الجمهورية الجزائرية الديمقراطية الشعبية

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida 1

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUES

MÉMOIRE

En vue de l'obtention du diplôme de Master

Spécialité : Mathématiques

Option : Recherche Opérationnelle

THÈME

PROPOSITION ET SIMULATION D'UN PROCESSUS PONCTUEL A INTERACTION DE LA PLUS COURTE DISTANCE

Présenté par : Benfettoum Kaouter et Djidel Karima

Soutenu devant le jury composé de :

Dr. TAMI O.	Maitre de Conférences B	Président
Dr. KERDJOUDJ S.	Maitre de Conférences A	Examinateur
Dr. ELMOSSAOUI H.	Maitre de Conférences A	Encadreur
Mr. AIT AMEUR A.	Doctorant	Co-encadreur

2023/2024

REMERCIEMENTS

Nous remercions **DIEU** ,le tout puissant pour la volonté,la patience et le courage qu'il nous a accordés pour mener à terme ce travail.

Nous tenons à exprimer notre sincère reconnaissance au **Dr. ELMOSSAOUI Hichem** pour avoir accepté de nous encadrer pour son attention discréte, ses recommandations mesurées et ses précieux conseils et surtout pour ses qualités hummaines et scientifiques toujours en toute modestie, sa passion du métier qu'il sait rendre contagieuse et la confiance qu'il a bien voulu nous accorder tout au long de ce travail.

Ce travail ne serait pas aussi riche et n'aurrait pas pu avoir le jour sans l'aide et l'encadrement de **Mr AIT AMEUR Ahmed**, on le remercie pour la qualité de son encadrement exceptionnel, pour sa patience, sa rigueur et sa disponibilité durant notre préparation de ce mémoire.

Nous tenons à remercier sincèrement **Dr TAMI Omar** et **Dr KERDJOUDJ Samia** d'avoir accepté d'examiner notre mémoire de fin d'études. Leurs expertises et conseils seront d'une grande valeur pour nous.

Nous tenons à exprimer notre gratitude à tous les enseignants du département de mathématiques. Particulièrement au chef de département **Dr. TAMI Omar**.

Nous remercions nos trés chers parents pour leur soutien moral qui nous a permis de réussir et de terminer nos études.

DÉDICACE

Avec toute ma reconnaissance, je dédie ce modeste travail à ceux pour qui, quels que soient les mots choisis, je n'arriverai jamais à exprimer mon amour sincère.

À la femme qui a souffert sans jamais me laisser souffrir, qui n'a jamais refusé mes demandes et qui n'a épargné aucun effort pour me rendre heureuse,

mon adorable mère.

À l'homme, mon précieux cadeau de Dieu, à qui je dois ma vie, ma réussite et tout mon respect,

mon cher père.

À **mes chères soeurs et mes frères**. Que Dieu les protège et leur offre la chance et le bonheur.

Sans oublier mon binôme "Karima" pour son soutien moral, sa patience et sa compréhension tout au long de ce projet.

À tous les membres de ma famille et à toute personne portant le nom BENFETTOUM ou HANI, je dédie ce travail à tous ceux qui ont contribué à ma réussite.

BENFETTOUM KAOUTER

DÉDICACE

Le voyage n'était pas court et n'aurait pas dû l'être, et la route n'était pas pleine de facilités, mais moi, par ma volonté, je l'ai fait. Avec l'aide du Tout-Puissant, j'ai pu réaliser ce modeste travail que je dédie.

À ma source de joie, **ma mère**. Elle m'a donné l'espoir et le courage pour les moments difficiles.

À mon **cher père**, qui m'a inculqué la discipline, les valeurs de la réussite et le respect d'autrui. Que Dieu, le Tout-Puissant, préserve leur santé et une longue vie.

À mes **chères soeurs**, témoins des étapes de ma vie, dans ma joie et ma tristesse, pour leurs encouragements permanents et leur soutien moral.

À mes **chers frères**, mon soutien et ma source de force, qui sont toujours derrière moi.

Sans oublier mon binôme "Kaouter" pour son soutien moral, sa patience et sa compréhension tout au long de ce projet.

À mes amis et à mes proches.

À tous ceux qui m'ont aidé après Dieu.

DJIDEL KARIMA

الملخص

تستكشف هذه الأطروحة التقاطع بين العمليات النقطية ومشاكل التحسين، وتقدم وجهات نظر جديدة في نظرية الاحتمالات و الإحصاء المكاني .فهي تجمع بين الأسس النظرية المتينة والتطبيقات العملية المتقدمة، من خلال فحص نماذج مختلفة من العمليات النقطية مثل عمليات بواسون والعمليات ذات الحدين، بالإضافة إلى مشاكل التحسين التوافقي الكلاسيكية مثل مشكلة الحقيبة ومشكلة البائع المتجول. يتم تقديم نوع جديد من العمليات النقطية القائمة على تفاعل أقصر مسافة, مع خوارزمية محاكاة لتحليل سلوكها. تقترح هذه الأطروحة نموذجًا مبتكرًا لإثراء النظرية الحالية وفتح آفاق جديدة للبحث.

الكلمات المفتاحية : العمليات النقطية، عمليات ماركوف النقطية ،عمليات التفاعل الزوجي ،مشكلة البائع المتجول،خوارزميات المحاكاة MCMC.

ABSTRACT

This thesis explores the intersection of point processes and optimization problems, given new perspectives in probability theory and spatial statistics. It combines solid theoretical foundations with advanced practical applications, examining various models of point processes such as Poisson and binomial processes, as well as classical combinatorial optimization problems like the knapsack and traveling salesman problems. A new type of point process based on the shortest distance interaction is introduced, with a simulation algorithm to analyze its behavior. This thesis proposes an innovative model to enrich existing theory and open new research avenues.

Keywords : Point processes, Markov point processes, Pairwise interaction processes, Traveling salesman problem, Simulation algorithms, MCMC method.

RÉSUMÉ

Ce mémoire explore l'intersection entre les processus ponctuels et les problèmes d'optimisation, offrant ainsi de nouvelles perspectives dans la théorie des probabilités et les statistiques spatiales. Il combine des fondements théoriques solides avec des applications pratiques avancées, en examinant divers modèles de processus ponctuels comme les processus de Poisson et binomiaux, ainsi que des problèmes classiques d'optimisation combinatoire tels que le problème du sac à dos et du voyageur de commerce. Un nouveau type de processus ponctuels basés sur l'interaction de la plus courte distance est introduit, avec un algorithme de simulation pour analyser son comportement. Ce mémoire propose un modèle innovant pour enrichir la théorie existante et ouvrir de nouvelles perspectives de recherche.

Mots clés : Processus ponctuels, Processus Ponctuels de Markov ,processus a interaction paire à paire, Problème du voyageur de commerce, Algorithmes de simulation par la méthode MCMC.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES ALGORITHMES

LISTE DES TABLEAUX

INTRODUCTION

Imaginez un monde où les événements ne suivent pas une logique linéaire, mais se dispersent aléatoirement comme des étoiles dans un ciel nocturne. Ce monde, où chaque événement est un point dans un espace, appartient aux processus ponctuels. Fascinants et complexes, les processus ponctuels se trouvent à l'intersection de la théorie des probabilités et des statistiques spatiales. Ils nous permettent de modéliser et de comprendre des phénomènes aussi variés que la répartition des arbres dans une forêt, l'apparition des étoiles dans le cosmos, ou encore les points de livraison dans une ville.

Ce mémoire vous invite à un voyage au coeur de ce domaine captivant. Dans le premier chapitre, nous poserons les bases en explorant les généralités sur les processus ponctuels. Nous commencerons par une introduction historique, retraçant l'évolution de ce concept depuis ses origines jusqu'à ses applications modernes. Nous définirons ensuite ce qu'est un processus ponctuel et examinerons les lois qui le régissent. Une section dédiée aux exemples, tels que les processus de Poisson et les processus binomiaux, illustrera la diversité et la richesse de ces modèles. Comme le souligne Anselin (1995 , [1]), les indicateurs locaux d'association spatiale jouent un rôle crucial dans l'analyse géographique des processus ponctuels.

Le deuxième chapitre nous transportera dans le domaine de la théorie des graphes et de la complexité algorithmique. Nous y découvrirons des notions fondamentales et des concepts clés, comme les classes de problèmes P et NP, et explorerons des problèmes d'optimisation combinatoire tels que le problème du sac à dos et le problème du voyageur de commerce. Ce dernier, véritable casse-tête mathématique, nous permettra de comprendre les défis et les stratégies pour optimiser des parcours et des réseaux. En s'appuyant sur les travaux de Barka (2015 , [5]) et de Bayou et Bensefia (2021 , [7]), nous examinerons les solutions exactes et approchées pour ces problèmes.

Notre exploration se poursuivra dans le troisième chapitre avec une étude approfondie des méthodes et algorithmes de résolution en optimisation combinatoire. Nous examinerons les méthodes exactes, telles que la méthode Branch and Bound et la méthode Branch and Cut, en détaillant leur fonctionnement, leur complexité et leurs avantages. Comme l'ont démontré Land et Doig (1960 ,[37]), la méthode Branch and Bound reste un pilier pour résoudre des problèmes de programmation discrète. Nous passerons ensuite aux méthodes approchées, comme les algorithmes de colonies de fourmis Dorigo (1992 , [24]), le recuit simulé Kirkpatrick (1983 ,[36]), et les algorithmes gloutons Dantzig (1954, [22]). En nous appuyant sur les travaux de Bendahmane (2011 ,[8]) et des analyses comparatives de Dorigo et Maniezzo (1991 , [15]), nous chercherons à comprendre les forces et les faiblesses de ces techniques dans la résolution de problèmes complexes. En particulier, nous examinerons comment le recuit simulé a été appliqué pour résoudre des problèmes tels que le voyageur de commerce, en s'inspirant des approches thermodynamiques comme celles décrites par Cerný (1985 ,[12]).

Le quatrième chapitre sera le point culminant de notre voyage, où nous introduirons et proposerons un nouveau type de processus ponctuel basé sur l'interaction de la plus courte distance. Ce processus novateur représente une avancée significative dans la modélisation des événements spatiaux, en intégrant des interactions complexes de manière plus réaliste. Nous définirons ce processus en détail et discuterons de ses propriétés uniques, en nous inspirant des travaux de Baddeley et Van Lieshout (1995, [3]) sur les processus d'interaction par zone. Ensuite, nous développerons un algorithme de simulation basé sur la méthode de Métropolis-Hastings, comme illustré par Chib et Greenberg (1995, [13]). Cette simulation nous permettra de visualiser et d'analyser le comportement de ce nouveau processus dans divers contextes, ouvrant ainsi de nouvelles perspectives de recherche et d'application. Les travaux de Daley et Vere-Jones (2003, [20]) sur la théorie des processus ponctuels fourniront un cadre théorique solide pour notre approche.

À travers ce mémoire, nous chercherons à offrir une vision holistique et approfondie des processus ponctuels, en passant des fondements théoriques aux applications pratiques les plus avancées. En intégrant des concepts issus des travaux de Baccelli et Brémaud (2003 ,[2]), ainsi que de Daley et Vere-Jones (2003, [20]), nous espérons ainsi contribuer à une meilleure compréhension de ces outils puissants et polyvalents, qui jouent un rôle crucial dans de nombreux domaines scientifiques et techniques. En explorant ce nouveau processus ponctuel, nous espérons non seulement enrichir la théorie existante, mais également fournir des outils pratiques pour des applications variées, allant de l'analyse spatiale à l'optimisation des réseaux.

CHAPITRE 1

GÉNÉRALITÉS SUR LES PROCESSUS PONCTUELS

1.1 Introduction

Les processus ponctuels offrent des modèles pour décrire les motifs irréguliers de points, et cette théorie a émergé en réponse à divers problèmes liés à la physique, la biologie et la théorie des files d'attente. Des pionniers tels que Palm (1943,[48]), Bartlett (1954,[6]),et Cox (1955,[17]) on contributes de manière significative à son développement initial. Pour approfondir l'histoire des différents modèles de processus ponctuels, des détails supplémentaires peuvent être consultés dans l'ouvrage de Guttorp et Thorarinsdottir (2012,[32]).

1.2 Définition d'un processus ponctuel

Le terme "processus ponctuel" est un concept utilisé dans plusieurs domaines, notamment les statistiques, les mathématiques et la physique. Dans un contexte statistique, les " processus ponctuels " font référence à des événements qui se produisent à des points spécifiques de l'espace ou du temps. En mathématiques, cela peut concerner le modèle de processus ponctuels d'opérations mathématiques effectuées sur des points particuliers dans l'espace mathématique. En physique, les processus ponctuels peuvent faire référence à des événements qui se produisent à des points spécifiques de l'espace ou du temps, tels que des réactions nucléaires ponctuelles.

Soit (Ω, A, P) un espace probabilisé. Cet espace caractérisera les aspects aléatoires des expréiences. Soit χ un ensemble non vide menu d'une tribu A, et cet ensemble servira de contenant.

Définition 1.2.1. [26] On appelle une configuration (Figure 1.1) tout ensemble dénombrable,

non ordonné de points : $x = (x_1, x_2, ..., x_n)$ o $x_i \in \mathbb{N}$ sont des points issus d'une expérience aléatoire .

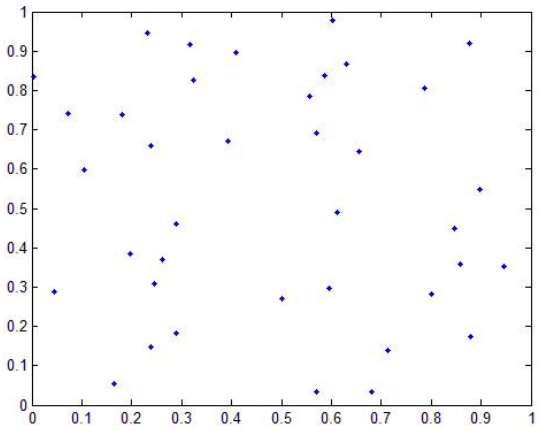

FIGURE 1.1 – Ensemble de points ou Configuration, de $\chi = [0,1]^2$.

Définition 1.2.2. [46] Une configuration $x \subseteq \chi$ est localement finie si dans tout borélien borné $A \subseteq \chi$ elle place un nombre $N_x(A)$ fini de points. La famille de toutes les configurations localement finies sera notée N^{lf}.
On définit alors la notion de processus ponctuel :

Définition 1.2.3. [46]Un processus ponctuel X est une application sur χ dans N^{lf}, telle que pour tout borélien $A \subseteq \chi$, $N(A) = N_x(A)$ est une variable aléatoire finie.
Ce qui induit une nouvelle définition :

Définition 1.2.4. [46]Si l'espace χ est borné ou si $N_x(A)$ est fini presque surement, le processus ponctuel est dit processus ponctuel localement fini.
Les réalisations d'un processus ponctuel X sont donc des configurations aléatoires de points tels que pour tout borélien $A \subseteq \chi$ le nombre de points dans A soit une variable aléatoire. Cela signifie qu'un processus ponctuel est une variable aléatoire à valeur dans l'espace mesurable (N^{lf}, N^{lf}), oú N^{lf} est la plus petite σ-algèbre telle que pour tout borélien borné $A \subseteq \chi$ l'application $X \to N_x(A)$ soit mesurable .
la mesure de probabilité induite sur N_{lf} est appelée la loi de X.

1.3 Loi d'un processus ponctuel

Définition 1.3.1. On appelle loi du processus ponctuel la mesure de probabilité π induite sur N^{lf}. La loi d'un processus ponctuel X devrait être la mesure image par l'application X de P

sur N^{lf} . Mais comme N^{lf} est définie par la mesurabilité des applications $X \to N_x(A)$ pour des boréliens $A \subseteq \chi$, l'analogue de la loi de probabilité pour des variables aléatoires dans un contexte de processus ponctuels est l'ensemble des lois jointes des vecteurs $(N(A_1), ..., N(A_m))$, où les A_i sont des boréliens bornés.

Définition 1.3.2. La famille de lois en dimensions finies (fidis) d'un processus ponctuel X sur un espace métrique (χ, d) complet et séparable est la collection des lois jointes $(N(A_1), ..., N(A_m))$ de pour tout vecteur fini $(A_1, ..., A_m)$ de boréliens bornés $A_i \subseteq \chi(\ i = 1, ..., m)$ de longueur quelconque $m \in \mathbb{N}$.
L'intérêt de cette définition est justifié par le théorème suivant [33] :

Théorème 1.3.1. *La loi d'un processus ponctuel X sur un espace métrique (χ, d) complet et séparable est entièrement déterminée par ses fidis.*
Donc, deux processus ponctuels partageant les mêmes fidis ont même loi.

1.3.1 La distribution d'un processus ponctuel

La distribution P d'un processus ponctuel X est déterminée par les probabilités définies comme suit :

$$\begin{aligned} P(Y) &= P(X \in Y) \\ &= P(\{\omega \in \Omega : X(\omega) \in Y\}) \quad \text{pour } Y \in N^{lf} \end{aligned}$$

Le terme $X \in Y$ signifie qu'il possède une certaine propriété, par exemple qu'il n'a pas de pont dans l'ensemble B. Alors $P(X \in Y)$ désigne la probabilité qu'il ait cette propriété. Les distributions de dimension finie sont d'une importance particulière. Elles ont des probabilités de la forme :

$$P(X(B_1) = n_1, ..., X(B_k) = n_k)$$

Si $B_1, ..., B_k$ sont des boréliens bornés et $n_1, ..., n_k \geq 0$, cette expression représente la probabilité qui a n_1 points à l'ensemble B_1, n_2 points à l'ensemble B_2 et ainsi de suite. La distribution de X sur l'intervalle$[\mathbb{N}, N^{lf}]$ est entièrement déterminée par le système de toutes ces valeurs pour tout $k = 1, 2, ...$ .En effet, elle est caractérisée par le sous-système où les ensembles sont manuellement disjoints. Pour plus de détails, voir [54].

1.3.2 Moments d'un processus ponctuel

Un processus ponctuel se réfère aux moments où des événements instantanés spécifiques se produisent, comme l'initiation de pannes au sein d'un ensemble de machines, les moments

où une particule ou un individu d'une population naît. Ce processus est caractérisé par une séquence croissante d'instants $t_1, t_2, ...$ où ces événements se produisent, par une série de points $P_1, P_2, ...$ sur l'échelle des temps, avec $P_n = t_n$, d'où son appellation de processus ponctuel (son détail est donné en [31]).

Pour ce type de processus, on peut associer un processus numérique X_t où X_t représente le nombre d'événements obtenus sur l'intervalle $[0, t]$. Cette fonction aléatoire constitue un cas particulier de processus numérique, caractérisé par une croissance par sauts unitaires à des instants aléatoires (discontinuités mobiles). Cependant, la nature spécifique de ce processus incite à le définir de manière plus appropriée en fonction des besoins.

Définition 1.3.3. Soit k intervalles disjoints $]t_i^{'}, t_i^{''}]$, avec $i = 1, 2, ..., k$ et soit N_i le nombre d'événements obtenus sur $]t_i^{'}, t_i^{''}]$ c'est-à-dire $N_i = X_{t_i^{'}}$ - $X_{t_i^{''}}$. On donne Pour tout k et chaque ensemble $\{t_i^{'}, t_i^{''}\}$ la loi jointe $\{N_1, N_2, ..., N_k\}$.

Définition 1.3.4. On définit $U_1 = t_1$ et $U_n = t_n - t_{n-1} = \overline{P_{n-1}P_n}$, la suite des lois conditionnelles des U_n connaissant $U_1, U_2, ..., U_{n-1}$ pour tout n entier positif .

1.4 Des exemples de processus ponctuels

1.4.1 Processus ponctuels simples

La distribution de épicentres peut présenter des multiples points dans le sens où deux épicentres ou plus peuvent être placés au même endroit. De multiples points seront discutés lors de la conjuration. Il est impossible d'avoir deux points ou plus dans la plupart des exemples pratiques souvent au même endroit parce que c'est physiquement impossible. Il est interdit d'avoir deux cellules différentes avec des centres confondus.

Plus précisément, notons N_s^{lf} l'ensemble des configurations x localement finies dont les points sont simples et soit l'ensemble $N_x(\{X\}) \in \{0, 1\}$ pour tout $x \in \chi$. Cet ensemble est N^{lf}-mesurable, car χ étant séparable, il existe une suite dense (x_i) de sorte qu'on puisse recouvrir χ par une intersection dénombrable de boules ouvertes $B(x_i, 2^{-j})$ de rayons arbitrairement petits et écrire N_s^{lf} sous la forme [26] :

$$\bigcap_{i\geq 0}\{\phi \in \Omega : N(B(x_i, 2^{-j})) \in \{0, 1\}\} \in N^{lf}\,.$$

Définition 1.4.1. Un processus ponctuel X est dit simple s'il prend ses valeurs dans N_s^{lf} presque surement.

Les processus ponctuels simples ont un avantage pratique. Ils ne nécessitent pas de connaitre une famille entière de fidis pour disposer de leur loi. En effet, il suffit seulement de connaitre les probabilités de vide $v(A) = \mathrm{P}(N(A) = 0)$ pour une collection suffisamment grande de boréliens A de χ .

