la distribution empirique des points et la distribution uniforme sur le domaine.

3.4.1 Critères de distance

Critère maximin :

Un plan est appelé plan de distance maximin [20] s'il maximise la distance minimale entre les points du plan.
Pour obtenir une répartition équilibrée des points de conception, il est possible d'introduire une distance D entre les points, connue sous le nom de distance euclidienne. Ensuite, il s'agit de rechercher un LH dont la distance minimale est maximisée. Ces LH sont appelées LH maximin et sont fréquemment utilisés dans diverses applications.
Soit :

$$D(LH) = d(x_i, x_j) \tag{3.1}$$

telle que D la distance minimale entre x_i et xj dans LH (par rapport à une certaine distance d).

$$d(x_i, x_j) = min_{i \neq j} \sqrt{\sum_{l=1}^{p} |x_{il} - x_{jl}|^2} \tag{3.2}$$

Un LH maximin est :

$$D = max\{min_{i \neq j} d(x_i, x_j)\} \tag{3.3}$$

Étant donné que différents LH peuvent atteindre la valeur optimale de D, nous pouvons les différencier en minimisant la valeur de D.

$$J(LH) = |(i,j) : d(x_i, x_j)| = |D(LH)| \tag{3.4}$$

J est le nombre d'occurrences de la distance minimale D dans LH, dans le cas où différents LH partagent les mêmes valeurs D et J, nous pouvons faire une discrimination supplémentaire en maximisant la deuxième plus petite distance dans LH, et si l'égalité subsiste en minimisant J', le nombre d'occurrences de D', et ainsi de suite.
Ce type de problème peut être assimilé à un problème de placement de magasins franchisés où le client le plus éloigné du magasin le plus proche est aussi proche que possible. Du point de vue de la conception, l'idée est que le plans LH sera bien prédit lorsque le client est près des sites de conception et moins lorsque le client est éloigné de tous les sites observés.

Critère énergie potentielle :

Le critère AE a été développé par [8]. Les auteurs ont affirmé que le critère peut être compris comme exprimant l'énergie potentielle d'un système de particules avec des forces répulsives entre chaque paire de particules, la minimisation de cette énergie potentielle optimise la disposition spatiale des points. Les forces répulsives entre paires de points sont fonction de leur distance. La distance euclidienne, D_{ij}, entre les points :

$$D_{ij} = d(x_i - x_j) = \sqrt{\sum_{l=1}^{p} |x_{il} - x_{jl}|^2} = \sqrt{\sum_{l=1}^{p} (\Delta_{ij,l})^2} \tag{3.5}$$

où $\Delta_{ij,l}$ est la distance entre deux points, p est la dimension de LH.
Le critère d'Audze Eglajs est défini en utilisant le carré des distances euclidiennes entre toutes les paires de points expérimentaux comme

$$E^{AE} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{D_{ij}^2} \tag{3.6}$$

Plusieurs auteurs affirment que les interactions des forces imitent les forces gravitationnelles. Par exemple, [7] affirment que "si l'ampleur des forces répulsives est inversement proportionnelle au carré de la distance entre les points, l'équation représente l'énergie potentielle". On

trouve des affirmations similaires dans [13],[18],[36], Dans [21], les auteurs affirment que le critère AE "est égal au minimum de l'énergie potentielle des forces répulsives pour les points de masse unitaire si la magnitude de ces forces répulsives est inversement proportionnelle à la distance entre les points". Si le critère quantifie l'énergie potentielle d'un système de particules, la force de répulsion entre paires de particules doit être égale à la dérivé négative de l'énergie potentielle .

$$F_{ij} = -\frac{dE_{ij}^{AE}}{dD_{ij}} = -\frac{\frac{d1}{D_{ij}^2}}{dDij} = \frac{2}{D_{ij}^3} = F_{ij} \tag{3.7}$$

Par conséquent, le critère AE suppose que les forces de répulsion entre les particules sont inversement proportionnelles à la distance qui les sépare, élevée à la troisième puissance. On peut développer des critères similaires basés sur une autre relation force-distance, par exemple avec $F_{ij} = \frac{1}{D_{ij}^2}$ ce qui se traduirait par une énergie potentielle (et un critère de conception correspondant).

Il est à noter qu'avec l'augmentation de la puissance du critère d'énergie, la contribution des courtes distances tend à être de plus en plus accentuée.

La limite du critère d'énergie lorsque la puissance s'approche de l'infini correspond à une simple dépendance de la distance la plus courte (la distance entre la paire de points la plus proche). Ceci peut être démontré en calculant la limite de la racine p^{me} de E^{AE} avec une puissance p

$$lim_{p\longrightarrow\infty}\sqrt{\sum_i X_i^p} = lim_{p\longrightarrow\infty}\sqrt{\sum_{i\neq j}\frac{1}{D_{ij}^p}} = \frac{1}{D_{ij}} = \frac{1}{min_{i\neq j}D_{ij}} \tag{3.8}$$

Pour minimiser l'énergie avec l'augmentation de la puissance (ce qui correspond à la minimisation de la racine p de celle-ci), il faudrait, à la limite, maximiser le minimum sur les distances dans le domaine de conception, c'est-à-dire que le critère maximin serait de maximiser le minimum sur les distances dans le domaine de conception, c'est-à-dire que l'on retrouverait le critère maximin.

donc :

$$U = \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{D_{ij}^2} \tag{3.9}$$

Critère ϕ_p :

Morris et Mitchell ont proposé dans [27] une extension intuitivement attrayante du critère de distance maximin. Pour une conception donné, en triant toutes les distances inter-

situées dij $(1 \leq i, j \leq n, i \neq j)$, une liste de distance $(d1, d2, ..., ds)$, et une liste d'indices $(j1, j2, ..., js)$ peut être obtenue, où les d_i sont des valeurs de distance distinctes avec $d1 < d2 < ... < ds$, j_i est le nombre de paires de sites dans la conception séparées par d_i, s est le nombre de valeurs de distance distinctes.
Une conception est dite ϕ_p-optimale si elle minimise :

$$\phi_p = [\sum_{i=1}^{s} J_i d_i^{-p}]^{\frac{1}{p}} \tag{3.10}$$

telle que p est un nombre entier positif. Si p est très grand, la distance minimale $d1$ dominera tous les éléments suivants. Dans ce cas, le critère ϕ_p est équivalent au critère de distance maximin.