Théorème 1.4.1. *La loi d'un processus simple X sur un espace séparable complet (χ, d) est uniquement déterminée par les probabilités de vide pour l'ensemble des boréliens bornés $A \subseteq \chi$.*

On peut aussi remplacer borélien par compact. Pour plus de détails, voir [38]

1.4.2 Processus ponctuels marqués

Nous allons aborder maintenant les aspects du processus ponctuel marqués. Cela peut être assez pratique lorsque l'on veut différencier la nature des points : magnitude, spectre d'émission des étoiles, sources de vie ou de mort, des cellules avec des signes radioactifs...etc.

Définition 1.4.2. Soit $(\chi\ ; d)$ et $(K\ ; d^{'})$ deux espaces métriques complets séparables. Un processus ponctuel marqué (Daley,[19]) à positions dans χ et à marques dans K est un processus ponctuel sur l'espace produit $\chi \times K$ de sorte que le processus ponctuel non marqué soit bien défini. Par processus non marqué, on entend le processus obtenu par les projections des points de chaque marque sur une marque de référence [33].
Sur la Figure (1.2), nous pouvons voir un exemple de processus marqué, processus ponctuel selon les marques $K = \mathbb{R}$, représenté par l'intermédiaire des différents plans, et processus ponctuel selon les points spatiaux de $\chi = \mathbb{R}^2$, représenté par les points sur chaque plan associé à une marque. Le processus non marqué résultant est obtenu par la projection des points spatiaux de chaque marque sur un plan de référence (plan défini par $\chi \times \{0\}$ ici). Afin de ne pas alourdir le schéma, les projections des différents points marqués ne sont pas toutes explicitées.

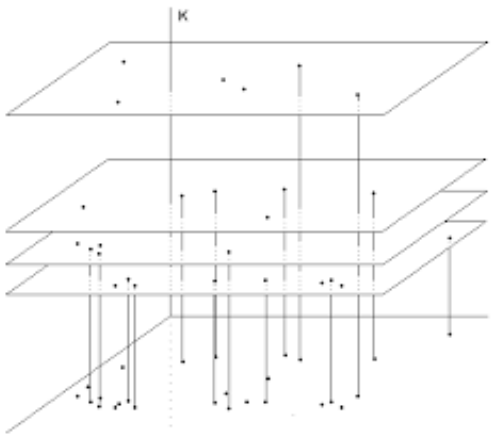

FIGURE 1.2 – Exemple d'un processus ponctuel marqué

1.4.3 Processus ponctuels finis

En plus d'être souvent simples, la plupart des configurations sont aussi finies. Cette finitude provient généralement de l'aspect borné de la fenêtre d'observation, mais peut aussi provenir du fait que le processus génère simplement un nombre fini de points.

Définition 1.4.3. Un processus ponctuel fini est défini comme un processus dont les configurations sont finies. Ces processus sont bien modélisés par [33] :

1. Une loi de probabilité discrète $(\mathrm{P}_n, n \in \mathbb{N})$, qui régit le nombre de points.
2. Une famille de densité de probabilité $j_n(x_1, x_2, ..., x_n)$ $n \in \mathbb{N}$, symétrique en ses variables, qui régissent la position des points sur χ^n .

Remarque 1. Nous supposons que χ est muni d'une mesure borélienne $v(.)$ de sorte que les densités j_n $(x_1, x_2, ..., x_n)$ soient définies par rapport à la mesure produite $v(.)^n$. De plus, l'aspect symétrique requis pour j_n provient du fait que les points générés par le processus ponctuel sont indifférents à l'ordre dans lequel on les indices. Ainsi, la densité j_n doit être la même quelle que soit la manière d'indicer les points de la configuration. (Pour plus de détails, voir[38]).

1.4.4 Processus ponctuel Binomial

Considérons χ comme un compact de $\mathbb{R}^d$ avec un volume strictement positif $\mu(\chi)$. Un processus ponctuel binomial $X = \{X_1, ..., X_n\}$ [46] est défini comme l'union d'un nombre fixé n de points indépendants et uniformément distribués $X_1, ..., X_n$. Comme $\mathrm{P}\ (X_i = X_j) = 0$ pour tout $i \neq j$, X est un processus simple. De plus, comme $\mathrm{P}\ (N(\chi) = n) = 1$, le processus est binomial et fini avec :

$$\mathrm{P}_m = \begin{cases} 0 & \text{si } m \neq n \\ 1 & \text{si } m = n \end{cases}$$

Les points sont uniformément distribués, donc :

$$j_n(X_1, ..., X_n) = \left(\frac{1}{\mu(\chi)}\right)^n$$

On peut observer que les j_n sont invariantes par permutation. Le processus binomial tire son nom du fait que pour tout ensemble borélien $A \subseteq \chi$:

$$N(A) = \sum_{i=1}^{i=n} 1_A(X_i)$$

suit une loi binomiale de paramètres n et $\mu(A)/\mu(\chi)$.

1.4.5 Processus ponctuel de Poisson

1.4.5.1 Processus de Poisson homogène

Le processus ponctuel de Poisson homogène présenté ici n'est qu'une généralisation du célèbre processus ponctuel de Poisson à une dimension.

Soit $B_n \subseteq \mathbb{R}^p$ une boule centrée à l'origine, avec un rayon tel que son volume $\mu(B_n)$ soit égal à λ, où λ est un réel strictement positif, P^n la loi d'un processus binomial à n points dans B_n . Soit A un borélien borné de χ et $m \in \mathbb{N}$ Comme $(B_n)_{n\in\mathbb{N}}$ est une suite croissante pour l'inclusion, dont l'union recouvre l'espace χ tout entier, on peut trouver un entier $n_0 \geq m$ tel que pour tout $n \geq n_0$, la boule B_n contient A. Nous avons alors [26] :

$$P^{(n)}\left(N(A) = m\right) = P^{(n)}\left(N(A) = m, N(B_n \backslash A) = n - m\right)$$

$$= C_m^n \left(\frac{\mu(A)}{\mu(B_n)}\right)^m \left(1 - \frac{\mu(A)}{\mu(B_n)}\right)^{n-m}$$

$$= C_m^n \left(\frac{\mu(A)}{\mu(B_n)}\right)^m \left(\frac{\mu(B_n \backslash A)}{\mu(B_n)}\right)^{n-m}$$

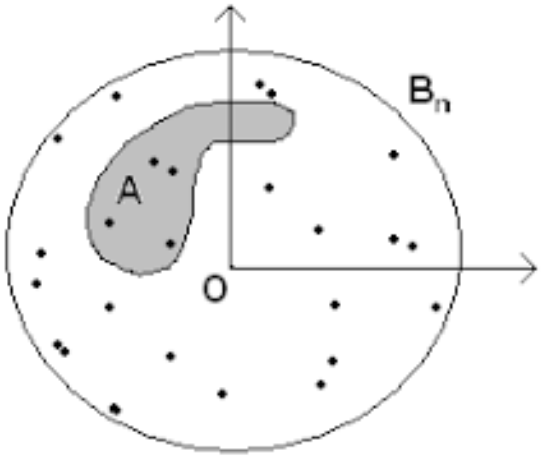

FIGURE 1.3 – $B_n \subset \mathbb{R}^p$ une boule centrée à l'origine

Définition 1.4.4. Un processus ponctuel X sur χ ($\subset \mathbb{R}^k$) est appelé un processus ponctuel de Poisson homogène d'intensité $\lambda > 0$ si :

1. Pour $(A_1, ..., A_m)$ boréliens disjoints sur χ, les variables aléatoires $N(A_1), ..., N(A_m)$ sont indépendantes.
2. Le nombre de points dans la configuration χ noté $N_x(A)$ suit une loi de Poisson de paramètre $\lambda\mu(A)$.

1.4.5.2 Processus de Poisson inhomogène

Le coefficient λ peut être interprété comme une densité si on décide de rendre ce coefficient variable $\lambda = \lambda(x), x \in \chi$. L'intensité devient variable selon l'endroit que l'on considère dans χ [26]. Donnons une définition pour préciser cette abstraction :

Définition 1.4.5. Un processus ponctuel sur χ est appelé un processus ponctuel de Poisson nonhomogène, s'il vérifie :

1. Pour $(A_1, ..., A_m)$ boréliens disjoints surχ, les variables aléatoires $N(A_1), ..., N(A_m)$ sont indépendantes.
2. Pour tout borélien borné A , $N_x(A)$ suit une loi de Poisson de paramètre $v(A)$. Où $\int_A \lambda d\mu$ et λ une fonction positive, Lebesgue mesurable sur χ. La fonction λ peut donc être vue comme la dérivée de Radon Nikodym de v par rapport à la mesure de Lebesgue μ.

1.4.5.3 Processus de Poisson

Le processus de Poisson[53] sur la droite est un processus à temps continu et à valeurs entières positives, également appelé un processus de comptage et noté $\{N(t) : t \geq 0\}$. Il étudie le nombre aléatoire $N(t)$ d'événements qui se produisent dans un intervalle de temps $[0, t]$. Sa grande popularité dans les applications vient du fait que de nombreux calculs explicites peuvent être effectués le concernant. Ce processus de comptage vérifie plusieurs propriétés avec des conditions précisées et a également plusieurs définitions [11].

Le processus de comptage $\{N(t) : t \geq 0\}$ est dit à accroissement stationnaire, si pour tout K, et pour toute suite $[I_1, ..., I_k]$ et toute translation, les lois de probabilité concèdent. Lorsque $k = 2$, on peut reformuler cette propriété de la manière suivante : si le processus de comptage est d'accroissement stationnaire, alors la loi de probabilité du nombre d'événements se produisant dans un intervalle de temps donné ne dépend que de la longueur de cet intervalle.

Un processus de comptage est dit à accroissements indépendants si le nombre d'événements se produisant dans des intervalles de temps disjoints sont indépendants.

En probabilité, on dit qu'un processus de comptage est localement continu si pour tout $t \geq 0$ on a :

$$\lim_{h \to 0} P\{N(t+h) - N(t) \geq 1\} = 0$$

Définition 1.4.6. Un processus de comptage est appelé processus de Poisson $\{N(t) : t \geq 0\}$ de densité $\lambda \geq 0$ si :

1. $N(0) = 0$.

2. Le processus est d'accroissements indépendants,
3. Le nombre de tops se produisant dans un intervalle de temps de longueurs $t \geq 0$, suit la loi de Poisson de paramètre λt. c'est-à-dire, pour tout $s \geq 0$ et tout $t \geq 0$ nous avons :

$$\mathrm{P}\{N(s+t) - N(t) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \quad (n \geq 0)$$

1.4.6 Processus ponctuel de Strauss

Le processus de (Strauss , [52, 35]) est un modèle stochastique utilisé pour décrire la distribution spatiale de points dans un espace donné. Il est souvent utilisé pour décrire des phénomènes tels que les particules dans la matière ou les galaxies dans l'univers. Le processus de Strauss repose sur l'idée de distribuer des points de manière aléatoire, mais il a également tendance à attirer ou à pousser en fonction de leur proximité les uns par rapport aux autres. Cette interaction entre points est contrôlée par deux paramètres : le facteur de densité, qui détermine la densité totale des points et le facteur d'impulsion qui mesure l'effet de la proximité sur l'attraction ou l'impulsion entre points.
Le processus de Strauss est un modèle d'interaction par paires où chaque paire contribue à la même fonction d'interaction .
La loi $\pi(.)$ de ce processus conditionnellement au nombre de points n est donnée par :

$$\pi(x) = k\gamma^{s(x)} \text{ où } 0 < \gamma \leq 1$$

Avec, γ est assimilé à un coefficient de répulsion, k une constante de normalisation et où la fonction $s(x)$ assimilée à un potentiel global d'énergie est définie par :

$$s(x) = s(x^1, ..., x^n) = \sum_i \sum_{i>j} 1_{\|x^i - x^j\| \leq R}$$

1.5 Les propriétés des processus ponctuels

1.5.1 Stationnarité

La stationnarité d'un processus ponctuel spatial se traduit par l'invariance de ses propriétés statistiques lors de déplacements dans l'espace. En d'autres termes, cela implique que des caractéristiques telles que la moyenne, la variance et la covariance des événements ne changent pas en fonction de la localisation spatiale.

Définition 1.5.1. Un processus ponctuel X sera dit stationnaire ou homogène si sa loi est invariante par translation.

$$\forall\ Y \in\ \ F\ , \forall x \in \mathbb{R}^d : \mathrm{P}(Y) = \mathrm{P}(Y_x)$$

ou $Y_x = \{\varphi_x = \sum_{x_i \in \varphi} \delta_{x_i + x}\ ,\ \varphi = \sum_{x_i} \delta_{x_i} \in Y\}$ est le translaté de Y le long de x .
Par conséquent $\quad P_x^!(Y_x) = P_0^!(Y)$.
Si X est stationnaire, alors $\forall D \in B_b^d$, $\Lambda_p(D) = \lambda_p v(D)$ ou λ_p est constante, dite **intensité** de P (nombre moyen de points de X par unité de volume) et v la mesure de Lebesgue sur $\mathbb{R}^d$.
Nous avons pour toute fonction h mesurable positive sur $\mathbb{R}^d \times \Omega$:

$$\int_\Omega \sum_{x \in \varphi} h(x, \varphi - \delta_x) P(d\varphi) = \lambda_p \int_{\mathbb{R}^d} \int_\Omega h(x, \varphi_x) P_0^!(d\varphi) v(dx)$$

1.5.2 Isotropie

Un processus ponctuel spatial est isotrope lorsque il ne présente pas de direction prilivégiée. Cela signifie que les caractéristiques statistiques du processus restent les mêmes quelle que soit l'orientation de l'espace. Par exemple, dans un processus de diffusion de particules dans un milieu homogène, la distribution spatiale des particules est la même dans toutes les directions. De même, dans un processus de répartition aléatoire d'événements, tels que des points de vente dans une ville, la densité de points est uniforme indépendamment de la direction dans laquelle on regarde. En résumé, l'isotropie d'un processus ponctuel spatial garantit que ses propriétés statistiques demeurent invariantes par rotation de l'espace.

Définition 1.5.2. Un processus ponctuel est isotrope si sa loi est invariante par rotation, pour chaque relation r dans $\mathbb{R}^d$ avec :

$$P(rY) = P(Y) \text{ avec } rY = \{rX = \sum_{x_i \in X} \sigma_{rx_i}\ ,\ X = \sum_{x_i \in X} \sigma_{x_i} \in Y\}.$$

1.5.3 Intensité

La mesure de l'intensité Λ de X est une caractéristique analogue à la moyenne dun échantillon aléatoire à valeur réelle. Sa définition est comme suit :

$$\Lambda(B) = E(X(B)) = \int X(B) P(dX)$$

Pour des ensembles Borélien B.

1.5.4 Propriété de Markov au sens de Ripley-Kelly

Dans un processus ponctuel spatial, la propriété de Markov est vérifiée lorsque la probabilité qu'un événement survienne à un moment donné dépend uniquement de l'état actuel du processus et non de son historique complet. Par exemple, dans le processus de diffusion de particules, la probabilité qu'une particule se déplace vers une certaine direction dépend uniquement de sa position actuelle et non des mouvements antérieurs de la particule. De même, dans un processus de marche aléatoire, la probabilité qu'une personne se déplace vers une certaine direction dépend uniquement de sa position actuelle et non des déplacements précédents. En résumé, les événements futurs sont indépendants des événements passés, étant donné l'état présent du processus [26].

Définition 1.5.3. Soit $\sim$ une relation binaire symétrique et réflexive sur χ. Deux points u et v sont voisins si $u \sim v$. Le voisinage de $A \subset \chi$ est donné par :

$$\partial(A) = \{u \in \chi \text{ et } u \notin A : \exists v \in A \text{ tel que } v \sim u\}.$$

Nous notons $\partial(u) = \partial u$ pour $u \in \chi$. La propriété de Markov au sens de Ripley-Kelly est la suivante :

Définition 1.5.4. Enonçons les critères de la propriété de Markov pour un processus ponctuel X par rapport à une relation de voisinage $\sim$. Pour toute configuration dans l'ensemble des configurations $x \in N^{lf}$ possibles, les critères suivants doivent être satisfaits :

1. Si la densité $f(x)$ est non nulle, alors la densité $f(y)$ est également non nulle pour toute sous-configuration y de x. Ce critère est appelé héréditaire.
2. Si la densité $f(x)$ est non nulle, alors le quotient :

$$\lambda(u, x) = \frac{f(x \cup \{u\})}{f(x)}$$

ne dépends que de u et $\partial u \cap x$, où $\partial u \cap x = \{v \in x : v \sim u\}$

Le quotient $\lambda(u, x)$, appelé intensité conditionnelle de Papangelou, représente la densité de probabilité qu'il y ait un point u sachant que x est réalisé ailleurs.
Ces critères expriment une propriété de Markov locale, où le comportement d'un point u par rapport à la configuration entière x ne dépend que de ses voisins proches dans cette configuration.

CHAPITRE 2

THÉORIE DES GRAPHES, COMPLEXITÉ ET PROBLÈMES D'ANALYSE COMBINATOIRE

2.1 Introduction

L'optimisation combinatoire est une branche très importante de la recherche opérationnelle et dans les mathématiques appliquées. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par la variété des applications pratiques et pouvant être formulée sous la forme d'un problème d'optimisation combinatoire. Les problèmes d'optimisation combinatoire sont souvent généralement difficiles à résoudre. ils appartiennent à la classe des problèmes NP-difficiles.

L'optimisation combinatoire minimise (ou maximise) la fonction objectif. Ayant atteint son plateau de productivité depuis quelques années, elle est considérée aujourd'hui moins hype que l'apprentissage automatique (Machine Learning) ou l'apprentissage profond (Deep Learning), mais elle l'était tout autant dans les années 1960-1970 avec des innovations algorithmiques comme la méthode de Branch-and-Bound (1960,[37]), et la résolution du problème du voyageur de commerce sur 48 villes (1954,[29]).

2.2 Notion de la théorie des graphes

La théorie des graphes est un outil très important de la recherche opérationnelle pour la résolution des problèmes d'optimisation comme le problème du voyageur de commerce et en particulier le problème de plus court chemin.

2.2.1 Historique

L'histoire de la théorie des graphes débute en 1736 quand Euler (le mathématicien Allemand) ,[9] démontra qu'il était impossible de traverser chacun des sept ponts de la ville russe de Königsberg exactement une fois et revenir au point de départ. Voir la Figure (2.1)

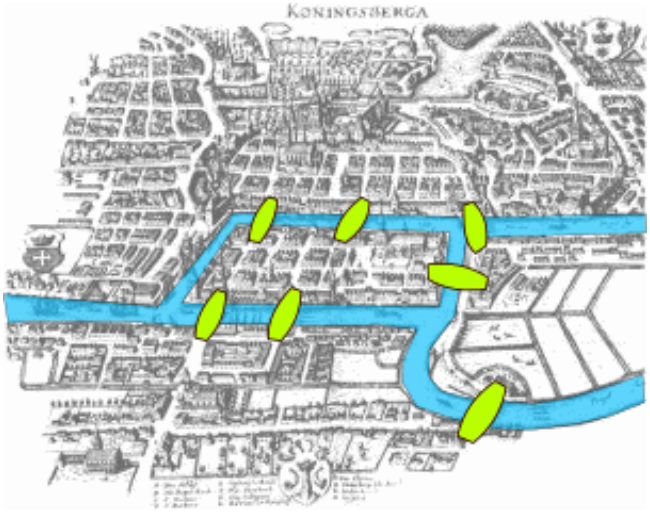

FIGURE 2.1 – Les sept ponts de Königsberg

2.2.2 Définition et notation

Un graphe G est défini par un couple $(X(G), E(G))$, oú $X(G)$ est un ensemble de sommets et $E(G)$ est un ensemble de paires de sommets appelées arêtes. Dans toute la suite, quand aucune confusion n'est possible, on écrira X et E au lieu de $X(G)$ et $E(G)$, respectivement.
Le nombre de sommets dans le graphe G est appelé ordre de G, est noté par $n = |X|$, et le nombre d'arêtes est appelé la taille de G, est noté par $m = |E|$. Un graphe est dit fini ou infini suivant son ordre.
soit G un graphe et soient x_i, x_j deux sommets de G. Une arête reliant deux sommets x_i, x_j est notée $x_i x_j$ au lieu de $\{x_i, x_j\}$. Si $x_i x_j \in E$, alors x_i et x_j sont dit adjacents et voisins.
Si $e = x_i x_j$ est une arête de G, alors x_i et x_j sont les extrémités de e, et e est dite incidente à x_i et x_j. Une boucle est une arête dont les extrémités sont confondues.

2.2.2.1 Graphe simple

Un graphe est dit simple s'il est sans boucles et sans arêtes multiples (i.e tout couple de sommets est relié par au plus une arête). voir la Figure (2.2).

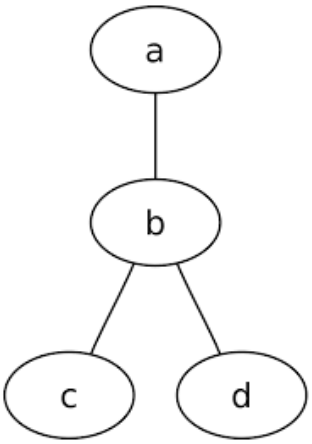

FIGURE 2.2 – Graphe simple

2.2.2.2 Graphe complet

Soit $G = (X, E)$ est complet si tous ses sommets sont adjacents.

On note par K_n un graphe complet dordre n, voir la figure (2.3).

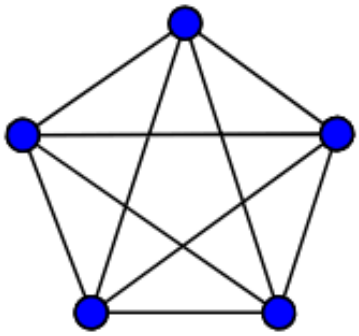

FIGURE 2.3 – Le Graphe Complet K_5

2.2.2.3 Graphe orienté

Soit $G = (X, E)$ un graphe orienté (voir la Figure (2.4)), tel que :

- X est un ensemble fini de sommets $\{x_1, x_2, ..., x_n\}$.
- E est un ensemble des couples ordonnés de sommets $(x_i, x_j) \in X^2$.
  Un couple (x_i, x_j) est appelé un arc, et est représenté graphiquement par $x_i \to x_j$ où x_i l'extrémité initiale , et x_j l'extrémité terminale, l'arc $u = (x_i, x_j)$ est dit sortant en x_i et incident en x_j, et x_j est un successeur de x_i, tandis que x_i est un prédécesseur de x_j .
  L'ensemble du successeur d'un sommet $x_i \in X$ est noté :

$$\Gamma^+(x_i) = \{x_j \in X, (x_i, x_j) \in E\}$$

$$\Gamma^-(x_i) = \{x_j \in X, (x_j, x_i) \in E\}$$

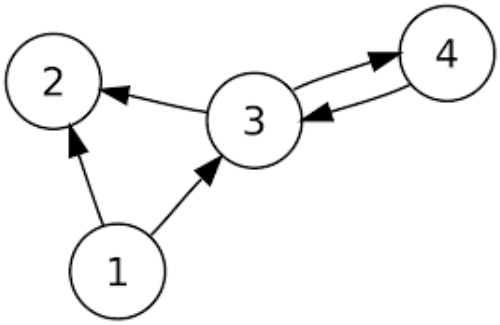

FIGURE 2.4 – Graphe orienté

2.2.2.4 Graphe non orienté

Un graphe G non orienté (Voir la Figure (2.5)) est la donnée d'un couple $G = (X, E)$. Tel que : X est un ensemble fini de sommets et E est un ensemble de couples non ordonnés de sommets $(x_i, x_j) \in X^2$. Une paire (x_i, x_j) est appelée une arête. On dit que les sommets x_i et x_j sont adjacents. L'ensemble des sommets adjacents au sommet $x_i \in X$ est noté : $N(x_i) = \{x_j \in X, (x_i, x_j) \in E\}$.

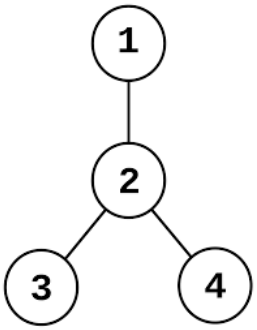

FIGURE 2.5 – Graphe non orienté

2.2.2.5 Graphe valué

Un graphe orienté (ou non orienté) dans lequel chaque arête est associée à un nombre réel, muni de l'application : $d : E \to \mathbb{R}_+^*$, appelée la fonction de coût, voir la Figure (2.6).

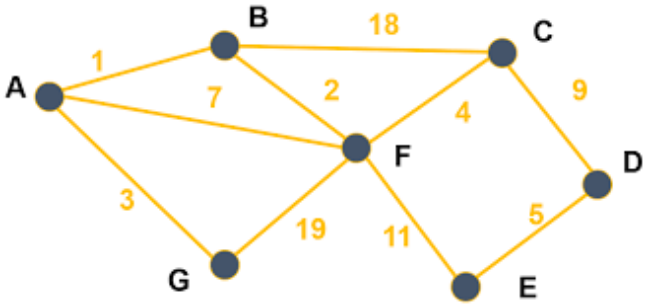

FIGURE 2.6 – Graphe valué

2.2.2.6 Circuit

Dans un graphe orienté, on appelle circuit une suite d'arcs consécutifs dont les deux sommets extrémités sont identiques.

2.2.2.7 Cycle

Un cycle est une chaîne de longueur > 1 simple et fermé. C'est donc une suite de la forme :

$$(x_0, e_1, x_1, \ldots, e_k, x_0)$$

où k est un entier > 1, les x_i et les e_j étant définis comme précédemment. L'entier k est la longueur du cycle.

Un cycle est élémentaire si les x_i pour $i = 0, \ldots, k-1$ sont distincts deux à deux. Tout cycle se décompose en cycles élémentaires deux à deux disjoints relativement aux arêtes.

2.2.2.8 Circuit hamiltonien

Un circuit ou cycle hamiltonien est un circuit (un cycle) qui passe une fois, et une seule fois, par chacun des sommets de G. On dit qu'un graphe G est hamiltonien s'il contient un cycle hamiltonien (cas non orienté) ou un circuit hamiltonien (cas orienté), voir la Figure (2.7).

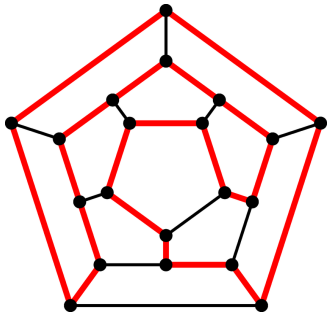

FIGURE 2.7 – Un graphe hamiltonien avec en rouge son cycle hamiltonien

2.2.2.9 Circuit absorbant

Soit C un circuit, C est appelé un circuit absorbant et $l(C)$ est sa longeur si : $l(C) = \sum_{u \in C} d(u) < 0$, voir la Figure(2.8).

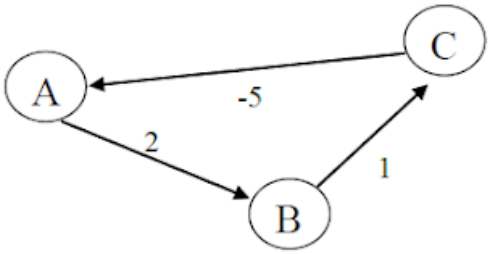

FIGURE 2.8 – Circuit absorbant

2.3 La théorie de la Complexité

La complexité d'un algorithme se définit comme le nombre d'instructions élémentaires utilisées pour résoudre un problème donné. Une instruction élémentaire peut être une affectation, une comparaison, une opération algébrique, une lecture ou une écriture. Cependant, il peut être difficile de compter précisément toutes les instructions et le nombre d'instructions peut varier entre deux exécutions du même algorithme avec des paramètres différents. Ainsi, on se contente généralement d'estimer un ordre de grandeur pour ce nombre d'instructions, ce qu'on appelle la complexité de l'algorithme. Pour évaluer la complexité temporelle d'un algorithme, on se concentre principalement sur les opérations les plus coûteuses, telles que les racines carrées, les logarithmes, les exponentielles, les additions réelles.

Définition 2.3.1. On dit que $f(n) = O(g(n))$ ($f(n)$ est de complexité $g(n)$), chaque fois qu'il existe k et n_0 tels que :

$$n > n_0 \implies f(n) < kg(n).$$

2.3.1 Classe P (Polynomial)

Un problème de décision est dans P s'il peut être décidé sur une machine déterministe en temps polynomial par rapport à la taille de la donnée. On qualifie alors le problème de polynomial, c'est un problème de complexité $O(nk)$ pour un certain k.

2.3.2 Classe NP (Non-déterministic Polynomial time)

C'est la classe de tous les problèmes de décision dont la solution peut être vérifiée en temps polynomial. Cela signifie que si une solution certifiée est fournie, il est possible de vérifier que cette solution est correcte en un temps polynomial par rapport à la taille de l'entrée.

2.3.3 Problème NP-complet

Un problème de décision σ est NP-complet s'il satisfait les deux conditions suivantes :

1. $\sigma \in NP$.
2. Tout problème NP se réduit à σ en temps polynomial.

Les problèmes NP-Complet sont liés par une relation d'équivalence dans le sens où s'il existe un algorithme polynomial pour un des problèmes de cette classe, alors tous les problèmes NP-Complet pourront être résolus en un temps polynomial. Cette classe englobe les problèmes les plus étudiés de l'optimisation combinatoire[44].

2.3.4 Problème NP-difficile

Un problème de NP est dit NP-difficile, si et seulement s'il existe un problème NP-complet qui'est réductible à lui en temps polynomial. De cette définition, on conclut que pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé à lui soit NP-complet[44].
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 (2.9)

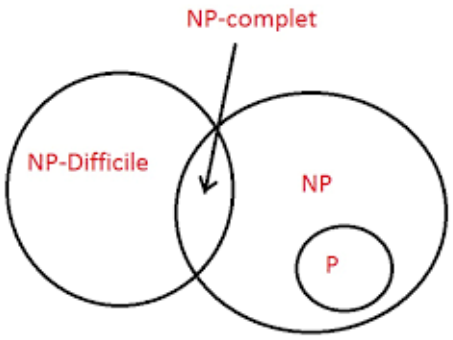

FIGURE 2.9 – La relation entre P , NP ,NP-complet et NP-dificille

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 pro- blème 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 $P = NP$, mais aucune preuve n'a encore été trouvée jusqu'à aujourd'hui.

2.4 Les problèmes d'optimisation combinatoire

Les problémes d'optimisation combinatoire consistent à trouver la meilleure solution parmi un ensemble fini de combinaisons possibles à l'aide des algorithmes d'optimisation pour trouver de meilleur solutions dans un temps raisonnable. Il y a plusieurs problèmes par exemple : l'ordonnancement, problème du sac à dos,l'emploi du temps, le problème de transport, le plus court chemin et le problème du voyageur de commerce...etc.

2.4.1 Problème du sac à dos

Le problème du sac à dos est l'un des 21 problèmes NP-Complets exposés par (Richard Karp ,[34]), également connu sous le nom de Knapsack Problem (KP), est un problème d'optimisation combinatoire largement étudié en informatique et en mathématiques. Il simule la situation à la quelle l'on doit remplir un sac à dos avec des objets de poids et de valeur différents, en maximisant la valeur totale des objets tout en respectant la capacité maximale du sac.

2.4.1.1 L'objectif de problème

L'objectif est de trouver la combinaison optimale d'objets à mettre dans le sac à dos, de manière à maximiser la valeur totale des objets, tout en ne dépassant pas la capacité de charge du sac. Ce problème est classé comme NP-complet, ce qui signifie qu'il est difficile à résoudre, surtout pour de grands ensembles d'objets.
Pour résoudre ce problème, plusieurs approches algorithmiques sont disponibles, notamment la programmation dynamique, les algorithmes gloutons et la programmation en nombres entiers. Ces méthodes sont utilisées dans divers contextes réels, tels que la planification de voyages, l'allocation de ressources et la gestion des stocks.

2.4.1.2 Modélisation de problème

la programmation linéaire (PL) est une branche des mathémaatique et plus précisément de l'optimisation dont l'objectif est minimiser ou maximiser une fonction numérique à plusieurs variables sachant que ces dernières sont liées par des relations appelées contraintes.
Un problème linéaire en nombre entiers (PLNE) est un problème d'optimisation dont toutes les variables sont entières.
La formulation PLNE du problème de sac à dos est très simple. On utilise pour chaque objet $i \in \{1, ..., n\}$, une variable binaire x_i correspond à 1 si l'objet i est pris dans le sac sinon 0.
Le problème du sac-à-dos est modélisé comme suit :

$$\begin{cases} Z = max \sum_{i=1}^{n} u_i x_i \\ \sum_{i=1}^{n} p_i x_i \leq C \\ x_i \in \{0,1\} \quad \forall i = \{1,...,n\} \end{cases}$$

u_i : valeur de l'objet i.

p_i : poids de l'objet i.

C : la capacité totale du sac.

2.4.2 Le problème de transport

Le problème de transport a été formalisé par le mathématicien français (Gaspard Monge en 1781,[43]).Il consiste à déplacer des marchandises ou des produits fabriqués depuis m origines (ou usines) vers n destinations (ou clients) de manière à minimiser le coût total de transport. Ainsi, résoudre un problème de transport revient à organiser le transport de manière à réduire au maximum le coût global du déplacement.

2.4.2.1 Modélisation de problème

Dans ce modèle, une entreprise dispose de m entrepôts et n points de vente, et un seul produit doit être expédié des entrepôts aux points de vente. Chaque entrepôt (origine) a un niveau d'approvisionnement donné, noté a_i, et chaque point de vente (destination) a un niveau de demande donné, noté bj. De plus, le coût de transport d'une unité du produit de l'entrepôt i à la destination j est représenté par c_{ij} unité. Telles que :

- La disponibilité de chaque entrepôt i est : a_i unité, ou $i = 1, 2, 3, ..., m$.
- La demande de chaque destination j est : b_j unité, ou $j = 1, 2, 3, ..., n$.
- le coût de transport d'une unité du produit de l'entrepôt i à la destination j est représenté par c_{ij} unité.

Où $i = (1, 2, 3, ..., m)$ et $j = (1, 2, 3, ..., n)$. Le coût total d'une expédition est linéaire en taille d'expédition.

Variables de décision Les variables de décision dans le modèle de programmation linéaire (PL) du problème de transport sont des entiers naturels représentant les unités transportées

d'une source vers une destination. Les variables de décision sont les suivantes : x_{ij} représentent la quantité a transporté de la source i vers la destination j, où $i = 1, 2, 3, ..., m$ et $j = 1, 2, 3, ..., n$[32].

Fonction objectif La fonction objectif du problème consiste à minimiser le coût total de transport, représentée par la somme des coûts de transport de chaque unité transportée de la source i à la destination j multipliée par la quantité x_{ij} associée. Ainsi, la fonction objective s'exprime comme suit :

$$Z = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Où la somme est prise sur toutes les sources i et toutes les destinations j.

Les contraintes Les contraintes du problème de transport garantissent que la disponibilité de chaque source est épuisée et que la demande de chaque destination est satisfaite. Elles se présentent comme suit :

1. Contrainte d'épuisement de la disponibilité à chaque source : $\sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \boldsymbol{x_{ij}} = \boldsymbol{a_i}$, pour tout i appartenant à l'ensemble des sources $\{1, 2, 3, ..., m\}$.
2. Contrainte de satisfaction de la demande à chaque destination : $\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{x_{ij}} = \boldsymbol{b_j}$, pour tout j appartenant à l'ensemble des destinations $\{1, 2, 3, ..., n\}$.
3. Non-négativité des quantités :$\boldsymbol{x_{ij}} \geq \boldsymbol{0}$, pour tout i appartenant à $\{1, ..., m\}$ et tout j appartenant à $\{1, ..., n\}$.

2.4.3 Le problème du voyageur de commerce

Le problème du voyageur de commerce est le problème NP-difficile le plus étudié. Sa formulation simple et ses multiples applications en font un problème largement connu.
Le problème du voyageur de commerce ne se limite pas uniquement aux villes et aux distances, il peut également être rencontré dans divers autres contextes ou être considéré comme un sous-problème.

FIGURE 2.10 – Problème du voyageur de commerce

2.4.3.1 Origine

Le problème de voyageur de commerce[29] a été examiné au 19^e siècle par William Rowan Hamilton, un mathématicien irlandais, ainsi que par Thomas Penyngton Kirkman, un mathématicien britannique. En 1859, Hamilton présente son travail sous la forme d'un jeu qu'il nomme ;"Icosian Game" dans une lettre.

FIGURE 2.11 – Icosian Game

2.4.3.2 Présentation du problème

Le problème du voyageur de commerce (PVC) consiste à trouver le chemin le plus court ou le plus économique permettant à un voyageur de visiter un ensemble donné de villes exactement une fois, en partant et en revenant à sa ville de départ. Chaque paire de villes est reliée par une distance, un temps de trajet ou un coût fixe. L'objectif principal du PVC est de minimiser la

distance totale parcourue par le voyageur tout en visitant toutes les villes précisément une fois. Pour représenter le problème du voyageur de commerce de manière pratique, il est bénéfique de le visualiser sous forme de graphe. Alors, soit $G = (X, E, d)$ un graphe pondéré non orienté dans lequel l'ensemble V des sommets représente les villes à visiter, ainsi que la ville de départ de la tournée, et E l'ensemble des arcs de G représente les parcours possibles entre les villes. Pour résoudre ce problème revient à trouver dans le graphe G un cycle hamiltonien passant par tous les sommets une et une seule fois, qui doit de longueur minimale.

2.4.3.3 Modélisation du problème

Il existe plusieurs modélisations du problème du voyageur de commerce dans le cas asymétrique (ou symétrique par l'application de la notion de dériver le graphe orienté du graphe non orienté), mais on va expliquer la modélisation de Dantzig en (1954, [22]).

La fonction objectif

$$Z = max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \quad i \neq j \tag{2.4.1}$$

La fonction objectif (2.4.1) est la somme des coûts unitaires des arcs pris dans le circuit.

Les contraintes

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., n \tag{2.4.2}$$

Ce groupe de contraintes (2.4.2) signifie qu'il y a un arc qui a le sommet i comme successeur.

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \tag{2.4.3}$$

Ce groupe de contraintes (2.4.3) signifie qu'il y a un arc qui a le sommet j comme successeur.

$$\sum_{i,j \in s} x_{ij} \leq |S| - 1, S \subset X, 2 \leq |S| \leq n - 2 \quad i \neq j \tag{2.4.4}$$

Ces groupes de contraintes (2.4.4) signifient que le graphe soit connexe.

$$x_{ij} \in \{0, 1\} \quad i, j = 1, ..., n. \quad i \neq j \tag{2.4.5}$$

Les variables $x_{ij} = \begin{cases} 0 & \text{si on prend pas l'arc } (i, j) \\ 1 & \text{sinon} \end{cases}$

2.4.4 Le problème du plus court chemin

La recherche de plus court chemin est un contexte fondamental de l'intelligence artificielle, relevant généralement du domaine de la planification et de la résolution de problèmes. Son objectif est de déterminer la meilleure manière de se déplacer dans un environnement, depuis un point initial jusqu'à un point final, en tenant compte de diverses contraintes.

Définition 2.4.1. Soit $R = (X, E, d)$ un réseau tel que $s \in X$ et $x \in X$.La fonction d représente la distance entre deux points dans le réseau, la longueur du plus court chemin entre s et x est appelée la plus petite distance de s à x .

Définition 2.4.2. La longueur du plus court chemin est égale à la somme des longueurs des arrêtes de C .

Définition 2.4.3. Soit $G = (X, E, d)$ un graphe orienté valué, un chemin C de sommet x à sommet y, est dit plus court chemin lorsque pour tout chemin C' dans un graphe G allant de x à y, nous avons $d(C') > d(C)$.

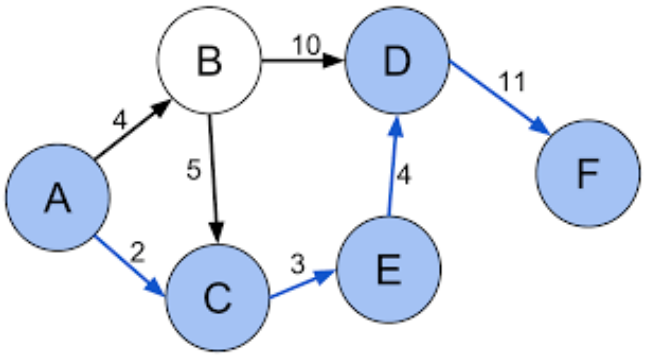

FIGURE 2.12 – Le plut court chemin

2.5 Méthodes de résolution d'un problème d'optimisation

Il existe deux méthodes de résolution : les méthodes exactes et les méthodes approchées.

Les méthodes exactes :permettent d'obtenir une solution optimale, mais dans un temps pouvant être trop long, si le problème est difficile à résoudre. Les méthodes les plus connues sont Branch and Bound, Branch and Cut.

Les méthodes approchées : encore appelées méta-heuristiques, permettent quant à elles dobtenir une solution approchée rapidement, mais qui n'est pas toujours optimale, sont des

algorithmes d'optimisation applicables à une grande variété des problèmes, elles sont généralement inspirées de la nature par exemple : colonie de fourmis, recuit simulé, essaim de particules colonie des abeilles.