Critère de l'arbre de longueur minimale :

Soit $G(X, U)$ un graphe avec :
- $X = \{x_1, x_2, ..., x_n\}; x_j \in R^p, j = 1, 2, ..., n$ l'ensemble des sommets.
- $U = \{u_1, u_2, ..., u_m\}$ l'ensemble des arêtes, où $\forall 1 \leqslant i \leqslant m$, $u_i \in X \times X$ et où $X \times X = \{(x_j, x_k) | x_j \in X, x_k \in X\}$

Un arbre est un graphe simple et connexe. Un arbre est dit maximal s'il connecte tous les sommets.
L' Arbre de Longueur Minimale (ALM) d'un plan d'expériences $X_n G(X_n, U)_{ALM}$ est un arbre maximal dont la longueur totale (somme des longueurs de ses arêtes) est la plus petite parmi toutes les arbres maximaux de $G(X_n, U)$.
$G(X_U)_{ALM}$ est utilisé comme outil de classification des plans d'expériences par [14].

$$\phi_{ALM}(X_n) = \sum_{i=1}^{n-1} \lambda(u_i) \tag{3.11}$$

où $\lambda(u_i) : (x_j, x_k) ß x_j - x_k$, est la pondération, $x_j - x_k$, le poids de l'arête u_i liant les sommets x_j et x_k.
L'objectif est de trouver une manière d'optimiser la disposition des points afin de les éloigner les uns des autres.. Cet arbre est construit en utilisant l'algorithme de Prim [31].

3.4.2 Critères d'uniformité

Critère de l'entropie :

Shannon [34] a utilisé l'entropie pour quantifier la "quantité d'information" : plus l'entropie est faible, plus la connaissance est précise. D'un point de vue bayésien, plus l'entropie postérieure est faible, plus l'incertitude dans la prédiction de la réponse à des endroits non observés est faible.Minimiser l'entropie postérieure équivaut à trouver un ensemble de points de conception sur lesquels nous avons le moins de connaissances. Il a également été démontré que le critère d'entropie est équivalent à la minimisation des éléments suivants (voir, l'exemple de [22]) :
$-log|R|$, Où R est la matrice de corrélation du plan d'expérience.
$L = [x_1, x_2, .., x_n]^p$ ses éléments sont :

$$R_{ij} = exp(-\sum_{k=1}^{p} \theta_k |x_{ik} - x_{jk}|^t), 1 \leq i, j \leq n; 1 \leq t \leq 2 \tag{3.12}$$

Où $\theta_k (k = 1, ..., p)$ sont les coefficients de corrélation.

Critère de la discrépance L_2 :

L'écart L_p est une mesure de la différence entre la fonction de distribution cumulative empirique d'un plan expérimental et la fonction de distribution cumulative uniforme.
En d'autres termes, l'écart L_p est une mesure de la non-uniformité d'un plan. Parmi les écarts L_p, l'écart L_2 est le plus souvent utilisé car il peut être exprimé analytiquement et il est beaucoup plus facile à calculer. [17] a proposé trois formules de divergence L_2, dont la divergence L_2 centrée. l'écart $L2$ centré $(CL2)$ semble la plus intéressante.

$$CL_2(X)^2 = (\frac{13^2}{12}) - \frac{2}{n}\sum_{i=1}^{n}\prod_{k=1}^{p}(1 + \frac{1}{2}|x_{ik} - 0.5| - \frac{1}{2}|x_{ik} - 0.5|^2) +$$

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\prod_{k=1}^{p}(1 + \frac{1}{2}|x_{ik} - 0.5| + \frac{1}{2}|x_{jk} - 0.5| - \frac{1}{2}|x_{ik} - x_{jk}|) \tag{3.13}$$

Un plan est dit uniforme s'il minimise l'écart L_2 centré [12].

3.5 L'utilisation des hypercubes latins :

Les hypercubes latins ont plusieurs utilisations et applications dans différents domaines. Voici quelques exemples :

1-Conception d'expériences : Les plans d'hypercube latin sont utilisés dans la conception d'expériences pour organiser des essais de manière efficace et optimale. Ils permettent de réduire le nombre d'essais nécessaires tout en garantissant une répartition équilibrée des conditions expérimentales.

2-Théorie des graphes : Les hypercubes latins sont étroitement liés aux graphes de Cayley, qui sont des graphes utilisés pour représenter les symétries de groupes. Les hypercubes latins peuvent être utilisés pour étudier les propriétés et les structures des graphes de Cayley.

3-Optimisation combinatoire : Les hypercubes latins sont également utilisés comme exemples de problèmes de recherche opérationnelle pour développer et tester des algorithmes d'optimisation combinatoire. Ils offrent un cadre intéressant pour étudier les propriétés des algorithmes de résolution et explorer de nouvelles approches de recherche.

4-Applications pratiques : Les hypercubes latins ont des applications pratiques dans la planification, la logistique, la gestion des ressources, l'ordonnancement et d'autres domaines où la répartition équilibrée des valeurs est importante.

La répartition équilibrée et l'unicité des valeurs sont essentielles. Ils offrent des structures mathématiques intéressantes et des outils pour résoudre des problèmes complexes.

Les hypercubes latins offrent la possibilité de créer des modèles expérimentaux avec un nombre adapté de points, une flexibilité pour différentes techniques de modélisation, ainsi qu'une réutilisation des données existantes même en cas de réduction des dimensions. Ces caractéristiques en font une méthode attrayante pour la conception et l'analyse d'expériences informatiques.

L'optimisation des hypercubes latins bénéficie de la combinaison de stratégies d'optimisation et de la puissance de calcul des ordinateurs. Cela a permis de créer des conceptions expérimentales offrant une bonne couverture de l'espace de manière raisonnable en termes de coût de calcul.

3.6 Avantages et inconvénients :

Les hypercubes latins présentent à la fois des avantages et des inconvénients. Voici une liste des principaux :

3.6.1 Avantages :

1-Répartition équilibrée des valeurs : Les hypercubes latins garantissent une répartition équilibrée des valeurs dans une grille cubique, ce qui peut être utile dans diverses applications où une répartition uniforme est souhaitée.

2-Contraintes claires et strictes : Les contraintes d'unicité dans les hypercubes latins sont définies de manière précise, ce qui facilite la validation des solutions et garantit que chaque valeur apparaît exactement une fois par ligne, colonne et plan.

3-Applications pratiques : Les hypercubes latins trouvent des applications pratiques dans la conception d'expériences, la cryptographie, la planification et d'autres domaines où une répartition équilibrée des valeurs est nécessaire.

3.6.2 Inconvénients :

1-Complexité exponentielle : Le problème de l'hypercube latin peut devenir rapidement complexe et difficile à résoudre pour des grilles de grande taille en raison du nombre exponentiel de combinaisons possibles. La recherche exhaustive devient alors inefficace.

2-Taille limitée : La taille pratique des hypercubes latins est limitée par les ressources disponibles, car leur résolution devient rapidement coûteuse en termes de temps et d'espace pour les grilles de grande taille.

3-Non-existence dans certains cas : Il existe des conditions où un hypercube latin valide n'existe pas pour certaines tailles de grille. Dans de tels cas, il peut être difficile de trouver une solution satisfaisante.

Chapitre 4

Optimisation d'un hypercube latin

Dans ce chapitre, nous proposons de construire des LH optimaux.
Nous commençons tout d'abord par utiliser le critère maximin distance, largement employé pour l'optimisation des LH, où nous avons programmé l'alogorithme maximin décrit par Lunani [25].
Ensuite, nous utilisons le critère de l'énergie potentielle basé sur la formulation de Audez-Englais dans [8] pour générer des LH optimaux en employant l'algorithme génétique décrit dans [7] dans le but de comparer les deux approches et de voir la qualité des LH optimaux construits à travers chacune de ces approches en terme de temps de calcul et de complexité algorithmique.

4.1 Approche Maximin :

Les plans LH maximin sont des plans LH optimaux selon le critère de distance maximin introduit par [20]. L'idée est d'améliorer la propriété space filling en utilisant le critère de la distance maximin. En d'autres termes, on part d'un LH quelconque généré aléatoirement selon la définition (paragraphe 3.3.3) pour oboutir, après optimisation à un LH optimal en terme de space filling.

Un plan est appelé plan de distance maximin s'il maximise la distance minimale entre chaque paire de points du plan pour obtenir une répartition équilibrée.
Un LH maximin est

$$D = max\{min_{i \neq j} d(x_i, x_j)\} \tag{4.1}$$

La distance utilisée est la distance euclidienne :

$$d(x_i, x_j) = min_{i \neq j} \sqrt{\sum_{l=1}^{p} |x_{il} - x_{jl}|^2} \tag{4.2}$$

L'lgorithme suivant a été décrit en 1995 par Lunani [25], il sert à générer un LH à l'aide du critère de distance Maximin :

Algorithme 1 : Algorithme Maximin

---

1. Préciser le nombre de points (variables d'entrée) n, la taille p du LH et le nombre d'individus candidats T.
2. Créer n niveaux pour chaque variables.
3. Générez LH.
   3.1. Pour chaque variable d'entrée et dans chaque niveau ,x_{ij}, où $i = 1, .., n$, $j = 1, .., p$:
   3.2. Permuter aléatoirement l'ordre des éléments de chaque colonne.
4. Calculer D_{min} la plus petite distance entre 2 points de conception quelconques.
   4.1. Calculer les distances entre les points comme suit :
   $$D(k) = \sqrt{\sum_{i=1}^{p} |x_{il} - x_{jl}|^2}$$
   pour $j, i = 1, ..., n$,$j \neq i$, $l = 1, ..., p$ où : $k = 1, ..., n(n-1)/2$.
   4.2. Déterminer la distance minimale :
   $$D_{min} = min_k D(k).$$
   4.3. Calculer le nombre de $D(k)$ égale à D_{min}
   $$D'_{min} = \sum_k J_k \text{ où } J_k = 1 \text{ si } D(k) = D_{min}$$
   $$J_k = 0 \text{ si } D(k) > D_{min}$$
5. Répéter les étapes **3** et **4** T fois et sélectionner le LH qui maximize D_{min}, en cas d'égalité choisir celle qui minimise D'_{min}

Cet algorithme est utilisé pour générer aléatoirement un plan LH optimal. Dans le paragraphe suivant nous en expliquons les étapes en détail :

1- Nous commençons par spécifier les paramètres d'entrée :
- n : le nombre de points.
- p : les dimensions.
- T : le nombre des LH candidats.

2- Créer n niveaux pour chaque variables d'entrée : crée n lignes,
diviser l'intervalle en n intervalle égaux.

3- Générer LH :

- Pour chaque variable d'entrée $[l = 1, ..., p]$, sélectionner un seul élément dans chaque niveaux $[i, j = 1, ..., n]$.

- L'élément sélectionné peut être choisi de manière aléatoire.

- Pour chaque colonne correspondant à une variable d'entrée, permuter de manière aléatoire l'ordre des éléments dans LH.

- Il ne doit y avoir aucune répétition dans chaque colonne.

4- Calculer les distances :

- Calculer le nombre de distances $d(k)$ égales à d_{min} (D'_{min}) où :
$k = 1, ..., \dfrac{n \times (n-1)}{2}$.

- Calculer toutes les distances entre les paires de points (x_{il}, x_{jl}) dans chaque LH, où j et i sont des indices des niveaux.

- Appliquer la formule de distance euclidienne : $d(k) = \sqrt{\sum_{l=1}^{p} |x_{il} - x_{jl}|^2}$.

- Trouver la plus petite distance D_{min} parmi toutes les distances calculées,
$D_{min} = min_{j \neq i} d(k)$

- Compter le nombre d'occurrence qui correspondent au nombre de distances égales à à D_{min}.

5-répéter les étape 3 et 4 T fois pour générer T LH candidat et sélectionner le LH qui maximise D_{min} :

- Parmi les distances minimales trouvées de chaque LH nous allons choisir celle qui maximise la distance minimale.

Si il en existe plusieurs (dmin)
alors choisir le LH correspondant au nombre d'occurrences qui est minimum.

Pour mieux comprendre l'algorithme donné, on suggère l'exemple suivant bien détaillé :

1- Nous commençons par spécifier les paramètres d'entrée :
- n : 4.
- p : 3.
- T : 3.

2- Nous avons crée n niveaux pour chaque variables d'entrée :

$$LH = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 3 & 1 \\ 2 & 4 & 4 \\ 1 & 2 & 3 \end{pmatrix}$$

3- Génération des LH :

$$LH = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 4 & 2 \\ 2 & 3 & 4 \\ 1 & 2 & 3 \end{pmatrix}$$

$$LH = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 4 & 3 \\ 2 & 3 & 1 \\ 1 & 2 & 4 \end{pmatrix}$$

4- calculons les distances :
-Calculons le nombre de distances k, et la distance euclidienne :
-$k = 1$, $i = 1$, $j = 2$, $l = 1 jusqu'a 3$.
$d(1) = \sqrt{|4-3|^2 + |1-4|^2 + |2-3|^2} = 3.31$.
-$k = 2$, $i = 1$, $j = 3$, $l = 1$ jusqu'à 3.
$d(2) = 3$.
-$k = 3$, $i = 1$, $j = 4$, $l = 1$ jusqu'à 3.
$d(3) = 3.74$.