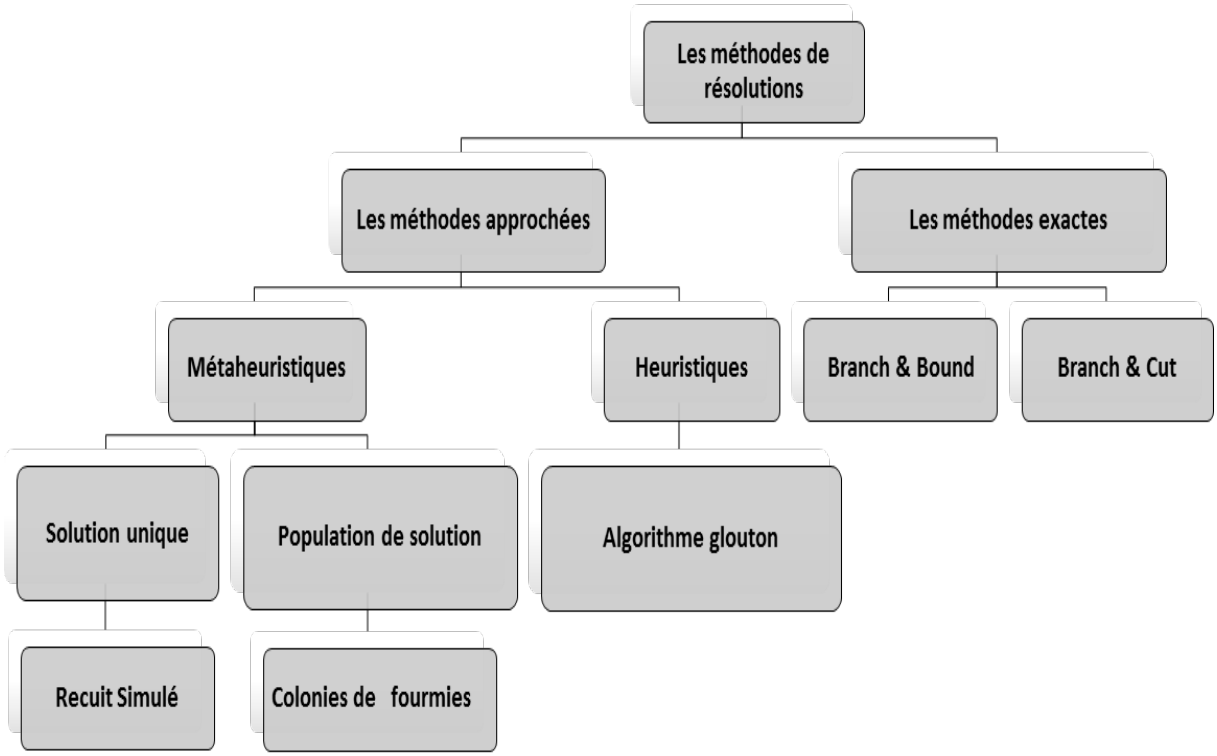

FIGURE 2.13 – Les méthodes de résolution des problèmes d'optimisation combinatoire

CHAPITRE 3

MÉTHODES ET ALGORITHMES POUR RÉSOUDRE LE PROBLÈME DU VOYAGEUR DE COMMERCE EN OPTIMISATION COMBINATOIRE

3.1 Introduction

La résolution des problèmes combinatoires est assez délicate puisque le nombre fini et/ou dénombrable et/ou infini de solutions réalisables croît généralement avec la taille du problème, ainsi que sa complexité. Cela a poussé les chercheurs à développer de nombreuses méthodes de résolution en recherche opérationnelle (RO). Ces approches de résolution peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées.

Les méthodes exactes gagnent en l'optimalité des solutions et perdent en temps d'exécution, ce qui est l'inverse pour les méthodes approchées qui ne garantissent pas de trouver une solution exacte, mais seulement une approximation en des temps raisonnables de calculs.

3.2 Les méthodes exactes

Les méthodes exactes sont des méthodes de résolution qui permettent d'obtenir la solution optimale à un problème d'optimisation en parcourant, de manière implicite, toutes les combinaisons possibles[39]. Parmi ces méthodes, on trouve les méthodes de séparation et d'évaluation (Branch and Bound) Land and Doig, (1960,[37]), la méthode de Branch and Cut Mitchell (2011, [41]).

3.2.1 La méthode Branch and Bound

L'algorithme de Branch and Bound (B&B), également appelé algorithme de séparation et évaluation, développé par Land et Doig en (1960, [37]), se fonde sur une approche arborescente pour rechercher une solution optimale. Il consiste à représenter les états solutions par un arbre d'états comprenant des noeuds et des feuilles. Les trois principaux axes du Branch and Bound sont les suivants :

1. **La séparation**

   La phase de séparation consiste à subdiviser le problème en sous-problèmes. En résolvant chaque sous-problème et en maintenant la meilleure solution parmi eux, on conservait la résolution du problème initial. Cette méthode revient à ériger un arbre qui explore toutes les solutions possibles. L'ensemble des noeuds restant à explorer dans cet arbre, susceptibles de contenir une solution optimale et donc nécessitant encore une division, est désigné comme l'ensemble des noeuds actifs.

2. **L'évaluation**

   L'évaluation vise à restreindre l'espace de recherche en éliminant certaines sous-sections qui ne peuvent pas contenir la solution optimale. Son objectif est de déterminer si l'exploration d'une sous-section de l'arborescence est prometteuse. Dans le cadre du branche-and-bound, l'élimination des branches dans l'arborescence de recherche fonctionne de la manière suivante : lors de la recherche d'une solution de coût minimal, on mémorise la solution de coût le plus bas rencontrée jusqu'à présent et on compare le coût de chaque noeud exploré à celui de la meilleure solution. Si le coût du noeud actuel dépasse le meilleur coût trouvé jusqu'à présent, l'exploration de cette branche est arrêtée, car toutes les solutions dans cette branche auront nécessairement un coût supérieur à la meilleure solution déjà trouvée.

3. **La stratégie de parcours**

   La stratégie de parcours désigne la manière dont l'arbre de recherche est exploré lors de l'application de méthodes de séparation et d'évaluation, telles que l'algorithme Branch and Bound. Trois stratégies principales sont couramment utilisées :

   - **La largeur d'abord :** Ce principe privilégie les sommets situés plus près de la racine, ce qui réduit le nombre de divisions du problème initial. Cependant, cette méthode est moins performante que les deux autres stratégies exposées.
   - **Profondeur d'abord :** cette stratégie favorise l'exploration des noeuds les plus éloignés de la racine, en appliquant davantage de séparations au problème initial.

Elle permet d'atteindre rapidement une solution optimale tout en économisant la mémoire, mais elle peut parfois s'enliser dans certaines branches de l'arbre.

- **La meilleure d'abord :** cette stratégie consiste à explorer en priorité les sous-problèmes ayant la meilleure borne. Elle permet d'éviter l'exploration de tous les sous-problèmes qui ont une mauvaise évaluation par rapport à la valeur optimale, ce qui peut accélérer le processus de recherche de la solution optimale.

Algorithm 3.1 Algorithme général de séparation et évaluation

Début d'initialisation avec le calcul d'une solution réalisable de valeur posée b_{best} ou posée $b_{best} = +\infty$.
Tant qu' il existe des noeuds non stérilisés. **Faire**
Chercher un noeud S tel que $b(S) < b_{best}$.
Si (S est une solution réalisable).

$b_{best} = b(S)$.
Sinon

Séparer S.
Fin si
Fin tant que
S est la solution optimale du problème.
Fin

3.2.1.1 Complexité de Branch and Bound

1. **Initialisation**

   Cette étape inclut la création de la solution initiale et le calcul de sa borne, ce qui peut être fait en $O(f(n))$, où $f(n)$ est la complexité de calcul de la borne.

2. **Exploration de l'arbre**

   La boucle principale consiste à explorer les noeuds de l'arbre jusqu'à ce que tous soient explorés ou qu'une solution optimale soit trouvée.

   - **Sélection du noeud à explorer**

     Choisir le prochain noeud à explorer (souvent le noeud avec la meilleure borne dans les méthodes de meilleure frontière). La Complexité dépend de la structure utilisée pour stocker les noeuds. Si une file de priorité est utilisée, la sélection peut être faite en $O(\log k)$, où k est le nombre de noeuds actifs.

   - **Borne inférieure pour le noeud actuel**

     Pour le calcul de la borne inférieure du noeud sélectionné, la complexité est $O(f(n))$.

- **Vérification de la solution**

  Vérifier si le noeud représente une solution complète et mise à jour de la meilleure solution trouvée. La vérification et la mise à jour peuvent être faites en $O(1)$ ou en $O(n)$, selon le problème.

- **Branching (expansion du noeud)**

  Génération des nuds enfants du noeud actuel. La Complexité dépend du nombre de branches b et de la complexité de la génération des enfants. Souvent, cela est $O(b \cdot g(n))$, où $g(n)$ est la complexité de génération d'un enfant.

- **Mise à jour des bornes et gestion des noeuds**

  Mise à jour des bornes des noeuds actifs et ajout des nouveaux noeuds enfants dans la structure de données. La complexité pour la mise à jour de la file de priorité peut être faite en $O(b \log k)$, où b est le nombre de nouveaux noeuds générés et k est le nombre de noeuds actifs.

3. **Critères d'arrêt**

   Arrêt de l'algorithme lorsque tous les noeuds sont explorés ou qu'une solution optimale est garantie.La complexité de cette vérification est constante ($O(1)$).

4. **Complexité totale**

   Pour estimer la complexité totale, nous devons prendre en compte les facteurs suivants :

   - N : Nombre total de noeuds générés et explorés.
   - b : Nombre moyen de branches par noeud.
   - $f(n)$: Complexité du calcul de la borne.
   - $g(n)$: Complexité de génération d'un enfant.
   - k : Nombre de noeuds actifs à tout moment (qui peut être proche de N dans le pire des cas).

La complexité totale de l'algorithme peut être exprimée comme :

$$O(N \cdot (f(n) + b \cdot g(n) + \log k))$$

En décomposant :

- $N \cdot f(n)$: Complexité pour calculer les bornes de tous les noeuds.
- $N \cdot b \cdot g(n)$: Complexité pour générer tous les enfants des noeuds.
- $N \cdot \log k$: Complexité pour gérer la structure de données des noeuds actifs.

Exemple 1. Pour un problème de type voyageur de commerce (TSP) :

- $f(n)$: Peut-être $O(n)$ pour calculer une borne comme la borne de Held-Karp.
- $g(n)$: Peut-être $O(1)$ pour générer un nouvel enfant (une nouvelle ville à visiter).
- b : Environ $n-1$ branches par noeud.

La complexité totale serait donc :

$$O(N \cdot (n + (n-1) \cdot 1 + \log N)) = O(N \cdot (n + \log N))$$

En résumé, la complexité du Branch and Bound dépend du nombre total de noeuds générés, de la complexité des calculs de borne, de la génération des enfants, et de la gestion des noeuds actifs. Cette complexité peut varier de polynomial à exponentiel en fonction de la structure de l'arbre de recherche et de l'efficacité des bornes utilisées.
La Figure (3.1) représente une estimation de la plus petite distance entre ensemble de 10 points obtenus par la méthode de branch and bound réalisée avec Python.

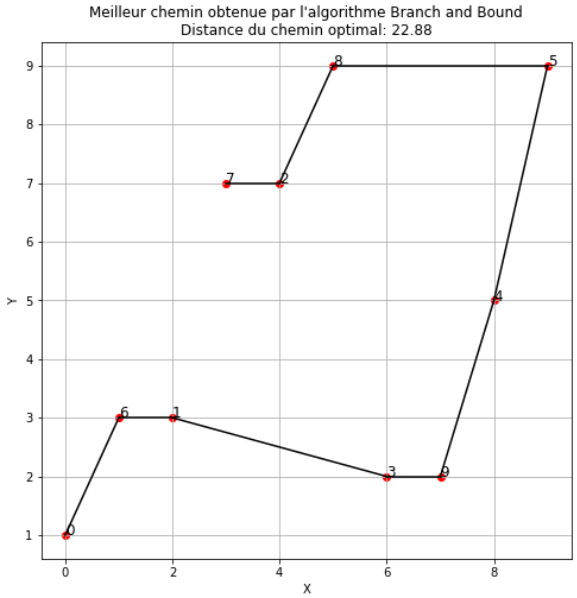

FIGURE 3.1 – Estimation de la plus courte distance entre un ensemble de 10 points obtenus par la méthode de Branch and Bound sur $\mathbb{R}^2$

3.2.2 La méthode de Branch and Cut

La méthode des coupes planes, bien qu'utile dans de nombreux cas, peut en effet rencontrer des limites lorsqu'elle est confrontée à des problèmes difficiles, notamment ceux impliquant des contraintes complexes ou des variables entières. De même, bien que l'algorithme du "Branch and Bound" soit efficace pour certains types de problèmes, il peut également être amélioré pour mieux gérer les problèmes de programmation linéaire en nombres entiers. C'est là qu'intervient

la méthode "Branch and Cut". Cette méthode combine les techniques du "Branch and Bound" et de la méthode des coupes planes pour résoudre les programmes linéaires en nombres entiers de manière plus efficace.

En résumé, la méthode "Branch and Cut" combine le branchement et la relaxation des variables entières avec l'application itérative de la méthode des coupes planes pour résoudre efficacement les programmes linéaires en nombres entiers, même dans les cas difficiles où les approches traditionnelles peuvent échouer.

Algorithm 3.2 Algorithme de Branch and Cut

/* On veut résoudre le problème d'optimisation combinatoire $min\ \{c^t x : Ax \geq b\,;\, x \in \mathbb{R}^n\}$ avec $A \in \mathbb{R}^{m \times n}$ et $b \in \mathbb{R}^m$ */

1. Liste des problèmes = $\varnothing$;
2. Initialiser le programme linéaire par le sous−problème de contraintes (A_1, b_1) avec $A_1 \in \mathbb{R}^{m1 \times n}$ et $b_1 \in \mathbb{R}^{m1}$ avec $m_1 << m$;
3. **Étapes d'évaluation d'un sous-problème**.
4. Calculer la solution optimale $\overline{x}$ du PL $c^t\ \overline{x} = min\{c^t x : A_1 x \geq b_1, x \in \mathbb{R}^n\}$;
5. Solution courante = Appliquer la méthode des coupes polyédrales() ;
6. **Fin étapes d'évaluation**
7. **Si** Solution courante est réalisable Alors
8. $x^* = \overline{x}$ est la solution optimale de $min\{c^t x : Ax \geq b, x \in \mathbb{R}^n\}$;
9. Sinon
10. Ajouter le problème dans Liste des sous-problèmes ;
11. **Fin si**
12. **Tant que** La liste des sous-problèmes $\neq \varnothing$ Faire
13. Sélectionner un sous-problème ;
14. Brancher le problème ;
15. Appliquer les étapes d'évaluation ;
16. **Fin Tant que**

3.2.2.1 Complexité de Branch and Cut

1. **Initialisation**
   - Liste des problèmes = $\varnothing$ ($O(1)$)
   - Initialiser le programme linéaire avec un sous-problème ($O(1)$)
2. **Étapes d'évaluation d'un sous-problème**

   Calculer la solution optimale $\overline{x}$ du PL : La complexité dépend de l'algorithme de résolution utilisé (par exemple, le Simplexe ou méthodes de points intérieurs). Supposons qu'il s'agit du Simplexe :

- Complexité en pire cas pour Simplexe est exponentielle, mais en moyenne, elle est polynomiale. Pour n variables et m_1 contraintes, la complexité moyenne est $O(m_1 n^2)$.
- Appliquer la méthode des coupes polyédrales : La complexité dépend du nombre de coupes ajoutées et du coût de génération des coupes.

  Supposons que générer et ajouter une coupe prend $O(f(n, m))$ temps. Si k coupes sont ajoutées, cela prend $O(kf(n, m))$.

3. **Vérification et ajout de sous-problèmes**
   - Vérifier si la solution est réalisable : $O(m)$ (vérification de toutes les contraintes)
   - Ajouter le problème dans la liste des sous-problèmes : $O(1)$
4. **Boucle principale**
   - Sélectionner un sous-problème (dépend de la stratégie de sélection) : O(1) pour une liste simple
   - Brancher le problème : La complexité dépend du nombre de sous-problèmes générés, en général, c'est constant pour chaque branchement (O(1)).
   - Appliquer les étapes d'évaluation : Complexité décrite ci-dessus
5. **Complexité globale**

   La complexité de l'algorithme dépend fortement du nombre total de sous-problèmes générés et de la complexité de résolution de chaque sous-problème.

   En moyenne, supposons que chaque sous-problème nécessite k coupes.
   - la complexité de la résolution de chaque sous-problème : $O(m_1 n^2 + kf(n, m) + m) \approx O(m_1 n^2 + kf(n, m))$
   - Multipliée par le nombre total de sous-problèmes P.

La complexité globale de l'algorithme peut être approximée par :

$$O(P(m_1 n^2 + kf(n, m)))$$

Où :

- P est le nombre total de sous-problèmes.
- m_1 est le nombre de contraintes dans le sous-problème initial.
- n est le nombre de variables.
- k est le nombre moyen de coupes ajoutées par sous-problème.

- $f(n, m)$ est la complexité de génération d'une coupe.

La complexité globale de l'algorithme de Branch-and-Cut est principalement influencée par le nombre total de sous-problèmes générés P et la complexité de la résolution de chaque sous-problème, qui dépend du nombre de coupes ajoutées et de la méthode utilisée pour résoudre les sous-problèmes linéaires. En pratique, bien que la complexité en pire cas puisse être exponentielle, les algorithmes de Branch-and-Cut sont souvent efficaces pour des instances de taille moyenne en raison des techniques heuristiques et d'optimisation utilisées pour réduire le nombre de sous-problèmes et améliorer la convergence.

La Figure (3.2) représente une estimation de la plus petite distance entre ensemble de 10 points obtenus par la méthode de branch and cut réalisée avec Python.

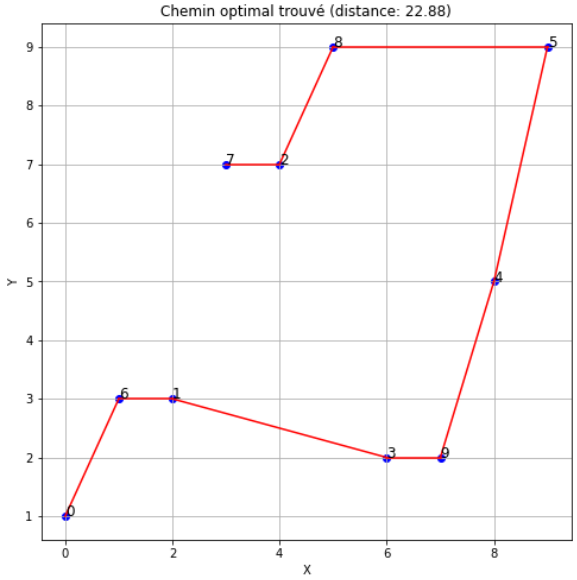

FIGURE 3.2 – Estimation de la plus courte distance entre un ensemble de 10 points obtenus par la méthode de branch and cut sur $\mathbb{R}^2$

3.3 Les méthodes Approchées

Les méthodes approximatives, bien qu'elles ne puissent pas garantir l'optimalité, sont souvent caractérisées par leur rapidité et leur utilisation plus efficiente des ressources. Ces approches sont fréquemment employées, notamment dans le contexte du problème du voyageur de commerce, afin de trouver des solutions de qualité dans des délais raisonnables. Ces méthodes approximatives se regroupent en deux catégories principales :

1. **Les heuristiques :** en optimisation combinatoire, une heuristique est un algorithme approximatif qui permet d'identifier rapidement au moins une solution réalisable, même si

elle n'est pas nécessairement optimale, en temps polynomial. Les heuristiques sont utiles pour obtenir rapidement une solution approchée à un problème, accélérant ainsi le processus de résolution exacte. Elles sont souvent conçues spécifiquement pour un problème donné, en exploitant sa structure propre, mais sans garantie quant à la qualité de la solution obtenue.

Les heuristiques peuvent être classées en deux catégories principales :

- **Les méthodes constructives :** qui génèrent des solutions à partir d'une solution initiale, en ajoutant progressivement des éléments jusqu'à obtenir une solution complète.
- **Les méthodes de recherche locale :** qui partent d'une solution complète initiale (peut-être moins intéressante) et tentent de l'améliorer en explorant les solutions voisines de manière répétée.

2. **Les méta-heuristiques :** Une méta-heuristique est un algorithme d'optimisation conçu pour résoudre des problèmes complexes d'optimisation pour lesquels aucune méthode classique n'est plus efficace. Ces problèmes sont souvent rencontrés dans des domaines tels que la recherche opérationnelle, l'ingénierie ou l'intelligence artificielle. Les méta-heuristiques sont généralement des algorithmes itératifs et stochastiques, qui progressent vers un optimum global en échantillonnant une fonction objectif. Elles agissent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques du problème pour trouver une approximation de la meilleure solution, de manière similaire aux algorithmes d'approximation.

   Les méta-heuristiques peuvent être classées en deux catégories :

   - **les méthodes basées sur une solution unique :** telles que la méthode de descente et la descente stochastique qui servent de base à des méta-heuristiques telles que le recuit simulé....etc.
   - **les méthodes basées sur une population de solutions :** telles que les algorithmes de colonies de fourmis, algorithmes d'essaim de particules...etc.

3.3.1 L'algorithme de colonies de fourmis

L'algorithme de colonies de fourmis est un concept méta-heuristique, connu sous le nom d'optimisation des colonies de fourmis (ACO) introduit par Marco Dorigo. En 1991,(Colorni, Dorigo et Maniezzo,[15]) ont présenté cet algorithme pour résoudre le problème du voyageur de

commerce. Cette approche a gagné en notoriété et a fait l'objet d'améliorations dès 1995 [24], avant d'être utilisée avec succès pour d'autres problèmes d'optimisation combinatoire dès 1994.

3.3.1.1 Le principe de colonies de fourmis

L'algorithme de colonies de fourmis est un algorithme inspiré de la nature qui appartient à la catégorie des algorithmes méta-heuristique, il est appelé "ant colony optimisation" en anglais. Cet algorithme vise à trouver le chemin optimal vers une source de nourriture utilisée par certaines espèces de fourmis. Les fourmis déposent au passage sur le sol une substance odorante appelée phéromone. Cette substance permet ainsi donc de créer une piste chimique, sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromone grâce à des capteurs sur leurs antennes. Ce comportement permet de trouver le chemin le plus court vers la nourriture lorsque les pistes de phéromone sont utilisées par la colonie entière. Autrement dit, lorsque plusieurs chemins marqués sont à la disposition d'une fourmi, cette dernière peut connaitre le chemin le plus court vers sa destination. Cette constatation essentielle est la base de toutes les méthodes que l'on va développer plus loin.