-$k = 4$, $i = 2$, $j = 3$, $l = 1$ jusqu'à 3.

$d(4) = 2.44$.

-$k = 5$, $i = 2$, $j = 4$, $l = 1$ jusqu'à 3.

$d(5) = 3$.

-$k = 6$, $i = 3$, $j = 4$, $l = 1$ jusqu'à 3.

$d(6) = 3.31$.

-La plus petite distance : $D_{min} = 2.44$, avec un nombre d'occurrence égale à 1.

5-Les étapes 3 et 4 doit répéter pour $T = 3$.

3-$LH = \begin{pmatrix} 1 & 4 & 3 \\ 2 & 3 & 4 \\ 4 & 2 & 2 \\ 3 & 1 & 1 \end{pmatrix}$

$LH = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 4 \\ 4 & 2 & 2 \\ 3 & 1 & 1 \end{pmatrix}$

$LH = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 4 & 1 \\ 4 & 3 & 2 \\ 3 & 1 & 3 \end{pmatrix}$

4- calculons les distances :

-Calculons le nombre de distances k, et la distance euclidienne :

-$k = 1$, $i = 1$, $j = 2$, $l = 1$ jusqu'à 3.

$d(1) = 3.74$

-$k = 2$, $i = 1$, $j = 3$, $l = 1$ jusqu'à 3.

$d(2) = 3.74$.

-$k = 3$, $i = 1$, $j = 4$, $l = 1$ jusqu'à 3.

$d(3) = 2.44$.

-$k = 4$, $i = 2$, $j = 3$, $l = 1$ jusqu'à 3.

$d(4) = 2.44$.

-$k = 5$, $i = 2$, $j = 4$, $l = 1$ jusqu'à 3.

$d(5) = 3.74$.

-$k = 6$, $i = 3$, $j = 4$, $l = 1$ jusqu'à 3.

$d(6) = 2.44$.

-La plus petite distance : $D_{min} = 2.44$, avec un nombre d'occurrence égale à 3.

3-$LH = \begin{pmatrix} 2 & 4 & 4 \\ 4 & 3 & 1 \\ 3 & 2 & 3 \\ 1 & 1 & 2 \end{pmatrix}$

$LH = \begin{pmatrix} 2 & 4 & 4 \\ 4 & 2 & 1 \\ 3 & 3 & 3 \\ 1 & 1 & 2 \end{pmatrix}$

$LH = \begin{pmatrix} 2 & 4 & 2 \\ 4 & 2 & 4 \\ 3 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix}$

4- calculons les distances :

-Calculons le nombre de distances k, et la distance euclidienne :

-$k = 1$, $i = 1$, $j = 2$, $l = 1$jusqu'à 3.

$d(1) = 3.46$

-$k = 2$, $i = 1$, $j = 3$, $l = 1$jusqu'à 3.

$d(2) = 1.73$.

-$k = 3$, $i = 1$, $j = 4$, $l = 1$ jusqu'à 3.

$d(3) = 3.31$.

-$k = 4$, $i = 2$, $j = 3$, $l = 1$ jusqu'à 3.

$d(4) = 3.31$.

-$k = 5$, $i = 2$, $j = 4$, $l = 1$ jusqu'à 3.

$d(5) = 3.31$.

-$k = 6$, $i = 3$, $j = 4$, $l = 1$ jusqu'à 3.

$d(6) = 3.46$.

-La plus petite distance : $D_{min} = 1.73$, avec un nombre d'occurrence égale à 1.

-On va sélectionner un LH qui maximise D_{min}.

-Parmi les distances minimales trouvées de chaque LH nous allons choisir celle qui maximise la distance minimale.

-On va créer un vecteur qui rempli les distances minimales des LH,

$LH = \begin{pmatrix} 2.44 & 2.44 & 1.73 \end{pmatrix}$

-Distance maximale des distances minimales choisies :

$maxDistance = 2.44$, $nombred'occurrence = 1$

-Meilleur LH :

$$LH = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 4 & 3 \\ 2 & 3 & 1 \\ 1 & 2 & 4 \end{pmatrix}$$

Dans la figure suivante (figure 4.1), on retrouve un LH (10x2) obtenu par l'algorithme maximin après T=30 itérations. On peut remarquer que c'est un LH optimisé puisque les points du plan sont bien répartis sur l'espace. A partir d'un LH quelconque (aléatoire), on a aboutit à un LH optimisé en termes de space filling (après optimisation).

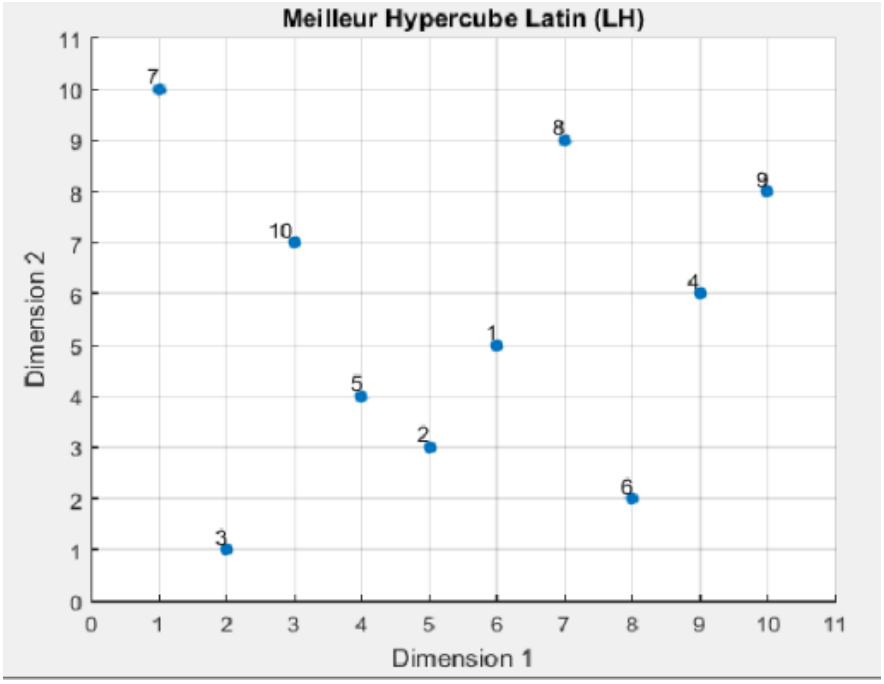

FIGURE 4.1 – Les points du plan LH pour $n = 10$ et $p = 2$ généré par algorithme maximin

Meilleur hypercube latin :

$$LH = \begin{pmatrix} 6 & 5 \\ 5 & 3 \\ 2 & 1 \\ 9 & 6 \\ 4 & 4 \\ 8 & 2 \\ 1 & 10 \\ 7 & 9 \\ 10 & 8 \\ 3 & 7 \end{pmatrix}$$

maxDistance=2.2361

Nombre d'occurences de dmin (dmin)=1

4.2 Approche par algorithme génétique :

L'algorithme génétique est utilisé dans de nombreux domaines pour résoudre des problèmes d'optimisation et rechercher des solutions approchées.

Nous allons rappeler le principe de l'énergie potentielle : un système composé de points de masse unitaire exercent des forces répulsives les uns sur les autres, ce qui confère au système une énergie potentielle. La minimisation de l'équation

$$U = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{D_{ij}^2} \tag{4.3}$$

produira un système de points répartis aussi uniformément que possible. Où U est l'énergie potentielle et D_{ij}^2 est la distance entre les points i et j

Avant d'expliquer en détails le fonctionnement d'un algorithme génétique, Résolvant par l'algorithme génétique qui est un processus itératif de recherche exploratoire.

Adaptation de l'algorithme génétique :

Algorithme 2 : Algorithme génétique

1. Initialisation de la population initiale $P0$ (population de génération 0) qui contient des LH,
2. Évaluation des LH de la population Pi par la fonction de fitness,
   $f_i = (F_{max} + F_{min=}) - F_i$

   F_{max}, F_{min} et F_i sont respectivement les valeurs maximum, minimum et individuelle de la fonction objectif.
   où $F = U$ si $p = 0$, sinon $F = 3 * p * U$, p est nb de points répéter dans le même niveau du LH, et U est l'énergie potentielle,
3. Sélection des meilleurs LH de Pi ayant une fitness minimale,
4. Mutation par permutation opérée aléatoirement sur un chromosome (LH converti en vecteur ligne) également choisi au hasard,
5. Croisement avec un point deux-à-deux des chromosomes (LH) sélectionnés pour générer de nouveaux individus, les « descendants »,
6. Remplacement et génération de la nouvelle population Pi+1,
7. Répéter le processus à partir de 2 tant que le critère d'arrêt (nombre de génération) n'est pas satisfait.

Dans le paragraphe suivant nous expliquons l'algorithme génétique en détails :

Les opérateurs génétiques de mutation et de croisement peuvent amener un individu à enfreindre la règle exigeant la présence d'un point pour chaque niveau. Ainsi, la fonction objectif équation (4.3) revoir la notation de l'énergie potentielle est modifiée pour pénaliser le plan comportant plus d'un point par niveau.

La fonction de pénalisation :

La fonction de pénalisation consiste à calculer le nombre de points dans un niveau et augmenter la fonction objectif, la fonction objectif modifiée F est alors :

$$\text{si } p = 0, \text{ alors } F = U$$

si $p > 0$, alors

$$F = 3 * p * U \tag{4.4}$$

où : p représente le nombre de points répétés dans le même niveau, U est l'énergie potentielle du LH et 3 est une constante choisie telle que l'AG fonctionne en recherchant dans l'espace de conception l'individu ayant la fitness la plus élevée l'équation (4.5), minimisant ainsi l'équation. (4.4). L'individu le plus apte est celui qui a une énergie plus faible. Par conséquent la fonction de fitness est :

$$f_i = (F_{max} + F_{min=}) - F_i \tag{4.5}$$

F_{max}, F_{min} et F_i sont respectivement les valeurs maximum, minimum et individuelle de la fonction objectif.

L'optimisation nécessite également le codage des variables de conception du problème. Les variables de conception de ce problème sont les LH, et le codage de ces derniers est donc nécessaire. Nous avons utilisé le codage réel. Le codage du LH est effectué à l'aide de la méthode des coordonnées, Les n premières variables de conception sont les coordonnées x_1, les n variables de conception suivantes sont les coordonnées x_2, etc. jusqu'à p variables. La formulation est comme suit :
les n premiers nombres sont une séquence aléatoire de nombres entre 1 et n , les n nombres suivants sont une séquence aléatoire de nombres entre 1 et n. Cette opération est répétée jusqu'à p. S'il n'y a pas de répétition de nombres dans chaque séquence et la règle d'un point par niveau n'est donc pas enfreinte.

Un chromosome est la représentation d'un individu par un vecteur ligne, ce vecteur est constitué de lignes de l'individu telle que chaque ligne de l'individu est suivie de la ligne suivante du même individu et ainsi de suite jusqu'à parcourir toutes les lignes et enfin construire le chromosome.

La mutation est opérée aléatoirement sur un chromosome codé également choisi au hasard, en sélectionnant deux positions aléatoirement dans ce chromosome et les interchangées, donc elle introduit des modifications aléatoires dans le chromosome.

Le croisement est appliqué pour créer de nouveaux chromosomes enfants en combinant les caractéristiques de deux chromosomes parents sélectionnés.

Cette opération se fait, en sélectionnant un point de croisement aléatoire dans l'un des chromosomes choisis, la première partie de l'enfant $E1$ est copiée à partir du premier parent jusqu'au point de croisement, puis la deuxième partie est copiée du deuxième parent à partir du point de croisement jusqu'à la fin du chromosome. Également pour l'enfant $E2$, la première partie du parent 2 est copiée à partir du deuxième parent jusqu'au point de croisement, puis la deuxième partie du premier parent à partir du point de croisement jusqu'à la fin du chromosome.

Dans la figure ci-après, nous présentons le LH(10x2) résultant de l'application de l'algorithme génétique.
On commence avec une population de LH quelconques (aléatoire) et on aboutit à un LH optimisé en terme de space filling (après optimisation).