Algorithm 3.3 l'algorithme de colonies de fourmis pour (TSP)

Début

Initialisation α, β , η_{ij} , Q, t_0, $\tau_{ij}(t)$, ρ,$t = 0$

$\tau_{ij}(t)$ valeur générée aléatoirement

Tant que critère d'arrêt non satisfait, **faire**

pour chaque Fourmi $(k = 1, ..., m)$ **faire**

$x_k(0) = \varnothing$

Tant que le Sommet d'arrivée n'est pas atteint, **faire** sélectionner la prochaine ville restante selon :

$$P_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha . \eta_{ij}^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha . \eta_{il}^\beta} & \text{si } j \in N_i^k \\ 0 & \text{autrement} \end{cases}$$

ajouter au chemin $x_k(t)$

fait

En levant le circuit de $x_k(t)$ Si la liste n'est pas utilisée

Évaluer la longueur de chemin $L_k(t)$

fin pour

Évaporer les pistes selon :

$$\tau_{ij}(t + n) = (1 - \rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$t = t + n$

fin Tanque

Fin

Choix de transition : une fourmi k placée dans la ville i à l'instant t va choisir sa ville j de destination en fonction de la visibilité η_{ij} de cette ville et de la quantité de phéromones $\tau_{ij}(t)$ déposées sur l'arc reliant ces deux villes. Ce choix sera réalisé de manière aléatoire, avec une probabilité de choisir la ville j donnée par :

$$P_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha . \eta_{ij}^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha . \eta_{il}^\beta} & \text{si } j \in N_i^k \\ 0 & \text{autrement} \end{cases}$$

Où, α et β sont deux paramètres qui contrôlent l'importance relative entre phéromones et visibilité.

Mise à jour des phéromones La concentration des phéromones change avec le temps diminuant progressivement en raison de l'évaporation. les variables des phéromones sont mises à jour selon la formule :

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

Où $\rho \in [0,1[$ est un coefficient qui définira la vitesse d'évaporation des phéromones sur les arcs entre l'instant t et l'instant $(t+n)$, et $\Delta\tau_{ij}(t)$ représente la quantité de phéromone déposée par les fourmis dans ce même intervalle de temps sur l'arc (i,j). Le choix de ρ est important, en effet si ρ se rapproche trop de 1.

Quantité de phéromones déposées Soit $\Delta\tau_{ij}^k$, la quantité de phéromones déposée par cette fourmi sur larc (i,j) dans ce même intervalle de temps. On le définit ainsi :

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{L_k(t)} & \text{si } (u \in T_k(t) \wedge u(i,j)) \\ 0 & \text{autrement} \end{cases}$$

Où, $T_k(t)$ est le trajet effectué par la fourmi k à l'itération t, et Q est une constante. On voit bien ici que les phéromones sont régulées en fonction de la qualité de la solution obtenue, car plus $L_k(t)$ est faible plus l'arc sera mis à jour en phéromones. On peut maintenant définir le $\Delta\tau_{ij}^k$ (la quantité de phéromone déposée par chaque fourmi) de la formule de mise à jour des phéromones ainsi : $\Delta\tau_{ij}^k = \sum_{k=1}^{m} \Delta\tau_{ij}^k$

3.3.1.2 Complexité de l'algorithme de colonies de fourmis

Pour évaluer la complexité de l'algorithme, nous le divisons en cinq étapes distinctes, comme indiqué dans [16] :

1. Initialisation.
2. Un cycle de l'algorithme.
3. Équilibrage du cycle et calcul des dépôts de phéromones.
4. Dissipation des phéromones.
5. Boucle de l'algorithme.

Analysons les étapes de manière individuelle :

1. Pour la première étape, la complexité est de $O(|L|+m) = O(n^2+m)$ car nous supposons une interconnexion complète entre les villes.
2. Pour la deuxième étape, la complexité est de $O(n^2 \cdot m)$, car calculer la prochaine ville à visiter nécessite de passer en revue toutes les villes.
3. Pour la troisème étape, la complexité est de $O(|L| + m \cdot |L|) = O(m \cdot |L|) = O(n^2 \cdot m)$.

4. Pour la quatrième étape, la complexité est de $O(|L|) = O(n^2)$.
5. Pour la cinquième étape, le test de stagnation a une complexité de $O(n \cdot m)$ car nous comparons les parcours de m fourmis, chaque parcours contenant n éléments.

La complexité générale est calculée en additionnant la complexité de l'étape 1 à celle des étapes 2 à 5, multipliée par le nombre total de cycles (NC_{max}). La complexité globale maximale est donc $O(n^2 * m)$. En résumé, la complexité totale de l'algorithme est : $O(n^2 + m + NC_{max} \cdot n^2 \cdot m)$
Soit : $AScomplexity = O(NC_{max} \cdot n^2 \cdot m)$.
Cependant, cette formule ne fournit pas d'informations sur le nombre d'étapes nécessaires pour atteindre l'optimum, qui pourrait être atteint bien avant d'atteindre NC_{max} cycles.

3.3.1.3 Les avantages et les inconvénients de colonies de fourmis

Parmi les avantages de la méthode des colonies de fourmis, on peut citer :

- Efficace pour le problème du voyageur de commerce (TSP) et des problèmes similaires.
- Peut-être utilisé dans des applications dynamiques.
- Cela présente un intérêt dans le routage des réseaux et les systèmes de transport urbain.

Et, les inconvénients de cette méthode sont :

- L'analyse théorique est difficile.
- La distribution de probabilité change à chaque itération.
- La recherche est expérimentale plutôt que théorique.

La Figure (3.3) représente une estimation de la plus petite distance entre un ensemble de 10 points obtenus par l'algorithme de colonies de fourmis réalisée avec Python.

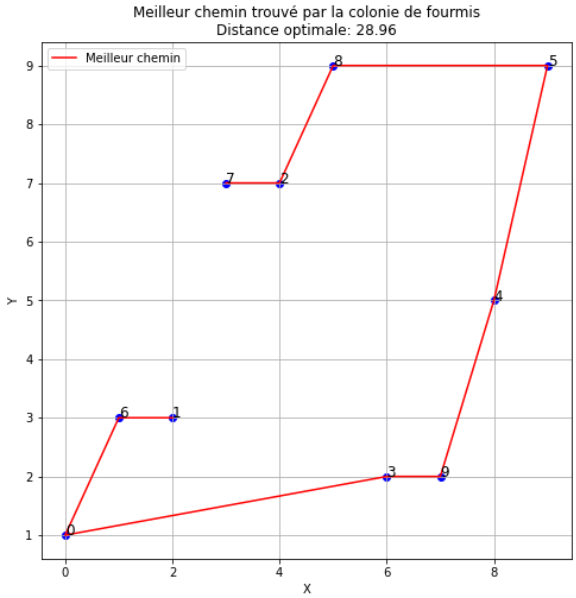

FIGURE 3.3 – Estimation de la plus courte distance entre un ensemble de 10 points obtenus par l'algorithme de colonies de fourmis sur $\mathbb{R}^2$

3.3.2 Algorithme du recuit simulé

La méthode du recuit simulé est inspirée de la méthode Monte-Carlo et vise à trouver une solution optimale à un problème donné. Elle a été élaborée par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi (1983,[36]), et de manière indépendante par V. Cerny (1985 ,[12]), à partir de l'algorithme de Metropolis, qui décrit l'évolution d'un système thermodynamique.

3.3.2.1 Le principe de recuit simulé

L'idée principale du recuit simulé est d'explorer le voisinage des solutions actuelles en acceptant parfois des solutions de qualité moindre. Cela permet d'éviter de rester piégé dans des optima locaux et d'explorer plus largement l'espace des solutions.
Au début de l'algorithme, un paramètre de température T est défini, puis décroît progressivement au cours de l'exécution de l'algorithme jusqu'à tendre vers zéro. La probabilité d'acceptation des solutions dégradées dépend de la valeur de ce paramètre, et cette probabilité diminue avec la diminution de la température.
L'intérêt du recuit simulé réside dans sa capacité à converger vers de bonnes solutions par rapport aux algorithmes de recherche classiques, même pour des problèmes complexes et difficiles.
Le principe du recuit simulé consiste en un processus itératif qui cherche des configurations présentant un coût plus faible tout en acceptant de manière contrôlée des configurations qui

dégradent la fonction de coût. Une méthode stochastique est utilisée pour générer une suite d'états successifs du système à partir d'un état initial donné. Chaque nouvel état est obtenu en appliquant une perturbation aléatoire au système.

Algorithm 3.4 Algorithme de recuit simulé

Initialisation

1. Choisir T_0 , T_f , N_{max}.
2. Générer aléatoirement une solution admissible initiale s_0.
3. Calculer $f_0 = f(s_0)$, $s^* \leftarrow s$, $f^* \leftarrow f_0$.
4. $i \leftarrow 1$.

Étape 1

1. Choisir $s_1 \in$ Rand $(V(s_0))$.
2. Calculer $\Delta f = f(s_1) - f(s_0)$.
3. Si $\Delta f \leq 0$ aller en étape 3.
4. Sinon aller en étape 2.

Étape 2

1. Choisir une variable aléatoire $\theta \in [0, 1]$ et calculer $\rho(\Delta f) = e^{-\Delta f/T}$.
2. Si $\theta < \rho(\Delta f)$ aller en étape 3.
3. Sinon, rejeter cette solution et aller en étape 4.

Étape 3

1. accepter la solution.
2. Poser $s_0 \leftarrow s_1$, $f_0 \leftarrow f_1$.
3. Aller en étape 4.

Étape 4

1. Si $i > N_{max}$ stop.
2. Sinon diminuer la température en utilisant $T_{i+1} = \gamma * T_i$.
3. Poser $i \leftarrow i + 1$ aller en étape 1.

3.3.2.2 Complexité de l'algorithme du recuit simulé

La complexité de L'algorithme de recuit simulé dépend de plusieurs facteurs : la fonction d'évaluation, le nombre d'itérations, la méthode de refroidissement et la génération des voisins. Voici une analyse détaillée de la complexité par étapes :

1. **Initialisation**

   Initialisation de la solution initiale et des paramètres (température initiale T_0 , facteur de refroidissement α, nombre d'itérations). La complexité de cette étape est généralement constante, $O(1)$.

2. **Évaluation de la solution initiale**

Calcul de la fonction d'évaluation pour la solution initiale. La complexité est $O(f(n))$, où n est la taille de l'instance du problème.

3. **Boucle principale (recuit)**

   La boucle principale consiste en plusieurs étapes internes répétées pour un nombre donné d'itérations ou jusqu'à ce que la température atteigne un certain seuil.

   - **Génération d'une solution voisine**

     La complexité dépend de la méthode de génération des voisins et est généralement $O(g(n))$.

   - **Évaluation de la solution voisine**

     Calcul de la fonction d'évaluation pour la solution voisine. Alors la complexité est similaire à l'évaluation de la solution initiale, $O(f(n))$.

   - **Acceptation de la nouvelle solution**

     Décision d'accepter ou de rejeter la nouvelle solution en fonction de la différence d'énergie et de la température actuelle. Alors la complexité est une opération constante $O(1)$.

   - **Mise à jour de la température**

     Mise à jour de la température en fonction de la méthode de refroidissement. Donc la complexité est une opération constante $O(1)$.

4. **Convergence**

   Critère d'arrêt basé sur la température ou le nombre d'itérations. Donc la complexité est une opération constante $O(1)$.

Complexité totale

Pour une analyse complète, nous considérons k comme le nombre total d'itérations (qui peut être fixé ou dépendre de la méthode de refroidissement et du critère de convergence).
La complexité totale de l'algorithme peut être estimée comme suit :

$$O(k \cdot (g(n) + 2f \cdot (n)))$$

En décomposant :

- $k \cdot g(n)$: Complexité pour générer k solutions voisines.
- $2k \cdot f(n)$: Complexité pour évaluer k solutions initiales et k solutions voisines.

Finalement :

$$O(k \cdot (g(n) + f(n)))$$

En fonction du problème spécifique et de la méthode utilisée pour générer les voisins, cette expression peut se simplifier ou nécessiter des ajustements.

Exemple 2. Pour un problème de type voyageur de commerce (TSP) :

- $f(n)$: Peut-être $O(n)$ pour évaluer une solution (par exemple, la longueur totale du chemin).
- $g(n)$: Peut-être $O(1)$ si on change simplement deux villes de place.

La complexité totale serait donc :

$$O(k \cdot (1 + n)) = O(k \cdot n)$$

En résumé, la complexité du recuit simulé est principalement déterminée par le nombre d'itérations k et la complexité de la fonction d'évaluation $f(n)$ et de la génération des voisins $g(n)$.

3.3.2.3 Les avantages et les inconvénients du recuit simulé

Parmi les avantages du recuit simulé, on peut citer :

- Fournit généralement de bonnes solutions par rapport aux algorithmes de recherche classiques : Le recuit simulé est efficace pour trouver des solutions de qualité raisonnable, même dans des cas où les algorithmes de recherche classiques peuvent être moins performants.
- Le recuit simulé peut être utilisé dans la plupart des problèmes d'optimisation.
- Il a la capacité de converger vers un optimum global lorsque le nombre d'itérations tend vers l'infini.

Et pour les inconvénients, on peut citer :

- Piégé dans un minimum local à basse de température.
- Difficulté de déterminer la température initiale.
- Impossibilité de savoir si la solution trouvée est optimale.

3.3.2.4 Critère d'arrêt

Le critère d'arrêt dépend du temps (nombre d'itérations) et de la dégradation de la solution (Δf).

La Figure (3.4) représente une estimation de la plus petite distance entre un ensemble de 10 points obtenus par l'algorithme du recuit simulé réalisée avec Python.

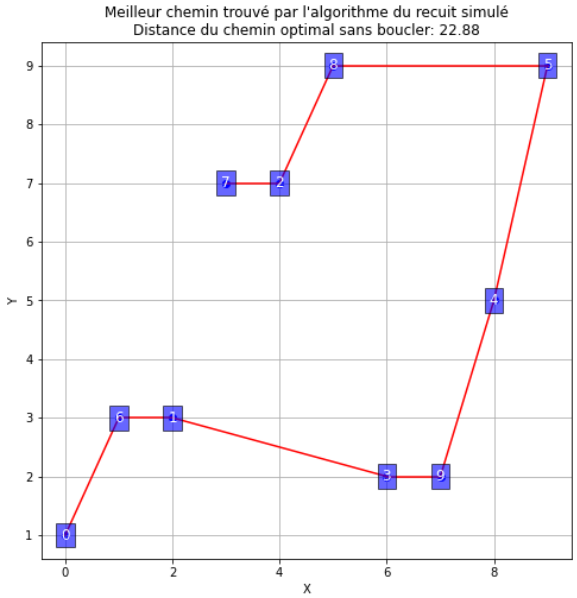

FIGURE 3.4 – Estimation de la plus courte distance entre un ensemble de 10 points obtenus par l'algorithme du recuit simulé sur $\mathbb{R}^2$

3.3.3 Algorithme glouton

Un algorithme glouton (greedy algorithme en anglais, également connu sous les noms d'algorithme gourmand ou goulu) est une méthode qui applique, étape par étape, le meilleur choix local possible dans le but d'obtenir un résultat optimal global. À chaque décision, l'algorithme résout le sous-problème résultant sans revenir en arrière sur ses choix. Il sélectionne à chaque étape l'option qui semble la plus avantageuse sur le moment, avec l'espoir d'atteindre une solution optimale. Toutefois, dans certains cas, la solution obtenue n'est pas optimale, et on parle alors d'heuristique gloutonne. Si la solution est optimale, l'algorithme est qualifié de glouton exact.

3.3.3.1 Principe général d'une méthode gloutonne

On est souvent dans le cadre de problèmes d'optimisation. Plus spécifiquement, le cas le plus fréquent est le suivant :

- On a un ensemble fini d'éléments, E.
- Une solution à notre problème est construite à partir des éléments de E : c'est par exemple une partie de E ou un multi-ensemble d'éléments de E ou une suite finie d'éléments de E ou une permutation de E qui satisfait une certaine contrainte.

- À chaque solution s est associée une fonction objectif $f(s)$: on cherche donc une solution qui maximise cette fonction objectif.

3.3.3.2 Complexité des algorithmes gloutons

Dans la complexité d'un algorithme glouton, lorsqu'il s'agit de trier un ensemble E de n éléments puis de vérifier une contrainte et d'ajouter des éléments en fonction de cette contrainte, nous pouvons décomposer les étapes de l'algorithme et analyser la complexité de chacune d'elles. Ensuite, nous combinons ces complexités pour obtenir la complexité totale de l'algorithme.

- **Étape 1 : Tri des éléments**

  Supposons que nous utilisions un algorithme de tri efficace comme le tri rapide (Quick Sort) ou le tri fusion (Merge Sort), dont la complexité temporelle est $O(n\ log\ n)$. Cela signifie que pour trier les n éléments de l'ensemble E, le temps requis est de l'ordre de $O(n\ log\ n)$.

- **Étape 2 : Vérification de la contrainte et ajout d'un élément**

  Après avoir trié les éléments, nous devons vérifier une contrainte pour chaque élément et éventuellement ajouter cet élément à une solution. Supposons que le coût de la vérification de la contrainte et de l'ajout d'un élément soit une fonction $(f(n))$.

- **Étape 3 : Analyse de la boucle**

  Étant donné que nous devons vérifier la contrainte et potentiellement ajouter chaque élément individuellement, cette vérification se fait pour chacun des n éléments. Ainsi, le coût total de cette étape est : $n \times f(n)$

Complexité totale

La complexité totale de l'algorithme est la somme des complexités des deux étapes principales : le tri et la vérification. Nous avons :

1. Complexité du tri : $O(n\ log\ n)$
2. Complexité de la vérification et ajout pour tous les éléments : $O(n \times f(n))$

En combinant ces deux contributions, nous obtenons :

$$O(n\ log\ n) + O(n \times f(n))$$

Comme la notation Big-O permet de combiner les termes en addition, la complexité totale de l'algorithme est de :

$$O(n\ log\ n + n \times f(n))$$

Les algorithmes gloutons sont donc généralement efficaces en termes de temps de calcul, car ils décomposent le problème en étapes simples et gérables.

Algorithm 3.5 Algorithme glouton

Trier(E)(en ordre décroissant) ;
Init(S) ;
i$\leftarrow 1$;
Tanque$(i \leq n)$**faire**
$S[i] \leftarrow S$ div $E[i]$;
$S \leftarrow S$ mod $E[i]$;
$i++$; **Fin tant que**

3.3.3.3 Avantages et inconvénients

Les algorithmes gloutons sont appréciés pour leur simplicité dans la construction d'une solution, généralement dans un délai raisonnable. L'inconvénient de cette méthode est qu'il faut souvent prouver l'optimalité de la solution ainsi obtenue, si elle existe. Les algorithmes gloutons ne garantissent pas toujours des solutions optimales, mais ils sont puissants et fonctionnent bien pour différents types de problèmes.

La Figure (3.5) représente une estimation de la plus petite distance entre ensemble de 10 points obtenue par l'algorithme glouton réalisé avec Python.

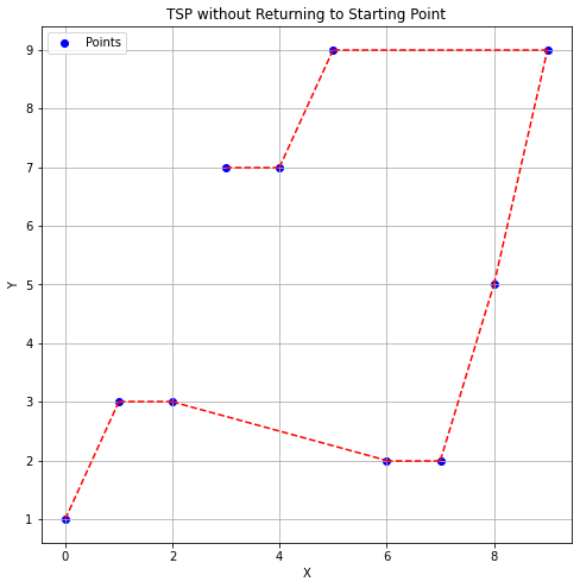

FIGURE 3.5 – Estimation de la plus courte distance entre un ensemble de 10 points obtenus par l'algorithme glouton sur $\mathbb{R}^2$

3.4 Etude comparative entre les algorithmes Approchées

Dans cette section, nous avons réalisé une étude comparative approfondie des résultats de trois algorithmes précédemment considérés. Ces résultats ont été obtenus après plusieurs tests en Python sur le problème du voyageur de commerce, pour chaque algorithme (recuit simulé, colonies de fourmis et algorithme glouton). L'objectif de cette étude est de déterminer quel algorithme offre les meilleurs résultats et une efficacité de recherche optimale dans un laps de temps réduit.