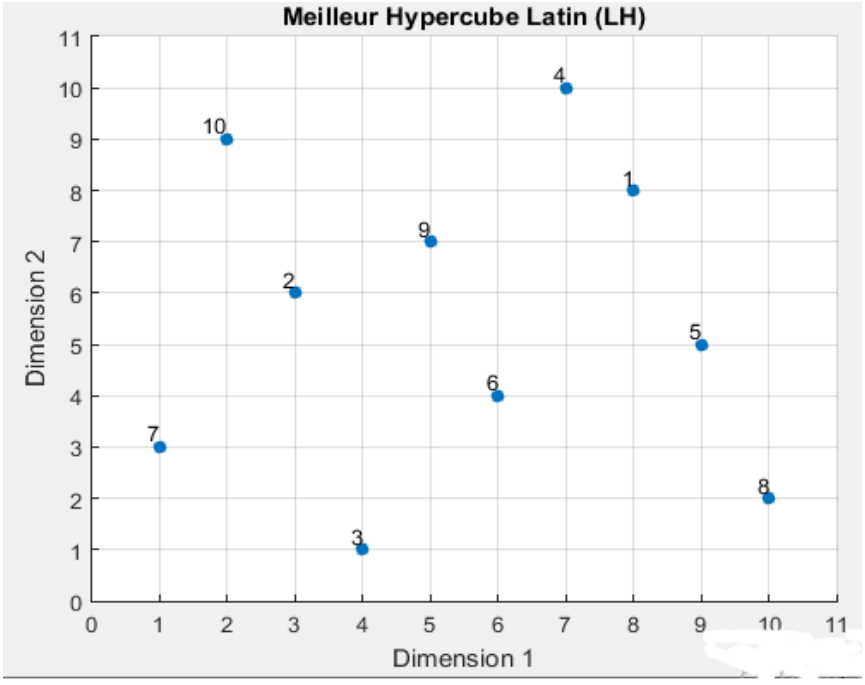

FIGURE 4.2 – Les points du plan LH pour $n = 10$ et $p = 2$ généré par algorithme génétique

Meilleur hypercube latin :

$$LH = \begin{pmatrix} 8 & 8 \\ 3 & 6 \\ 4 & 1 \\ 7 & 10 \\ 9 & 5 \\ 6 & 4 \\ 1 & 3 \\ 10 & 2 \\ 5 & 7 \\ 2 & 9 \end{pmatrix}$$

Génération 10 :

Fitness maximale de la génération : 81.493.

-Le tableau suivant présente une comparaison entre les deux approches en termes de temps d'exécution. Différents LH sont générés et le temps de calcul nécessaire et évalué pour les deux approches.

LH	Algorithme maximin nb candidats T=30	Algotithme génétique, maxGénération=30
10 × 4	9.131 s	27.304 s
50 × 5	10.042 s	126.206 s
70 × 10	15.292 s	141.828 s
100 × 10	17.462 s	289.832 s

comparaison entre algorithme maximin et algorithme génétique

Conclusion

Dans ce mémoire, on s'est proposé d'étudier et d'optimiser un type des plans d'expériences dit les plans hypercubes latins (LH). Les LH sont largement utilisés en simulation numérique. Ils permettent un choix judicieux pour la localisation des points de la base d'apprentissage et réduisent considérablement le temps de calcul.

Nous avons souligné que pour construire de bons LH, il est nécessaire de passer par un critère d'optimisation spécifique. Dans notre travail, nous nous sommes basés sur deux critères particuliers : le critère maximin et le critère de l'énergie potentielle. Nous avons traité et programmé deux métaheuristiques pour la conception de plans hypercubes latins répondant à ces deux critères.

A travers ce mémoire, nous avons contribué à la compréhension des méthodes de conception de plans d'expériences en se concentrant sur les plans hypercubes latins et en mettant en œuvre deux approches métaheuristiques efficaces pour les critères maximin et l'énergie potentielle. Les résultats obtenus ouvrent la voie à de nouvelles perspectives de recherche et d'application dans le domaine de l'optimisation des expériences.

Aussi, Il pourrait être intéressant d'étudier la prédiction des métamodèles obtenus par les LH produits à l'aide de ces deux approches, ou encore par d'autre approches et d'autres critères, sur des problèmes réels et de faire une analyse statistique (variance, biais, etc) permettant ainsi de venir en support aux techniques de modélisations et surtout d'améliorer les paramètres du système étudié.

Annexes

Le code maximin implémenté sous matlab :

```
clc
clear all
n = input('donner la taille de l hypercube = ');
p = input('donner la dimension de l hypercube = ');
T = input('donner le nombre de candidats = ');

    % Initialiser les variables
bestLH = zeros(n, p);
bestDmin = -Inf;
bestNumDmin = Inf;
chosenDmin = zeros(T, 1);

    % Répéter pour T candidats LH
for t = 1 : T
% Générer les points Hypercube Latin
lh = lhsdesign(n, p);
lh = ceil(lh * n);
lh = lh(:, randperm(p));
for i = 1 : p
permindices = randperm(n);
lh(:, i) = lh(permindices, i)
end

    % Calculer les distances inter-points
distances = zeros(n * (n - 1)/2, 1);
k = 1;
forj = 1 : n
forl = (j + 1) : n
```

$distances(k) = sqrt(sum(abs(lh(j,:) - lh(l,:)).^2))$;
$k = k + 1$;
end
end

% Trouver la distance minimale et son nombre d'occurrences
$[minDistance] = min(distances)$
$numOccurrences = sum(distances == minDistance)$

% Mettre à jour le meilleur LH si nécessaire
if $numOccurrences < bestNumDmin || (numOccurrences == bestNumDminminDistance > bestDmin)$
$bestLH = lh$;
$bestDmin = minDistance$;
$bestNumDmin = numOccurrences$;
end

% Sauvegarder la distance minimale dans chosenDmin
$chosenDmin(t) = minDistance$;
end

% Trouver la distance maximale parmi les distances minimales choisies
$maxDistance = max(chosenDmin)$

% Afficher le meilleur LH et ses propriétés
disp('Meilleur Hypercube Latin (LH) :')
disp(bestLH)
disp('Nombre d'occurrences de dmin (dmin) :')
disp(bestNumDmin)
disp('Distances minimales choisies :')
disp(chosenDmin)
disp('Distance maximale des distances minimales choisies :')
disp(maxDistance)
Extraire les coordonnées des points du meilleur LH

$x = bestLH(:, 1)$;
$y = bestLH(:, 2)$;

% Tracer les points
scatter(x, y, 'filled') ;
xlabel('Dimension 1') ;
ylabel('Dimension 2') ;
title('Meilleur Hypercube Latin (LH)') ;

% Ajouter des étiquettes aux points (facultatif)
labels = cellstr(num2str((1 :n)')) ;
text(x, y, labels, 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right') ;

% Ajuster les limites de l'axe
axis([0 n+1 0 n+1]) ;

% Afficher la grille (facultatif)
grid on ;

Le code algorithme génétique implémenté sous matlab :

---

clc ;
clear all ;
taille hypercube = input('donner la taille de l hypercube = ') ;
dimensions = input('donner les coordonnées de chaque point = ') ;
taillepopulation = input(' donner la taille de la population') ;
maxGenerations = input('donner le nombre de Generations = ') ;
select=input('donner le nombre d individus selectionés, obligatoirement < a la taille de la population =') ;
k=1 ;
% Création de la population
population = cell(taillepopulation, 1) ;
while k <=taillepopulation
% Générer les points Hypercube Latin
$lh = lhsdesign(taille_hypercube, dimensions)$;