TABLE 3.1 – Tableau comparant entre Les méthodes approchées et La résultat obtenus par Les méthodes exactes pour N=10

les méthodes de résolution	La longueur de chemin optimale	le temps dexécution (secondes)
Recuit simulé	22.88	2.032444953918457
Colonies de Fourmis	28.96	80.04433035850525
Algorithme glouton	22.88062486640328	0.005976200103759766
Branch & Bound	22.88062486640328	12.07601022720337
Branch & Cut	22.88062486640328	46.87531924247742

Les résultats du tableau (3.1) démontrent que l'algorithme glouton est le plus rapide et le plus optimal, suivi par l'algorithme de recuit simulé, puis par l'algorithme des colonies de fourmis.

TABLE 3.2 – Tableau comparant entre Les méthodes approchées et Les résultats obtenus par Les méthodes exactes pour N=50

les méthodes de résolution	La longueur de chemin optimale	le temps d'exécution (secondes)
Recuit simulé	75,74	7,617201328277588
Colonies de Fourmis	56,95	3795,393703699112
Algorithme glouton	61,87911039658598	0,7030801773071289
Branch&Bound	Pas d'information	illimité
Branch&Cut	Pas d'information	illimité

L'augmentation du nombre de points dans le tableau (3.2) confirme que l'algorithme des colonies de fourmis nous a donné les meilleurs résultats, tandis que l'algorithme glouton est resté le plus rapide.

TABLE 3.3 – Tableau comparant entre Les méthodes approchées et Les résultats obtenus par Les méthodes exactes pour N=100

les méthodes de résolution	La longueur de chemin optimale	le temps d'exécution (secondes)
Recuit simulé	1830,23	16,733082056045532
Colonies de Fourmis	939,35	6324,1771836280823
Algorithme glouton	1023,0163402144152	4,168710708618164
Branch&Bound	Pas d'information	illimité
Branch&Cut	Pas d'information	illimité

Les résultats du tableau (3.3) montrent que l'algorithme glouton est la plus rapide pour trouver le chemin optimale.
Donc quelle que soit la taille du problème l'algorithme glouton reste la plus rapide et l'optimalité de distance dépend du nombre d'itérations et de la taille de problème.

3.5 Conclusion

Dans ce chapitre, nous avons mené une étude comparative pour évaluer l'efficacité des algorithmes méta-heuristiques (colonies de fourmis, recuit simulé et algorithme glouton) sur le problème du voyageur de commerce en modifiant la taille du problème. Nous avons enregistré la longueur du chemin optimal ainsi que le temps d'exécution de chaque algorithme. Cette comparaison met en évidence que l'algorithme de colonies de fourmis donne les meilleurs résultats lorsque la taille du problème est grande. En revanche, l'algorithme glouton demeure le plus rapide tout en offrant une qualité de minimisation significative.

CHAPITRE 4

SIMULATION D'UN NOUVEAU PROCESSUS PONCTUEL A INTERACTION DE LA PLUS COURTE DISTANCE

4.1 Introduction

Dans ce chapitre, nous proposons un nouveau processus ponctuel à interaction de la plus petite distance. Nous décrivons en détail son fonctionnement et les paramètres qui le définissent. Ensuite, nous procédons à une analyse approfondie de ses propriétés, en mettant l'accent sur sa capacité à capturer efficacement les interactions entre les points dans un contexte de plus petite distance. Nous introduisons ainsi, une famille de processus ponctuels de Markov. Dans le cas le plus simple, la densité de probabilité d'un motif ponctuel $x = \{x_1, ..., x_n\}$ dans une fenêtre $A \subset \mathbb{R}^d$ est définie comme :

$$p(x) = \alpha\beta^n\gamma^{-l(x)} \tag{4.1.1}$$

Où $l(x)$ est la distance la plus courte entre les point x_i . Ici, $\beta > 0$ et γ sont des paramètres et α sont la constante de normalisation. Comparez cela avec le processus d'interaction à paire de Strauss [52] dans la même situation :

$$p(x) = \alpha\beta^n\gamma^{s(x)} \tag{4.1.2}$$

Où $s(x)$ est le nombre de paires de points distincts x_i , x_j qui se trouvent à une distance $r > 0$ l'un de l'autre. Et le processus d'interaction d'air de Baddeley (1995) [3] :

$$p(x) = \alpha\beta^n\gamma^{-u(x)} \tag{4.1.3}$$

Où $u(x)$ est la superficie de l'ensemble formé en prenant l'union de disques de rayon centrés sur les points x_i . Les trois densités se réduisent à un processus de Poisson lorsque $\gamma = 1$.
Notre processus (4.1.1) est bien défini pour toutes les valeurs, contrairement aux processus (4.1.2) et (4.1.3) où notre approche ne repose pas sur un rayon fixe r . Nous définissions un cas particulier pour la norme euclidienne (englobe à la fois les types regroupés et ordonnés) et une généralisation pour n'importe quelle distance.
Il est utile de noter que le calcul de $l(x)$ puisque, c'est un problème connu sous le nom du problème de voyageur de commerce [29] introduit par Euler au XIXe siècle. Donne notre cas, on cherche la plus petite distance entre les points de la configuration x .

4.2 Processus ponctuel a interaction de la plus courte distance

4.2.1 Définition du processus

La démarche formelle pour construire les processus ponctuels finis de Gibbs est décrite dans des ouvrages comme celui de Daley et Vere-Jones (p. 121 et suivantes) [21], Preston (1976,[49]), ou encore la section 2 de Baddeley et Moller (1989,[4]).

Définition 4.2.1. (Cas standard) Le processus à interaction de la plus petite distance dans une région compacte $A \subseteq \mathbb{R}^p$ est le processus avec la densité suivante :

$$p(x) = \alpha\beta^{n(x)}\gamma^{-l(T(x))} \tag{4.2.1}$$

Par rapport au processus de Poisson de taux unitaire sur $A \subseteq \mathbb{R}^p$.
Avec, $\beta > 0$ et $\gamma > 0$ qui sont des paramètres et α est la constante de normalisation, $n(x)$ est le nombre de points de x et $l(T(x)) = min\ d\{x_1, x_2, ..., x_n\}$ est la plus courte distance entre les point $x_1, ..., x_n$ avec $d(x) = d(x_1, x_2, ..., x_n) = \sum_{i=1}^{n-1} \|x_{i+1} - x_i\|$. Pour $\gamma = 1$, cela se réduit bien sûr à un processus de Poisson avec une intensité $\beta\mu$.

Définition 4.2.2. : (Cas général) Soit d une distance sur un espace métrique χ. Le processus à interaction de la plus petite distance général est défini par :

$$p(x) = \alpha\beta^{n(x)}\gamma^{-l(T(x))} \tag{4.2.2}$$

par rapport à p, la distribution du processus de Poisson fini avec une intensité μ.

Avec, $n(x)$ est le nombre de points de x , $T(x)$ représente le plus cours graphe qui relie tous les point de la configuration x et $l(T(x))= min\ d\{x_1, x_2, ..., x_n\}$ tel que $d(x) = d(x_1, x_2, ..., x_n)$ $=\sum_{i=1}^{n-1} d(x_i, x_{i+1})$. $T(x)$ est le plus court graphe qui relie tous les points de la configuration x (i.e. $T(x) = \bigcup_{i=1}^{n-1}[x_i, x_{i+1}]$ avec $[x_i, x_{i+1}]$ est le segment dont l'extrémité sont x_i et x_{i+1}). On note dans ce qui suit $l(T(x))$ par $l(x)$.

Remarque 2. On peut considérer l comme une mesure sur le graphe $T(x)$, on peut facilement le vérifier.

4.2.2 Propriété du processus

Lemme 4.2.1. La densité (4.2.2) est mesurable et intégrable pour toute valeur fixe de $\beta > 0$ et $\alpha > 0$.

preuve

Mesurabilité :

Pour montrer que $p(x)$ est mesurable, nous devons montrer que l'inverse-image de tous ensemble mesurables de réels est mesurable. Cela revient à montrer que les ensembles de la forme $\{x : p(x) > a\}$ sont mesurables pour tout a. Considérons $\{A_a = x : p(x) > a\}$. Nous avons :

$$A_a = \{x : \alpha\beta^{n(x)}\gamma^{-l(x)} > a\}$$
$$A_a = \{x : n(x)\log\beta - l(x)\log\gamma > \log(\alpha/a)\}$$

Maintenant, décomposons A_a en deux ensembles :

$$A_1 = \{x : n(x)\log\beta > l(x)\log\gamma\}$$
$$A_2 = \{x : n(x)\log\beta < l(x)\log\gamma + \log(\alpha/a)\}$$

Ces ensembles sont mesurables, car ils sont construits à partir de fonctions mesurables (fonctions $n(x)$ et $l(x)$) et d'opérations mesurables.

Ainsi, A_a est mesurable comme l'intersection de A_1 et A_2 , deux ensembles mesurables. Puisque cela est vrai pour tout a , alors $p(x)$ est mesurable.

Intégrabilité :

Pour montrer que $p(x)$ est intégrable, il suffit de montre que $p(x)$ est dominé par une fonction intégrable. Montrons que $p(x)$ est dominée par $x \mapsto \alpha\beta^{n(x)}$ La condition de domination serait donc $p(x) \leq \alpha\beta^{n(x)}$. La densité $p(x)$ est définie comme suit :

$$\alpha\beta^{n(x)}\gamma^{-l(x)} \leq \alpha\beta^{n(x)}$$

En d'autres termes, nous devons montrer que $\gamma^{-l(x)} \leq 1$, ce qui est équivalent à montrer que $\gamma^{l(x)} \geq 1$. Comme $l(x)$ est la distance la plus courte, donc elle est positive et finie, alors $0 < l(x) < \infty$ et $\gamma^{l(x)} > 1$, ce qui assure la condition de domination puisque $\gamma > 1$.
Puisque la fonction $x \mapsto \beta^{n(x)}$ est intégrable donnant le processus de Poisson avec une mesure d'intensité $\beta\mu$. Par conséquent, (4.2.2) est dominé par une fonction intégrable, donc intégrable.
Pour $0 < \gamma < 1$: Puisque $\beta > 0$ et $0 < \gamma < 1$, la fonction $p(x)$ est bornée, car $n(x) < \infty$ et $l(x) < \infty$ pour toute configuration de point finie x . Donc, nous pouvons ignorer la constante α et nous concentrer sur les termes $\beta^{n(x)}$ et $\gamma^{-l(x)}$.
Commençons par examiner $\beta^{n(x)}$. Puisque $n(x)$ est une fonction mesurable et intégrable, $\beta^{n(x)}$ est également mesurable et bornée. Par conséquent, il existe une constante $M_1 > 0$ telle que $|\beta^{n(x)}| \leq M_1$ pour tout x dans χ . De même, $\gamma^{-l(x)}$ est bornée puisque $0 < \gamma < 1$. Donc, il existe une constante $M_2 > 0$ telle que $|\gamma^{-l(T(x))}| \leq M_2$ pour tout x dans χ .
En combinant ces bornes, nous avons :

$$p(x) = M_1.M_2 = M$$

Où M est une constante positive. Ainsi, $|p(x)|$ est bornée pour tout x dans χ .
Maintenant, pour montrer que $p(x)$ est intégrable, nous devons démontrer que l'intégrale de $|p(x)|$ sur χ est finie. Puisque $|p(x)|$ est bornée par M , nous avons :

$$\int_x |p(x)|d(x) \leq \int_x Md(x) = Mmes(\chi)$$

où $mes(\chi)$ est la mesure de l'ensemble χ .
Puisque $mes(\chi)$ est finie (puisque les fonctions $n(x)$ et $l(T(x))$ sont intégrables, ce qui implique que χ est de mesure finie), alors $\int_x |p(x)|d(x)$ est finie. Ainsi, la fonction $p(x)$ est intégrable sur χ.

Propriété de Markov :

Théorème 4.2.2. *Le processus ponctuel à interaction de la plus petite distance est un processus ponctuel markovien respectant la relation de voisinage $\sim$ au sens de Ripley and Kelly.*

Preuve

1. C'est clair que si, $p(x) > 0$ alors $p(y) > 0$ $\forall y \in x$ car α, β ,$\gamma > 0$.
2. La fonction de vraisemblance est donnée par

$$\frac{p(x \cup \{x_i\})}{p(x)} = \frac{\alpha\beta^{n(x\cup\{x_i\})}\gamma^{-l(x\cup\{x_i\})}}{\alpha\beta^{n(x)}\gamma^{-l(x)}} = \beta^{n(x\cup\{x_i\})-n(x)}\gamma^{l(x)-l(x\cup\{x_i\})} = \beta\gamma^{l(x)-l(x\cup\{x_i\})}$$

La quantité $l(x)-l(x\cup\{x_i\})$ représente le changement dans la longueur totale du chemin le plus court du processus ponctuel d'interaction de la plus petite distance lorsqu'un nouveau pointx_i est ajouté à l'ensemble x . Elle mesure comment l'ajout de x_i modifie la longueur totale du chemin le plus court. Si cette différence est positive, cela signifie que la nouvelle configuration $x \cup \{x_i\}$ a un chemin plus court que x , et si elle est négative, cela signifie que la nouvelle configuration a un chemin plus long. Ce qui implique que $\frac{p(x \cup x_i)}{p(x)}$ ne dépend que du point x_i et son voisinage $N(x_i) \cap x$.

4.3 Simulation du processus

Il existe plusieurs approches pour simuler les processus ponctuels spatiaux, dont bon nombre sont abordées dans le chapitre 9 de [25] telles que les processus de naissance et de mort spatiaux [4, 50], la méthode de balayage aléatoire de Gibbs [30] et d'autres.

4.3.1 Algorithme de Métropolis-Hastings

Dans cette section, nous examinons une méthode pour simuler (4.2.1) en utilisant l'algorithme de Metropolis-Hastings [27, 28]. Cette méthode consiste à construire une chaîne $X_1, X_2, ..., X_n$ qui converge vers la distribution p souhaitée. Rappelons que l'algorithme passe par deux étapes.

1. Nous proposons un changement d'état de x à y selon la loi de probabilité $Q(x, .)$.
2. On accepte y avec la probabilité $a(x, y)$, sinon on reste dans l'état x (où $a : \Omega\times\Omega \mapsto [0, 1]$).

Notons $q(x, y)$ est la densité de $Q(x, y)$, la transition P de MH s'écrit [13] :

$$P_{MH}(x, y) = a(x, y)q(x, y) + 1[1 - \int_{\Omega} a(x, z)q(x, z)d(z)]\delta_{x(y)}$$

Avec $\delta_x(.)$ et la masse du point en x . Pour simplifier les calculs, on utilise la mesure de Dirac en x ($\delta_x(y) = 1$ si $x = y$ et 0 sinon).
Le choix de (Q, a) assurera la p-réversibilité de P_{MH} si l'équation d'équilibre suivante est satisfaite :

$$\forall(x, y) \in \Omega : p(x) \times q(x, y) \times a(x, y) = p(y) \times p(y, x) \times a(y, x)$$

Le choix de la probabilité d'acceptation a est plus limité : il est dicté essentiellement par l'objectif de simuler (asymptotiquement) une loi de probabilité p donnée. C'est le cas du choix usuel, où :

$$a(x,y) = \frac{p(y)q(y,x)}{p(x)q(x,y}$$

Deux points importants doivent être soulignés. Tout d'abord, le calcul de $a(x,y)$ ne nécessite aucune connaissance de la constante de normalisation de l'équation (4.1.1). Deuxièmement, dans cette étude, nous nous concentrons sur le cas où deux configurations x et y diffèrent exactement en un point. On parle alors de dynamique de renversement de spin (en anglais, "spin flop Dynamique"), où la densité q est symétrique :

$$q(y,x) = q(x,y)$$

Dans ce scénario, la probabilité d'acceptation lorsque le nombre de points souhaité $n(x)$ est fixe se réduit à :

$$a(x,y) = \frac{p(y)}{p(x)} = \frac{\beta^{n(y)}\gamma^{-l(y)}}{\beta^{n(x)}\gamma^{-l(x)}} = \frac{\gamma^{-l(y)}}{\gamma^{-l(x)}}$$

Algorithm 4.1 Algorithme de Métropolis-Hastings

---

i. Étape d'initialisation :
Choisir une configuration initiale ($X_0 = x$ ou $x = \{x_1, x_2, ..., x_n\}$ et $x \in [0,1]^k$) selon une loi de probabilité donnée, par exemple la loi uniforme.
ii. Étape d'itérations :

Pour $N = 1, 2, ..., N_{MCMC}$
Étape 1 :

1. Pour chaque état, x échantillonner y par l'utilisation de dynamiques de renversement de spin.
2. Choisir un spin s au hasard uniformément sur $\{1, ..., n\}$.
3. Simuler une expérience y_j selon la loi uniforme sur$[0,1]^p$. On prend alors comme nouvelle configuration :$y = \{x_1, x_2, ..., x_{s-1}, y_j, x_{s+1}, ..., x_n\}$

Étape 2 :

1. Calculer la probabilité d'acceptation :$a(x,y) = min(1, \gamma^{l(x)-l(y)})$ avec et $\gamma > 0$.
2. Prendre $x = \begin{cases} \text{y} & \text{avec une probabilité : } a \\ \text{x} & \text{avec une probabilité : } 1-a \end{cases}$
3. Répéter ces deux dernières étapes n fois pour chaque itération N .
4. Prendre $X_N = x$.

Pour $N = 1000$ la Figure (4.1) est représente une simulation de (4.1.1) sur $[0,1]^2$ pour $\gamma = 0.9$ et $\gamma = 3.9$

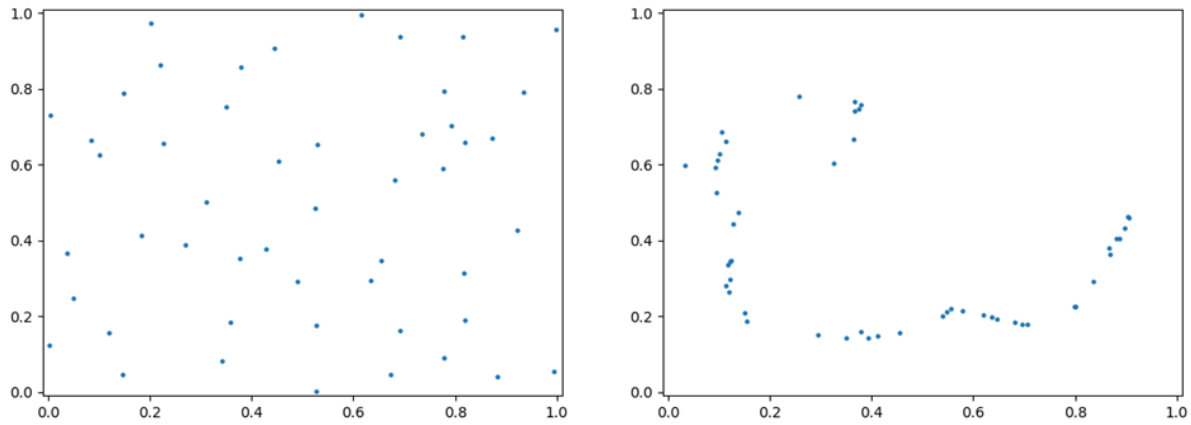

FIGURE 4.1 – Simulation de 50 points d'un processus à interaction de la plus petite distance sur $[0,1]^2$. À gauche, une réalisation bien repartie avec $\gamma = 0.9$ et à droite une réalisation regroupée avec $\gamma = 3.9$

4.3.1.1 Étude de convergence de l'algorithme proposé

Pour chaque itération N de l'algorithme de construction décrit précédemment, nous effectuons n transformation de base d'où, la chaîne des plans d'expériences $(X_N)_{N\geq 0}$ ainsi générée est la réalisation d'une chaine de Markov de noyau de transition :

$$P(x,y) = P_{MH}^n(x,y)$$

À ce niveau, la question fondamentale se pose de savoir si la chaine converge vers la distribution $p(x)$ définie en (4.1.1). La chaine converge vers la distribution invariante p si :

$$P^t(x,y) \xrightarrow[n\to\infty]{} p(A)$$

Où A un borélien de B et $P(x,A) = P(X_t = A/X_0 = x)$ est un noyau de transition de pas t. Énonçons le résultat principal qui nous intéresse ici :

Proposition 4.3.1. sur un espace fini, le noyau de transition P de la chaine de Markov $(X_N)_{N\geq 0}$ obtenue à partir de l'algorithme de construction est récurrent positif, P-stationnaire, apériodique et primitif (primitive Kernel).

Preuve

Montrons que le mécanisme de transition P_{MH} satisfait les trois conditions suivantes relatives à la distribution p du processus du composant connexe, définies en (4.1.1) : p-réversibilité, p-stationnarité et p-irréductibilité.

La chaîne est dite réversible par rapport à la distribution cible $p(\cdot)$ si son noyau de transition satisfait :

$$\forall x, y \in \Omega, B \in \mathcal{B} : \int_\Omega 1_B(x,y)p(x)P_{MH}(x,y)dx = \int_\Omega 1_B(x,y)p(y)P_{MH}(y,x)dy$$

Soient $x, y \in \Omega$ et $B \in \mathcal{B}$. Nous avons alors :

$$\int_\Omega 1_B(x,y)p(x)P_{MH}(x,y)dx = \int_\Omega 1_B(x,y)p(x)a(x,y)q(x,y)dx$$
$$+ \int_\Omega 1_B(x,y)p(x)\left[\int_\Omega 1 - a(x,z)q(x,z)dz\right]\delta_x(y)dx$$

$$= \int_\Omega 1_B(x,y)p(x)a(x,y)q(x,y)dx + \int_\Omega 1_B(x,x)p(x)\left[\int_\Omega 1 - a(x,z)q(x,z)dz\right]dx$$

Par construction, a et q satisfont,

$$p(x)a(x,y)q(x,y) = \alpha\beta^{n(x)}\gamma^{-l(x)}\min\left\{1, \gamma^{l(x)-l(y)}\right\}q(x,y) = \alpha\min\left\{\beta^{n(x)}\gamma^{-l(x)}, \beta^{n(x)}\gamma^{-l(y)}\right\}q(x,y)$$

Et puisque $n(x) = n(y)$ et $q(x,y) = q(y,x)$, nous avons alors :

$$\pi(x)a(x,y)q(x,y) = \alpha\min\left\{\beta^{n(x)}\gamma^{-l(x)}, \beta^{n(y)}\gamma^{-l(y)}\right\}q(y,x) = \alpha\beta^{n(y)}\gamma^{-l(y)}\min\left\{\gamma^{l(y)-l(x)}, 1\right\}q(y,x)$$

$$= p(y)a(y,x)q(y,x)$$

Enfin,

$$\int_\Omega 1_B(x,y)p(x)P_{MH}(x,y)dx = \int_\Omega 1_B(x,y)p(y)a(y,x)q(y,x)dx$$

$$+ \int_\Omega 1_B(y,y)p(y)\left[\int_\Omega 1 - a(y,z)q(y,z)dz\right]dy$$

$$= \int_\Omega 1_B(x,y)p(y)P_{MH}(y,x)dy$$

C'est la condition de réversibilité de p du mécanisme de transition P_{MH}. Une mesure $p(\cdot)$ est dite stationnaire pour le noyau de transition P_{MH} si :

$$\forall x, y \in \Omega; B, A \in \mathcal{B} : \int_{\Omega} 1_B(x,y)p(x)P_{MH}(x,A)dx = \int_{\Omega} 1_B(x,y)p(x)dx$$

Soit $x \in \Omega$, et $B \in \mathcal{B}$. Alors pour tout A de $\mathcal{B}$ nous avons :

$$\int_{\Omega} 1_B(x,y)p(x)P_{MH}(x,A)dx = \int_{\Omega} 1_B(x,y)p(x)\left[\int_{\Omega} a(x,y)q(x,y)dy\right]dx$$

$$+ \int_{\Omega} 1_B(x,y)p(x)\left[\int_{\Omega} 1 - a(x,z)q(x,z)dz\right]\delta_x(y)dx$$

$$= \int_{\Omega} 1_B(x,y)p(x)\left[\int_{\Omega} a(x,y)q(x,y)dy\right]dx + \int_{\Omega} 1_B(x,x)p(x)\left[\int_{\Omega} 1 - a(x,z)q(x,z)dz\right]dx$$

$$= \int_{\Omega}\int_{\Omega} 1_B(x,y)(x)a(x,y)q(x,y)dydx + \int_{\Omega} 1_B(x,x)(x)dx - \int_{\Omega}\int_{\Omega} p(x)a(x,z)q(x,z)dzdx$$

$$= \int_{\Omega} 1_B(x,x)p(x)dx$$

Par conséquent, la chaîne assume p comme distribution stationnaire. Une mesure $p(\cdot)$ est dite irréductible pour le noyau de transition P_{MH} d'une chaîne de Markov si :

$$\forall A \in \mathcal{B} \text{ tel que } p(A) > 0 \Rightarrow \exists t : P^t_{MH}(x,A) > 0$$

Soit A un borélien de $\mathcal{B}$, et pour $t = 1$ nous avons :

$$\int_{\Omega} 1_B(x,A)P_{MH}(x,A)dx = \int_{\Omega} 1_B(x,A)a(x,A)q(x,A)dx$$

$$+ \int_{\Omega} 1_B(x,A)\left[\int_{\Omega} 1 - a(x,z)q(x,z)dz\right]\delta_x(A)dx$$

$$= \int_{\Omega} 1_B(x,A)a(x,A)q(x,A)dx + \int_{\Omega} 1_B(x,x)\left[\int_{\Omega} 1 - a(x,z)q(x,z)dz\right]dx$$

$$= \int_{\Omega} 1_B(x,A)a(x,A)q(x,A)dx + 1 - \int_{\Omega}\int_{\Omega} a(x,z)q(x,z)dzdx$$

Puisque $a(x,A) = \min(1;\gamma^{l(x)-l(A)})$ et $a(x,z) = \min(1;\gamma^{l(x)-l(z)})$, alors quatre cas possibles peuvent être distingués : Si $a(x,A) = 1$ et $a(x,z) = 1$ alors :

$$\int_\Omega 1_B(x,A)P_{MH}(x,A)dx = \int_\Omega 1_B(x,A)q(x,A)dx + 1 - \int_\Omega\int_\Omega q(x,z)dzdx$$

$$= \int_\Omega 1_B(x,A)q(x,A)dx > 0$$

Si $a(x,A) = 1$ et $a(x,z) = \gamma^{l(x)-l(z)}$ alors :

$$\int_\Omega 1_B(x,A)P_{MH}(x,A)dx = \int_\Omega 1_B(x,A)q(x,A)dx + 1 - \int_\Omega\int_\Omega \gamma^{l(x)-l(z)}q(x,z)dzdx$$

$$= \int_\Omega 1_B(x,A)q(x,A)dx + 1 - \gamma^{l(x)-l(z)} > 0$$

Si $a(x,A) = \gamma^{l(x)-l(A)}$ et $a(x,z) = 1$ alors :

$$\int_\Omega 1_B(x,A)P_{MH}(x,A)dx = \int_\Omega 1_B(x,A)\gamma^{l(x)-l(A)}q(x,A)dx + 1 - \int_\Omega\int_\Omega q(x,z)dzdx$$

$$= \gamma^{l(x)-l(A)}\int_\Omega 1_B(x,A)q(x,A)dx > 0$$

Si $a(x,A) = \gamma^{l(x)-l(A)}$ et $a(x,z) = \gamma^{l(x)-l(z)}$ alors :

$$\int_\Omega 1_B(x,A)P_{MH}(x,A)dx = \int_\Omega 1_B(x,A)\gamma^{l(x)-l(A)}q(x,A)dx + 1 - \int_\Omega\int_\Omega \gamma^{l(x)-l(z)}q(x,z)dzdx$$

$$= \gamma^{l(x)-l(A)}\int_\Omega 1_B(x,A)q(x,A)dx + 1 - \gamma^{l(x)-l(z)}\int_\Omega\int_\Omega q(x,z)dzdx$$

$$= \gamma^{l(x)-l(A)}\int_\Omega 1_B(x,A)q(x,A)dx + 1 - \gamma^{l(x)-l(z)} > 0$$

Ainsi, $\int_\Omega 1_B(x,A)P^t_{MH}(x,A)dx > 0$, $\forall t \geq 0$, alors P_{MH} est p-irréductible.

Puisque p est la distribution invariante de P_{MH}, alors elle le reste avec P. En fait, $pP_{MH} = p$, et par récurrence sur n, nous obtenons :

$$pP_{MH} = pP_{MH}^2 = pP_{MH}^3 = \cdots = pP_{MH}^n = p$$

Puisque $P = P_{MH}^n$, alors nous obtenons : $pP = p$. D'autre part, la p-réversibilité de P_{MH} conduit à la p-réversibilité de P, c'est-à-dire :

$$p(x)P_{MH}(x,y) = p(y)P_{MH}(y,x) \Rightarrow p(x)P(x,y) = p(y)P(y,x)$$

Puisque $pP_{MH} = pP_{MH}^n = p$, alors nous obtenons :

$$p(x)P_{MH}(x,y) = p(x)P_{MH}^n(x,y) = p(y)P_{MH}(y,x) = p(y)P_{MH}^n(y,x)$$

Et puisque $P_{MH}^n = P$, alors nous obtenons : $p(x)P(x,y) = p(y)P(y,x)$.

En construisant $P = P_{MH}^n$, l'irréductibilité p de P_{MH} conduit à l'irréductibilité p de P. Si P est irréductible p et a une distribution invariante p, alors p est récurrente positive et p est la seule distribution invariante de P [14] (voir, Proposition 1).

D'autre part, la chaîne générée par l'algorithme de construction sera également apériodique à condition qu'il existe au moins une paire de configurations (x, y) telle que $a(x, y) < 1$, et nous obtiendrons finalement $P(x, x) > 0$. Nous remarquons rapidement que la chaîne est apériodique puisque l'événement $X_{(N+1)} = X_{(N)}$ est probable à presque n'importe quel moment. En fait, chaque état peut alors être visité à deux itérations successives, donc $P^1(x, x) > 0$, et leur période est alors égale à 1.

Puisque la chaîne générée avec l'algorithme est irréductible et apériodique, alors son noyau de transition P est primitif (une caractérisation plus courante d'un noyau de Markov primitif en théorie des probabilités est de dire qu'ils sont irréductibles et apériodiques [51]).

Théorème 4.3.1. *La chaine de Markov $(X_N)_{N\geq 0}$ obtenue à partir de lalgorithme de construction proposé est géométriquement ergodique et son noyau P réalise la simulation d'un processus ponctuel a interaction de la plus courte distance, simulation marqué à deux types de densité :*

$$P(x) = \alpha\beta^{n(x)}\gamma^{-l(x)}$$

c'est-à-dire vP^n tend vers p quand n tend vers l'infini où v est une distribution initiale et on a :

$$\lim_{n\to\infty} \|vP^n - p\| = 0$$

Preuve

Soit v une distribution initiale, pour tout entier, m et $\forall x \in N^{lf}$ nous avons :

$$\|vP^m(x,.) - P\| = \|vP^m - pP^m\| \leq 2C(P^m) \leq 2(C(P))^m$$

Où, $C(P)$ est le coefficient de contraction de Dobrushin de P [23]. D'après la proposition 1, le noyau P est primitive, alors $0 \leq C(P) < 1$ [55] (voir lemme 4.2.3 P.72) donc quand m tends vers l'infini,$\|vP^m - \pi\| \xrightarrow[m\to\infty]{} 0$. D'où, la chaine est uniformément ergodique et converge vers la distribution définie en (4.1.1).

4.3.1.2 Etude de la distribution spatiale des points du processus

L'analyse numérique de la répartition spatiale des points d'un processus ponctuel vise à quantifier et à caractériser la structure spatiale des points. Nous utilisons quelques méthodes d'analyse numérique courantes pour notre processus que nous avons proposé :

L'indice de Moran [1] : est une mesure de l'autocorrélation spatiale globale dans un ensemble de points. Il permet de quantifier la tendance des points à se regrouper ou à se disperser dans l'espace. La formule de l'indice de Moran est donnée par :

$$I = \frac{n}{S_0} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(x_i - \overline{x})(x_j - \overline{x})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Où :

- x_i est la valeur de la variable à l'emplacement i ,
- $\overline{x}$ est la moyenne de toutes les valeurs de la variable,
- w_{ij} sont les poids spatiaux entre les emplacements et donne par [45] :$w_{ij} = \dfrac{1}{d}$ pour $i \neq j$ et 0 pour $i = j$. Et d la distance entre les point x_i et x_j .
- S_0 est la somme des poids spatiaux.

La Figure (4.2) suivante représente des boxe plote de I pour des valeurs de γ supérieur à 1, inférieur à 1 et égale à 1 pour 100 réalisation du processus ponctuel à interaction de plus courte distance en 2 et 3 dimensions.

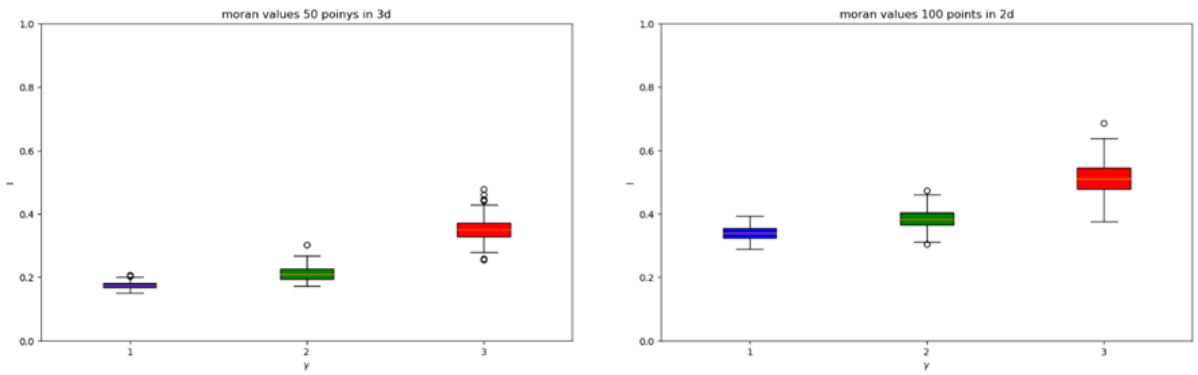

FIGURE 4.2 – présente un box plot des différentes valeurs de I pour différentes valeurs de γ . Les valeurs où $\gamma < 1$ sont représentées en bleu, celles où $\gamma = 1$ en vert, et celles où $\gamma > 1$ en rouge.

Lorsque $\gamma < 1$, une corrélation faible entre les points du processus est observée, ce qui explique la dispersion des points. En revanche, lorsque $\gamma > 1$, la corrélation entre les points est plus forte, entraînant des regroupements. Enfin, lorsque $\gamma = 1$, le processus suit une distribution de Poisson homogène.

CONCLUSION

Ce mémoire a été une exploration passionnante des processus ponctuels et de l'optimisation, deux domaines clés pour comprendre les phénomènes spatiaux et probabilistes. En parcourant ses pages, nous avons découvert comment ces outils mathématiques peuvent modéliser des réalités aussi variées que la répartition des arbres dans une forêt ou la distribution des étoiles dans le cosmos. En plongeant dans l'optimisation, nous avons rencontré des défis complexes, comme celui du voyageur de commerce, et appris comment les algorithmes peuvent nous aider à trouver les meilleures solutions dans un océan de possibilités infinies. Puis, nous avons proposé un nouveau modèle de processus ponctuel, offrant une perspective innovante sur la manière dont les événements spatiaux interagissent. Cette découverte promet d'enrichir notre compréhension et nos applications pratiques dans divers domaines.

En conclusion, ce mémoire représente bien plus qu'une simple étude académique. Il illustre comment la science nous aide à décoder les mystères du monde qui nous entoure. Que ces connaissances inspirent de nouvelles recherches et ouvrent de nouvelles voies vers la compréhension et l'innovation, car chaque découverte éclaire un peu plus notre chemin dans l'univers infini du savoir.

ANNEXE A : CODE EN PYTHON POUR LES RÉSULTATS PRÉSENTÉS EN CHAPITRE 3

La méthode de Branch and Bound

```
import numpy as np
import heapq
import matplotlib.pyplot as plt
import time

class BranchAndBoundTSP:
    def __init__(self, processus, rayon_gaussien=0.1):
        self.processus = processus
        self.nombre_processus = len(processus)
        self.meilleur_chemin = []
        self.meilleure_distance = float('inf')
        self.rayon_gaussien = rayon_gaussien

    def calculer_distance(self, chemin):
        distance_totale = 0
        for i in range(len(chemin) - 1):
            point1 = chemin[i]
            point2 = chemin[i + 1]
            distance_totale += np.linalg.norm(self.processus[point1] - self
                .processus[point2])
        return distance_totale

    def branch_and_bound(self):
        def bound(chemin_partiel):
```

```
            distance = self.calculer_distance(chemin_partiel)
            non_visites = set(range(self.nombre_processus)) - set(
                chemin_partiel)
            if non_visites:
                min_dist = min(np.linalg.norm(self.processus[chemin_partiel
                    [-1]] - self.processus[nv]) for nv in non_visites)
                return distance + min_dist
            return distance

        file_pq = []
        chemin_initial = [0]
        distance_initiale = 0
        heapq.heappush(file_pq, (distance_initiale, chemin_initial))

        while file_pq:
            distance_actuelle, chemin_actuel = heapq.heappop(file_pq)

            if len(chemin_actuel) == self.nombre_processus:
                distance_totale = self.calculer_distance(chemin_actuel)
                if distance_totale < self.meilleure_distance:
                    self.meilleur_chemin = chemin_actuel
                    self.meilleure_distance = distance_totale
            else:
                for i in range(self.nombre_processus):
                    if i not in chemin_actuel:
                        print(i)
                        nouveau_chemin = chemin_actuel + [i]
                        nouvelle_distance = bound(nouveau_chemin)
                        if nouvelle_distance < self.meilleure_distance:
                            heapq.heappush(file_pq, (nouvelle_distance,
                                nouveau_chemin))

    def afficher_solution(self):
        print(f"Meilleur chemin : {self.meilleur_chemin}")
        print(f"Distance du plus court chemin: {self.meilleure_distance}")

    def afficher_graphe(self):
        plt.figure(figsize=(8, 8))
        for i, point in enumerate(self.processus):
            plt.scatter(point[0], point[1], c='r')
```

```
61              plt.text(point[0], point[1], str(i), fontsize=12)
62
63          if self.meilleur_chemin:
64              for i in range(len(self.meilleur_chemin) - 1):
65                  index1 = self.meilleur_chemin[i]
66                  index2 = self.meilleur_chemin[i + 1]
67                  plt.plot([self.processus[index1][0], self.processus[index2
                        ][0]],
68                           [self.processus[index1][1], self.processus[index2
                              ][1]], 'k-')
69
70          plt.title(f'Meilleur chemin obtenue par l\'algorithme Branch and
                Bound\nDistance du chemin optimal: {self.meilleure_distance:.2f
                }')
71          plt.xlabel('X')
72          plt.ylabel('Y')
73          plt.grid(True)
74          plt.show()
75
76      def calculate_execution_time(self):
77          start_time = time.time()
78          self.branch_and_bound()
79          end_time = time.time()
80          execution_time = end_time - start_time
81          return execution_time
82
83  # les points fixe
84  #vous pouvez modifier le nombre et les coordonnées des poins.
85  processus = np.array([
86          [0, 1], [2, 3], [4, 7], [6, 2], [8, 5],
87          [9, 9], [1, 3], [3, 7], [5, 9], [7, 2]
88  ])
89
90  # Paramétres
91  rayon_gaussien = 0.1
92
93  # Initialiser l'algorithme Branch and Bound TSP
94  branch_and_bound_tsp = BranchAndBoundTSP(processus, rayon_gaussien)
95
96  # Calculer et afficher le temps d'éxecution
```

```
execution_time = branch_and_bound_tsp.calculate_execution_time()
print("Temps d'execution:", execution_time)
 #Afficher les resultats
branch_and_bound_tsp.afficher_solution()

# Afficher le graphe de la meilleure solution
branch_and_bound_tsp.afficher_graphe()
```

L'algorithme de Branch and Cut

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import permutations
import time

def generer_points_aleatoires(nombre_processus, rayon_gaussien=0.1):
    """Générer des points  dans R^2."""
    return np.random.normal(0, rayon_gaussien, (nombre_processus, 2))

def distance_euclidienne(point1, point2):
    """Calcule la distance euclidienne entre deux points."""
    return np.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)

def creer_matrice_distance(points):
    """Créer une matrice de distance entre les points."""
    nombre_processus = len(points)
    matrice = np.zeros((nombre_processus, nombre_processus))
    for i in range(nombre_processus):
        for j in range(nombre_processus):
            matrice[i][j] = distance_euclidienne(points[i], points[j])
    return matrice

def calculer_distance_chemin(chemin, matrice_distance):
    """Calculer la distance totale du chemin donné."""
    distance = 0
    for i in range(len(chemin) - 1):
        distance += matrice_distance[chemin[i]][chemin[i + 1]]
    return distance

```

```
def branch_and_cut(matrice_distance):
    """Appliquer la méthode Branch-and-Cut pour résoudre le problème du TSP.
    """
    nombre_processus = len(matrice_distance)
    distance_minimale = float('inf')
    chemin_optimal = None

    for perm in permutations(range(nombre_processus)):
        chemin = list(perm)
        distance = calculer_distance_chemin(chemin, matrice_distance)
        if distance < distance_minimale:
            distance_minimale = distance
            chemin_optimal = chemin

    return chemin_optimal, distance_minimale

def afficher_solution(points, chemin, distance):
    """Afficher le chemin et la distance."""
    plt.figure(figsize=(8, 8))
    for i, point in enumerate(points):
        plt.scatter(point[0], point[1], c='b')
        plt.text(point[0], point[1], str(i), fontsize=12)
    for i in range(len(chemin) - 1):
        plt.plot([points[chemin[i]][0], points[chemin[i + 1]][0]],
                 [points[chemin[i]][1], points[chemin[i + 1]][1]], 'r-')
    plt.title(f"Chemin optimal trouvé (distance: {distance:.2f})")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
    plt.show()

def calculate_execution_time(matrice_distance):
    """Calculer le temps d'exécution pour la méthode Branch-and-Cut."""
    start_time = time.time()
    branch_and_cut(matrice_distance)
    end_time = time.time()
    execution_time = end_time - start_time
    return execution_time