% Convertir les échantillons en entiers
$lh = ceil(lh * taille_hypercube)$;

% Permuter aléatoirement l'ordre des colonnes
$lh = lh(:, randperm(dimensions))$;

populationk = lh ;
$k = k + 1$;
end
$bestGeneration = 0$;
$bestLH = []$;

for generation = 1 :maxGenerations
disp(['Generation ', num2str(generation)]) ;

% Calcul de l'énergie pour chaque individu

energie = zeros(taillepopulation, 1) ; for $k1 = 1 : taillepopulation$

$lh = populationk1$;
$energie(k1) = calculenergie(lh, taillehypercube)$;

end
$p = 0$;
repetition=zeros(taillepopulation,1) ;
for $k2 = 1 : taillepopulation$
$repetition(k2) = calculrepetition(lh, p, taillehypercube, dimensions)$;
end
$F = zeros(taillepopulation, 1)$;
for $k3 = 1 : taillepopulation$
if $repetition(k3) == 0$
$F(k3) = energie(k3)$;
else
$F(k3) = 3 * repetition(k3) * energie(k3)$;
end
end
$F_{max} = max(F)$;
$F_{min} = min(F)$;
$fitness = zeros(taillepopulation, 1)$;
for $s = 1 : taillepopulation$
$fitness(s) = (F_{max} + F_{min}) - F(s)$;

end
% Sélection des 20 individus de fitness minimale
, sortedIndices
= sort(fitness) ;
$selectedIndices = sortedIndices(1 : select)$;
$selectedPopulation = population(selectedIndices)$;

% Transformation des individus sélectionnés en vecteurs ligne
$selectedPopulationVectors = cell(select, 1)$;
for $k4 = 1 : select$
$selectedPopulationVectorsk4 = reshape(selectedPopulationk4, 1, [])$;
end

% Répéter le processus de sélection, croisement et mutation pour les 20 prochaines générations
for $k5 = 1 : select$
% Sélection aléatoire d'un vecteur parmi les individus sélectionnés
$randomIndex = randi([1, select])$;
$selectedVector = selectedPopulationVectorsrandomIndex$;

% Sélection aléatoire de deux indices uniques
$randomIndices = randperm(length(selectedVector), 2)$;

% MUTATION (Échange de positions entre les deux éléments)
$indiceale = selectedVector(randomIndices(1))$;
$selectedVector(randomIndices(1)) = selectedVector(randomIndices(2))$;
$selectedVector(randomIndices(2)) = indiceale$;

% Sélection aléatoire de deux indices uniques pour les parents
$parentIndices = randperm(select, 2)$;

% Récupération des vecteurs ligne des parents sélectionnés
$parent1 = selectedPopulationVectorsparentIndices(1)$;
$parent2 = selectedPopulationVectorsparentIndices(2)$;

% Sélection aléatoire de l'indice de division
$indicedivision = randi([1, length(parent1)])$;

% Division des parties avant et après l'indice de division
$part1Parent1 = parent1(1 : indicedivision)$;
$part2Parent1 = parent1(indicedivision + 1 : end)$;

$part1Parent2 = parent2(1 : indicedivision)$;
$part2Parent2 = parent2(indicedivision + 1 : end)$;

% Formation des enfants en combinant les parties des parents
$enfant1 = [part1Parent1, part2Parent2]$;
$enfant2 = [part1Parent2, part2Parent1]$;

end

% Remplacement de l'enfant1, l'enfant2 et l'individu échangé dans la population
$selectedPopulationVectorsparentIndices(1) = enfant1$;
$selectedPopulationVectorsparentIndices(2) = enfant2$;
$selectedPopulationVectorsrandomIndex = selectedVector$;

% Conversion des vecteurs ligne en matrices lh
$populationselectedIndices(parentIndices(1)) = reshape(enfant1, taillehypercube, dimensions)$;
$populationselectedIndices(parentIndices(2)) = reshape(enfant2, taillehypercube, dimensions)$;
$populationselectedIndices(randomIndex) = reshape(selectedVector, taillehypercube, dimensions)$;

$energie = zeros(taillepopulation, 1)$;
$repetition = zeros(taillepopulation, 1)$;
for $k6 = 1 : taillepopulation$
$lh = populationk6$;
$energie(k6) = calculenergie(lh, taillehypercube)$;
$p = 0$;
$repetition(k6) = calculrepetition(lh, p, taillehypercube, dimensions)$;
end

$F = zeros(taillepopulation, 1)$;
for $k7 = 1 : taillepopulation$
if $repetition(k7) == 0$
$F(k7) = energie(k7)$;
else
$F(k7) = 3 * repetition(k7) * energie(k7)$;

end
end
$F_{max} = max(F)$;
$F_{min} = min(F)$;
for $k9 = 1 : taillepopulation$

$fitness(k9) = (F_{max} + F_{min}) - F(k9)$;
end
$[maxFitnessGeneration, bestIndexGeneration] = max(fitness)$;
$bestFitness = maxFitnessGeneration$;
$bestGeneration = generation$;

disp(['Fitness maximale de la génération : ', num2str(maxFitnessGeneration)]) ;
disp('————————————————') ;

% Affichage du meilleur LH de la génération actuelle
disp('Meilleur LH de la génération :') ;
disp(populationbestIndexGeneration) ;
disp('————————————————') ;

imagesc(populationbestIndexGeneration) ;
bestLH=populationbestIndexGeneration ;
end

% Appel de la fonction calculenergie
energie= calculenergie(lh, taille hypercube)
% Affichage
disp(energie) ;

% Appel de la fonction calculrepetition
repetition= calculrepetition (lh, p, taille hypercube, dimensions)
% Affichage
disp(repetition) ;

```
    % Extraire les coordonnées des points du meilleur LH
x = bestLH( :, 1) ;
y = bestLH( :, 2) ;

    % Tracer les points
scatter(x, y, 'filled') ;
xlabel('Dimension 1') ;
ylabel('Dimension 2') ;
title('Meilleur Hypercube Latin (LH)') ;

    % Ajouter des étiquettes aux points (facultatif)
labels = cellstr(num2str((1 :taille hypercube)')) ;
text(x, y, labels, 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right') ;

    % Ajuster les limites de l'axe
axis([0 taille hypercube+1 0 taille hypercube+1]) ;
% Afficher la grille (facultatif)
% Afficher la grille
grid on ;
```

Bibliographie

[1] -A. Marrel, B. Iooss, F, Van Dorpe, and E. Volkova, (2008). An efficient methodology for modeling complex computer codes with Gaussian processes. Computational Statistics and Data Analysis, 52 :4731-4744.