# Paramètres
```

```
nombre_processus = 10
rayon_gaussien = 0.1

# Générer des points  et créer une matrice de distance
#vous pouvez modifier le nombre et les coordonées  des points
points = np.array([
     [0, 1], [2, 3], [4, 7], [6, 2], [8, 5],
     [9, 9], [1, 3], [3, 7], [5, 9], [7, 2]

   ])
matrice_distance = creer_matrice_distance(points)

# Calculer le temps d'exécution
execution_time = calculate_execution_time(matrice_distance)
print("Temps d'exécution:", execution_time)

# Appliquer la méthode Branch-and-Cut
chemin_optimal, distance_optimal = branch_and_cut(matrice_distance)

# Afficher la solution
afficher_solution(points, chemin_optimal, distance_optimal)
print(f"Distance totale du chemin optimal : {distance_optimal:.2f}")
```

L'algorithme de colonies de fourmis

```
import numpy as np
import random
import matplotlib.pyplot as plt
import time

class ColonieFourmis:
    def __init__(self, nombre_fourmis, nombre_points, iterations_max, alpha
        =1, beta=2, rho=0.5, rayon_gaussien=0.1):
        self.nombre_fourmis = nombre_fourmis
        self.nombre_points = nombre_points
        self.iterations_max = iterations_max
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
```

```
        self.pheromones = np.ones((nombre_points, nombre_points))
        self.meilleure_solution_globale = None
        self.meilleure_distance_globale = float('inf')
        self.rayon_gaussien = rayon_gaussien
        self.epsilon = 1e-10  # Petite valeur pour éviter la division par
            zéro

    def initialiser_points(self):
        self.points = np.zeros((self.nombre_points, 2))
        for i in range(self.nombre_points):
            self.points[i][0] = np.random.normal(0, self.rayon_gaussien /
                2)
            self.points[i][1] = np.random.normal(0, self.rayon_gaussien /
                2)

    def initialiser_distances(self):
        self.distances = np.zeros((self.nombre_points, self.nombre_points))
        for i in range(self.nombre_points):
            for j in range(i + 1, self.nombre_points):
                distance = np.sqrt((self.points[i][0] - self.points[j][0])
                    **2 + (self.points[i][1] - self.points[j][1])**2)
                self.distances[i][j] = distance
                self.distances[j][i] = distance

    def lancer_fourmis(self):
        for _ in range(self.iterations_max):
            solutions = []
            distances = []

            for _ in range(self.nombre_fourmis):
                solution, distance = self.construire_solution()
                solutions.append(solution)
                distances.append(distance)

            meilleure_index = np.argmin(distances)
            meilleure_solution = solutions[meilleure_index]
            meilleure_distance = distances[meilleure_index]

            if meilleure_distance < self.meilleure_distance_globale:
                self.meilleure_solution_globale = meilleure_solution
```

```
                self.meilleure_distance_globale = meilleure_distance

            self.mettre_a_jour_pheromones(solutions, distances)

    def construire_solution(self):
        solution = []
        disponible = list(range(self.nombre_points))

        point_depart = random.choice(disponible)
        solution.append(point_depart)
        disponible.remove(point_depart)

        while disponible:
            pheromone_poids = [
                self.pheromones[point_depart][j] ** self.alpha * (1 / (self
                    .distances[point_depart][j] + self.epsilon)) ** self.
                    beta
                for j in disponible
            ]
            somme_pheromone_poids = sum(pheromone_poids)
            if somme_pheromone_poids == 0:
                probabilites = [1 / len(disponible)] * len(disponible)
            else:
                probabilites = [pheromone / somme_pheromone_poids for
                    pheromone in pheromone_poids]
            prochain_point = random.choices(disponible, weights=
                probabilites)[0]
            solution.append(prochain_point)
            disponible.remove(prochain_point)
            point_depart = prochain_point

        distance = self.calculer_distance(solution)
        return solution, distance

    def calculer_distance(self, solution):
        distance = 0
        for i in range(self.nombre_points - 1):
            point1 = solution[i]
            point2 = solution[i + 1]
            distance += self.distances[point1][point2]
```

```
        distance += self.distances[solution[-1]][solution[0]]
        return distance

    def mettre_a_jour_pheromones(self, solutions, distances):
        self.pheromones *= (1 - self.rho)  # Évaporation des phéromones
        for k in range(self.nombre_fourmis):
            solution = solutions[k]
            distance = distances[k]
            pheromone_depose = 1 / (distance + self.epsilon)
            for i in range(self.nombre_points - 1):
                point1 = solution[i]
                point2 = solution[i + 1]
                self.pheromones[point1][point2] += pheromone_depose
                self.pheromones[point2][point1] = self.pheromones[point1][
                    point2]
            point1 = solution[-1]
            point2 = solution[0]
            self.pheromones[point1][point2] += pheromone_depose
            self.pheromones[point2][point1] = self.pheromones[point1][
                point2]

    def afficher_solution(self):
        plt.figure(figsize=(8, 8))

        # Tracer les points
        for i, point in enumerate(self.points):
            plt.scatter(point[0], point[1], c='b')
            plt.text(point[0], point[1], str(i), fontsize=12)

        # Tracer le chemin optimal trouvé par l'algorithme
        meilleur_chemin_x = [self.points[i][0] for i in self.
            meilleure_solution_globale]
        meilleur_chemin_y = [self.points[i][1] for i in self.
            meilleure_solution_globale]
        plt.plot(meilleur_chemin_x, meilleur_chemin_y, 'r-', label='
            Meilleur chemin')

        # Ajouter la distance optimale au titre du graphe
        plt.title(f'Meilleur chemin trouvé par la colonie de fourmis\
            nDistance optimale: {self.meilleure_distance_globale:.2f}')
```

```
120
121         plt.xlabel('X')
122         plt.ylabel('Y')
123         plt.legend()
124         plt.grid(True)
125         plt.show()
126
127     def calculate_execution_time(self):
128         start_time = time.time()
129         self.lancer_fourmis()
130         end_time = time.time()
131         execution_time = end_time - start_time
132         return execution_time
133
134 # Paramètres
135 nombre_fourmis = 20
136 nombre_points = 10  # Correspondre aux points fournis
137 iterations_max = 10000
138 rayon_gaussien = 0.1
139
140 # Initialisation de l'algorithme
141 colonie_fourmis = ColonieFourmis(nombre_fourmis, nombre_points,
    iterations_max, rayon_gaussien=rayon_gaussien)
142
143 # Utiliser les points fixes fournis
144 #vous pouvez modifier le nombre et les coordonées  des points
145 colonie_fourmis.points = np.array([
146     [0, 1], [2, 3], [4, 7], [6, 2], [8, 5],
147     [9, 9], [1, 3], [3, 7], [5, 9], [7, 2]
148
149 ])
150
151 colonie_fourmis.initialiser_distances()
152
153 # Calcul et affichage du temps d'exécution
154 execution_time = colonie_fourmis.calculate_execution_time()
155 print("Execution time:", execution_time)
156
157 # Affichage des résultats
158 colonie_fourmis.afficher_solution()
```

```
159
160 # Afficher la distance optimale à la fin
161 print(f'Distance optimale: {colonie_fourmis.meilleure_distance_globale:.2f}
    ')
```

L'algorithme de recuit simulé

```
1  import numpy as np
2  import random
3  import matplotlib.pyplot as plt
4  import time
5
6  class SimulatedAnnealingTSP:
7      def __init__(self, temperature_initiale, taux_refroidissement,
           nombre_iterations):
8          self.temperature_initiale = temperature_initiale
9          self.taux_refroidissement = taux_refroidissement
10         self.nombre_iterations = nombre_iterations
11         self.meilleure_solution = None
12         self.meilleure_distance = float('inf')
13
14     def calculer_distance(self, chemin, points):
15         distance = 0
16         for i in range(len(chemin) - 1):
17             point1 = chemin[i]
18             point2 = chemin[i + 1]
19             distance += np.linalg.norm(points[point1] - points[point2])
20         return distance
21
22     def recuit_simule(self, points):
23         nombre_points = len(points)
24         chemin_actuel = list(range(nombre_points))
25         random.shuffle(chemin_actuel)
26         distance_actuelle = self.calculer_distance(chemin_actuel, points)
27         self.meilleure_solution = chemin_actuel
28         self.meilleure_distance = distance_actuelle
29         temperature = self.temperature_initiale
30
31         for _ in range(self.nombre_iterations):
```

```
            index1, index2 = random.sample(range(nombre_points), 2)
            nouvelle_solution = chemin_actuel.copy()
            nouvelle_solution[index1], nouvelle_solution[index2] =
                nouvelle_solution[index2], nouvelle_solution[index1]
            nouvelle_distance = self.calculer_distance(nouvelle_solution,
                points)

            if nouvelle_distance < distance_actuelle:
                chemin_actuel = nouvelle_solution
                distance_actuelle = nouvelle_distance
                if nouvelle_distance < self.meilleure_distance:
                    self.meilleure_solution = nouvelle_solution
                    self.meilleure_distance = nouvelle_distance
            else:
                probabilite_acceptation = np.exp(-(nouvelle_distance -
                    distance_actuelle) / temperature)
                if random.random() < probabilite_acceptation:
                    chemin_actuel = nouvelle_solution
                    distance_actuelle = nouvelle_distance

            temperature *= self.taux_refroidissement

        return self.meilleure_solution, self.meilleure_distance

    def afficher_solution(self, points):
        plt.figure(figsize=(8, 8))
        for i, point in enumerate(points):
            plt.scatter(point[0], point[1], c='b')
            plt.text(point[0], point[1], str(i), fontsize=12, ha='center',
                va='center', color='white', bbox=dict(facecolor='blue',
                alpha=0.6))

        for i in range(len(self.meilleure_solution) - 1):
            point1, point2 = self.meilleure_solution[i], self.
                meilleure_solution[i + 1]
            plt.plot([points[point1][0], points[point2][0]], [points[point1
                ][1], points[point2][1]], 'r-')

        plt.title(f'Meilleur chemin trouvé par l\'algorithme du recuit
            simulé\nDistance du chemin optimal sans boucler: {self.
```

```
            meilleure_distance:.2f}')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)
        plt.show()

    def calculate_execution_time(self, points):
        start_time = time.time()
        self.recuit_simule(points)
        end_time = time.time()
        execution_time = end_time - start_time
        return execution_time

# Parameters
temperature_initiale = 1000
taux_refroidissement = 0.99
nombre_iterations = 10000

# Initialization of the algorithm
sa_tsp = SimulatedAnnealingTSP(temperature_initiale, taux_refroidissement,
    nombre_iterations)

# Manually entered points
#You can change the number of points and their coordinates
points = np.array([
  [0, 1], [2, 3], [4, 7], [6, 2], [8, 5],
  [9, 9], [1, 3], [3, 7], [5, 9], [7, 2]
])

# Calculate and display the execution time
execution_time = sa_tsp.calculate_execution_time(points)
print("Execution time:", execution_time)

# Execute the simulated annealing algorithm to find the optimal path
meilleure_solution, meilleure_distance = sa_tsp.recuit_simule(points)

# Display the best solution found
sa_tsp.afficher_solution(points)

# Display the numerical value of the optimal path distance
```

```
print(f"Distance du plus court chemin sans boucler: {meilleure_distance:.2f}")
```

Algorithme glouton

```
import numpy as np
import matplotlib.pyplot as plt
import time

def distance(point1, point2):
    return np.sqrt((point2[0] - point1[0])**2 + (point2[1] - point1[1])**2)

def total_distance(path, points):
    total = 0
    for i in range(len(path) - 1):
        total += distance(points[path[i]], points[path[i + 1]])
    return total

def nearest_neighbor_tsp(points):
    n = len(points)
    current_point = 0
    path = [current_point]
    unvisited = set(range(1, n))

    while unvisited:
        next_point = min(unvisited, key=lambda x: distance(points[
            current_point], points[x]))
        path.append(next_point)
        current_point = next_point
        unvisited.remove(next_point)

    return path

def tsp_without_return(points):
    best_distance = float('inf')
    best_path = None

    for starting_point in range(len(points)):
```

```
        remaining_points = [i for i in range(len(points)) if i !=
            starting_point]
        path = nearest_neighbor_tsp([points[starting_point]] + [points[i]
            for i in remaining_points])
        path_distance = total_distance(path, points)

        if path_distance < best_distance:
            best_distance = path_distance
            best_path = path

    return best_path, best_distance

def calculate_execution_time(func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    execution_time = end_time - start_time
    return result, execution_time

# Generate random points in the interval [0, 1]^2
#You can change the number of points and their coordinates
num_points = 10
points = np.array([
     [0, 1], [2, 3], [4, 7], [6, 2], [8, 5],
     [9, 9], [1, 3], [3, 7], [5, 9], [7, 2]

])

# Apply the algorithm to find the optimal path without returning to the
    starting point
(optimal_path, optimal_distance), execution_time = calculate_execution_time
    (tsp_without_return, points)

print("Optimal path:", optimal_path)
print("Optimal distance:", optimal_distance)
print("Execution time:", execution_time, "seconds")

# Plot the graph with the points and the optimal path
plt.figure(figsize=(8, 8))
plt.scatter(points[:, 0], points[:, 1], c='blue', label='Points')
```

```
for i in range(len(optimal_path) - 1):
    plt.plot([points[optimal_path[i]][0], points[optimal_path[i + 1]][0]],
             [points[optimal_path[i]][1], points[optimal_path[i + 1]][1]],
             c='red', linestyle='--')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('TSP without Returning to Starting Point')
plt.legend()
plt.grid(True)
plt.show()
```

BIBLIOGRAPHIE

[1] Anselin, L. (1995). Local indicators of spatial associationLISA. Geographical analysis, 27(2), 93-115.

[2] Baccelli, F. and Bremaud, P. (2003). Elements of Queueing Theory : Palm Martingale Calculus and Stochastic Recurrences. Springer-Verlag, Berlin.

[3] Baddeley, A. J., & Van Lieshout, M. N. M. (1995). Area-interaction point processes. Annals of the Institute of Statistical Mathematics, 47, 601-619.

[4] Baddeley, A., & Møller, J. (1989). Nearest-neighbour Markov point processes and random sets. International Statistical Review/Revue Internationale de Statistique, 89-121.

[5] Barka,M.,(2015).Le problème du voyageur de commerce :solution exacte et approchée,mémoire de master,Université de M'sila .

[6] Bartlett, M. S. (1954). Processus stochastiques ponctuels. Ann. Inst. H. Poincarfe 14, 3560.

[7] Bayou.A,Bensefia.A.(2021).Un algorithme hybride (Acto-2opt)pour la résolution du problème de voyageur de commerce. mémoire de master,Université de Bordj Bouarreridj.

[8] Bendahmane.A.,(2011),Le recuit simulé.,Master2 RFIA.Université des sciences et de la technologie d'Oran.

[9] Biggs, N. Lloyd, E and Wilson, R.(1986).Graph Theory. 1736-1936.

[10] Billiot,Jean-Michel .(1995). Estimation pour les Processus ponctuels spatiaux de Gibbs. Université paris.

[11] Boumaza,F.(2020). Processus Ponctuels et fiabilité .mémoire de master .Université 8 mai 1945 Guelma.

[12] Cerný,V.(1985).Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm . Journal of Optimization Theory and Applications, vol. 45, no 1, 1er janvier . p. 4151.

[13] Chib, S., &Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. The american statistician, 49(4), 327-335.

[14] Chib, S., & Greenberg, E. (1996). , Markov Chain Monte Carlo Simulation Methods in Econometrics, Econ. Theory. 12 , 409431. https ://doi.org/10.1017/s0266466600006794.

[15] Colorni,A. Dorigo,M et V, Maniezzo.(1991). Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle. Paris, France, Elsevier Publishing. 134-142.

[16] Costanzo,A.Luongo,T,V et G,Marille.(2006).optimisation par colonies de fourmis

[17] Cox, D. R. (1955). Some statistical models connected with series of events. J. Roy. Statist. Soc. B 17, 64-129.

[18] Cox, D. R. and Isham, V. (1980). Point Processes. Chapman & Hall, London.

[19] Daley, D, J. and Vere-Jones, D. (1988). An Introduction to the Theory of Point Processes. Springer Verlag. New York.

[20] Daley, D. J. and Vere-Jones, D. (2003). An Introduction to the Theory of Point Processes. Volume I : Elementary Theory and Methods. Springer-Verlag, New York, 2nd edition.

[21] Daley, D,J et Vere-Jones, D. (2007). Une introduction à la théorie des processus ponctuels : volume II : théorie générale et structure. Médias scientifiques et commerciaux Springer.

[22] Dentzig,G.fulkerson,R.Joheson.S.(1954).Solution of a Lrge-Scale traveling selesman problem,the Rand Corporation.Santa Monica california.

[23] Dobrushin, R. L. (1956). Central limit theorem for nonstationary Markov chains. I. Theory of Probability & Its Applications, 1(1), 65-80.

[24] Dorigo,M.(1992) Optimization, Learning and Natural Algorithms. PhD thesis. Politecnico di Milano. Italie.

[25] Droesbeke, JJ, Lejeune, M. et Saporta, G. (2006). Analyse statistique des données spatiales . Éditions Technip.

[26] Elmoossaoui,H.(2020).Contribution à la méthodologie des plans dexpériences. thèse de Doctorat .Université Blida 1.

[27] Elmossaoui H., Oukid N., & Hananne F., (2020). Construction of computer experiment designs using marked point processes, Afrika Matematika, Doi.org /10.1007/ s13370-020-00770-9, Springer.

[28] Elmossaoui H., Oukid N.,(2023). New computer experiment designs using continuum random cluster point process, International Journal of Analysis and Applications, vol. 21, pp. 5151.

[29] Flood, M. M. (1956). The traveling-salesman problem. Operations research, 4(1), 61-75.

[30] Geman, S. et Geman, D. (1984). Relaxation stochastique, distributions de Gibbs et restauration bayésienne des images. Transactions IEEE sur l'analyse de modèles et l'intelligence artificielle , (6), 721-741.

[31] Girault.M, Stochastique processus ou processus aléatoires,Université de paris 1.

[32] Guttorp, P. and Thorarinsdottir, T. L. (2012). What happened to discrete chaos, the Quenouille process. and the sharp Markov property ? Some history of stochastic point processes. Int. Stat. Rev. 80, 25368.

[33] Jaulin,F.(2008). Processus ponctuels markoviens.

[34] Karp,R, M.(1972). Reducibility Among Combinatorial Problems.

[35] Kelly, F. P., & Ripley, B. D. (1976). A note on Strauss's model for clustering. Biometrika, 357-360.

[36] Kirkpatrick,S. Gelatt,C.D. et Vecchi,M.P.(1983). Optimization by Simulated Annealing . Science, vol. 220. no 4598. p. 671680.

[37] Land,A.H et Doig,A.G.(1960). An Automatic Method of Solving Discrete Programming Problems . Econometrica, vol. 28, no 3,p. 497520

[38] Lieshout.V, M. N. M, (2000)Markov Point Processes and their Applications, Imperial college press, London.

[39] Maqrot,S.(2019).méthodes d'optimisation combinatoire en programmation mathématique,Application à la conception des systhémes verger-maraicher.Université de toulouse.

[40] Mase, S. (1990). Mean characteristics of Gibbsian point processes. Annals of the Institute of Statistical Mathematics, 42, 203-220.

[41] Mitchell,J.E.(2011),Branch and cut,Wiley Encyclopedia of operations Research and Management science.

[42] Moller, J. (1994). Markov chain Monte Carlo and spatial point processes, Research Report, 293, Mathematical Institute, University of Aarhus, Denmark.

[43] Monge,G.(1781). Mémoire sur la théorie des déblais et de remblais. Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année, pages 666704.

[44] Morelle,A.(2006).Introduction à L'algorithme ,esiee Paris.

[45] Niu, H., & Lekse, W. (2018). Carbon emission effect of urbanization at regional level : empirical evidence from China. Economics, 12(1), 20180044.

[46] ORTNER,O.(2004), Processus Ponctuels Marqués pour l'Extraction Automatique de Caricatures de Bâtiments à partir de Modèles Numériques d'Elévation , thèse pour obtenir le titre de Docteur en Sciences, Université de Nice Sophia Antipolis.

[47] Oumane,M.W.(2020).Recherche de chemin et optimisation multi-objectifs,mémoire de master,Université de Biskra.

[48] Palm, C. (1943). Intensitatsschwankungen im Fernsprechverkehr. Ericsson Technics 44, 1189.

[49] Preston, C. J. (1976). random Fields, Springer Verlag Berlin.

[50] Preston, C. J. (1977). Spatial birth-and-death processes, Bull. InterTzat. Statist. Inst., 46, 371- 391.

[51] Senata.E.(1981). Non-Negative Matrices and Markov Chains, 2^{nd} edition. Springer, New York Heidelberg Berlin.

[52] Strauss, D. J. (1975). A model for clustering. Biometrika, 62(2), 467-475.

[53] Streit.Roy. L. (2010)."Poisson Point Processes : Imaging, Tracking, and Sensing.Springer Science & Business Media. p 1314.

[54] Sung Nok Chiu , Dietrich Stoyan ,Wilfrid S.Kendall,Joseph Mecke .,(2013). Stochastic Geometry and its Applications ,.p 110-341.

[55] Winkler, G. (2012). Image analysis, random fields and Markov chain Monte Carlo methods : a mathematical introduction (Vol. 27). Springer Science & Business Media.