[2] -Audze, P. and Eglais, V. (1977). New approach for planning out of experiments. Problems of Dynamics and Strengths, 35 :104–107.

[3] -A. Saltelli, K. Chan, and E.M, (2000). Scott, editors. Sensitivity analysis. Wiley Series in Probability and Statistics. Wiley.

[4] -Baeck et al., Baeck T., Fogel D.B., and Michalewicz Z, (1997). Handbook of Evolutionary Computation. Institute of Physics Publishing and Oxford University Press.

[5] -Baeck T., Fogel D.B., and Michalewicz Z, (2000). editors. Evolutionary Computation : Advanced Algorithms and Operators. Institute of Physics Publishing, Bristol.

[6] -Banzhaf et al, Banzhaf W., Colin R, (1999). Foundations of Genetic Algorithms .

[7] -Bates, S. J., Sienz, J. and. Langley, D. S, 2003, "Formulation of the Audze–Eglais Uniform Latin Hypercube design of experiments", J. Advances in Engineering Software, vol. 34/8 pp. 493 – 506.

[8] -B. Iooss, F. Van Dorpe, and N. Devictor, (2006). Response surfaces and sensitivity analyses for an environmental model of dose calculations. Reliability Engineering and System Safety, 91 :1241-1251.

[9] -Charbonneau, Charbonneau P, (2002). An introduction to genetic algorithms for numerical optimization. NCAR Technical Note 450+IA (Boulder : National Center for Atmospheric Research), Etats-Unis.

[10] -Dréo et al. Dréo J., Pétrowski, A. Siarry P., and Taillard É. D. (2006), Metaheuristics for Hard Optimization : Methods and Case Studies, 3-540-23022-X, Springer.

[11] -E. Volkova, B. Iooss, and F. Van Dorpe, (2008). Global sensitivity analysis for a numerical model of radionuclide migration from the RRC "Kurchatov Institute" radwaste disposal site. Stochastic Environmental Research and Risk Assesment, 22 :17-31.

[12] -Fang, K.-T., Ma, C.-X., and Winker, P. (2002). Centered L2-discrepancy of random sampling and Latin hypercube design, and construction of uniform designs. Mathematics of Computation, 71(237) :275–296.

[13] -Felipe A. C. Viana,(2013),. "Things you wanted to know about the Latin hypercube design and were afraid to ask." Proceedings of the 10th World Congress on Structural and Multidisciplinary Optimization, May 19-24, Orlando, Florida, USA. Probabilistics Laboratory, GE Global Research, Niskayuna, NY, USA.

[14] -Franco, J. (2008). Planification d'expériences numériques en phase exploratoire pour la simulation des phénomènes complexes. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne.18

[15] -Goldberg, Goldberg G.E. (1994), Algorithmes génétiques, Éditions Addison-Wesley, Paris, 1994.

[16] -Heudin, Heudin J-C, (1994). La Vie artificielle, éditions Hermès Science, Paris, 1994.

[17] -Hickernell, F. J., 1998, "A generalized discrepancy and quadrature error bound", Mathematics of Computation, 67, 299-322.

[18] - Husslage BGM, Rennen G, van Dam ER, den Hertog D, 2011. Space-filling latin hypercube designs for computer experiments. Optim Eng ;12(4) :611–30.

[19] -Jin, R., Chen, W., and Sudjianto, A. (2005). An efficient algorithm for constructing optimal design of computer experiments. Journal of Statistical Planning and Inference, 134(1) :268–287.

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

[21] -Kovalovs A., Rucevskis S.. Identification of elastic properties of composite plate.In : Proceedings of annual conference on functional materials and nanotech nologies-FMNT ; vol. 23 of IOP conf. series : Materials Science and Engineer ing. Iop publishing, Ltd.

[22] -Koehler, J.R. and Owen, A.B., (1996), Computer experiments, in Ghosh, S.and Rao, C.R., eds, Handbook of Statistics, 13, 261 –308, Elsevier Science, New York.

[23] -LE GUIBAN, Kaourintin.(2018). Hypercubes Latins maximin pour l'échantillonnage de systèmes complexes. Thèse de doctorat, Université Paris-Saclay, Ecole doctorale n°580, Sciences et Technologies de l'Information et de la Communication.

[24] - M. Dorigo, V. Maniezzo, and A. Colorni. "The ant system : Optimization by a colony of cooperating ants". IEEE, Transactions on Systems, Management and Cybernetics, 26, pp.1–13.

[25] M. Lunani, A. Sudjianto, P.L. Johnston, (1995). Generating efficient training samples for neural networks using Latin Hypercube samplingIntelligent Engineering Systems Through Artificial Neural Networks, vol. 5, ASME Press, pp. 209–214

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

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

[28] -Papadimitriou et al., 1982 Papadimitriou C.H and Steiglitz K., Combinatorial Optimization : Algorithms and Complexity. Prentice-Hall, 1982.

[29] -PAP 94 C, (1994). Papadimitriou. Computational Complexity. Addison Wesley.

[30] -Park J.S. (1994). Optimal Latin hypercube designs for computer experiments. J. of Statist. Planning and Inference 39, 95-111.

[31] -Prim, R. C. (1957). Shortest connection networks and some generalizations. Bell system technical journal, 36(6) :1389–1401.

[32] -Pronzato, L., Müller, W.G., (2012). Design of computer experiments : space filling and beyond. Stat. Comput. 22 (3), 681e701.

[33] -Rimmel, A., Teytaud, F. (2023). A Survey of Meta-heuristics Used for Computing Maximin Latin Hypercube. Supélec E3S, France. Université Lille Nord de France.

[34] -Shannon, C.E., 1948, A mathematical theory of communication, Bell System Technical Journal, 27 :379-423, 623-656.

[35] -Van Dam, E.R., Husslage, B., Den Hertog, D., Melissen, H.(2007). Maximin latin hypercube designs in two dimensions. Operations Research 55(1), 158–169 .

[36] -Vose, 1998 Vose M. D., (1998). The Simple Genetic Algorithm : Foundations and Theory. MIT Press.

[37] -Wallis, W.D., George, J., (2011). Introduction to Combinatorics. CRC Press, London.

[38] -Xiong, F., Xiong, Y., Chen, W., Yang, S., (2009). Optimizing Latin hypercube design for sequential sampling of computer experiments. Eng. Optim. 41 (8), 793e810.