

DEPARTEMENT DE PHYSIQUE APPLIQUEE ET OPTIQUE

MEMOIRE

Présenté par :

BRAKNA Chemse Eddine

Pour l'obtention du diplôme de Master

Spécialité : Instrumentation et métrologie

Thème

Optimisation du Trajet d'une Machine CNC à l'Aide de Divers Algorithmes

Soutenu le devant la commission d'examen

Dr A. GOURBI	Institut des Sciences et Techniques Appliquées Université d'Oran 1	Président
Dr E.AIT SI LARBI	Institut des Sciences et Techniques Appliquées Université Oran 1	Examineur
Dr M. CHAIB	Institut des Sciences et Techniques Appliquées Université Oran 1	Encadreur
Dr S. DAHMANE	Faculté de Technologie Sidi Bel Abbes	Co-encadreur

Année Universitaire 2024-2025

I. Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet de fin d'études

Avant tout, je remercie chaleureusement mon encadrant, **Dr M. CHAIB** pour son accompagnement constant, ses conseils avisés et sa disponibilité tout au long de ce travail.

Je souhaite également adresser mes sincères remerciements à mon co-encadrant, **Dr S. DAHMANE** pour son soutien technique, ses remarques pertinentes et son implication précieuse dans l'avancement de ce projet.

Je remercie l'ensemble des enseignants et responsables pédagogiques de **ISTA** pour la qualité de la formation que j'ai reçue, ainsi que pour leur engagement envers la réussite des étudiants.

Mes remerciements vont aussi à mes camarades et amis, pour leur soutien, leur bonne humeur, et les échanges enrichissants qui ont marqué cette année.

Enfin, je dédie une pensée pleine de reconnaissance à ma famille, pour son soutien moral, sa patience et sa confiance indéfectible, qui m'ont permis d'aller au bout de ce parcours.

À toutes et à tous, merci.

II. Résumé

Le perçage est un procédé d'usinage par enlèvement de copeaux largement utilisé dans le domaine de la fabrication. Le terme perçage regroupe l'ensemble des méthodes permettant de réaliser des trous cylindriques dans une pièce à l'aide d'outils coupants. Ce procédé est appliqué dans de nombreux domaines, notamment pour le perçage de trous dans les circuits imprimés (PCB). Aujourd'hui, des machines CNC sont utilisées pour automatiser le processus de perçage. L'un des problèmes majeurs liés à leur utilisation est la séquence de perçage, car il existe généralement un grand nombre de points à visiter. La détermination de la séquence optimale est similaire à un problème de synchronisation des déplacements et de réduction du temps de parcours.

L'objectif principal de ce travail est d'optimiser le temps et la trajectoire d'une machine CNC à trois axes équipée d'un foret automatique, afin d'atteindre les meilleures performances possibles. Pour cela, nous avons mis en œuvre des algorithmes d'optimisation tels que l'algorithme génétique, l'algorithme ACO (Ant Colony Optimization) et l'algorithme A* pour réduire le temps d'usinage nécessaire au perçage d'un ensemble donné de trous, tout en minimisant les coûts de production, et ce sans compromettre la précision du mouvement.

Les résultats obtenus à travers des simulations démontrent que l'approche proposée est à la fois efficace et applicable. Elle constitue une solution pertinente pour la planification des opérations dans les systèmes de fabrication assistés par machine CNC, en particulier dans les contextes où la rapidité et l'automatisation sont essentielles.

III. Table des matières

I. Remerciements.....	2
II. Résumé	3
III. Table des matières	4
IV. Liste des abréviations.....	6
V. Liste des tableaux.....	7
VI. Liste des figures	8
VII. Introduction générale	9
1. Chapitre 1: Introduction aux Machines CNC et Carte PCB.....	10
1.1. Introduction.....	10
1.2. Carte à Circuits Imprimés (PCB).....	10
1.3. Types de PCB selon les Applications.....	11
1.4. Conception et Fabrication de Cartes PCB	14
1.5. Étapes du Processus de Fabrication d'un PCB	14
1.6. Les Machines CNC et leur Importance dans la Fabrication des PCB	17
1.7. Le Fichier Gerber : Format Standard pour la Fabrication des PCB.....	19
1.8. Conclusion	21
2. Chapitre 2 : Algorithmes d'Optimisation	22
2.1. Introduction.....	22
2.2. Optimisation Combinatoire et Application au Trajet CNC.....	22
2.3. L'Algorithme Génétique (AG).....	24
2.4. L'Algorithme de la Colonie de Fourmis (ACO)	26
2.5. L'Algorithme A* (A Star)	29
2.6. Modélisation de Monte Carlo	31
2.7. Conclusion	32
3. Chapitre 3: Fenêtre graphique MATLAB et simulation	34

Table des matières

3.1. Introduction.....	34
3.2. Outils et environnement de développement.....	34
3.3. Structure générale de l'interface.....	35
3.4. Chargement et traitement des données.....	37
3.5. Implémentation des algorithmes	38
3.6. Visualisation des résultats	40
3.7. Simulation de l'usinage.....	43
3.8. Exportation du G-code.....	45
3.9. Conclusion	48
4. Chapitre 4: Résultats et interprétations.....	50
4.1. Introduction.....	50
4.2. Données de test et scénarios	50
4.3. Analyse comparative des algorithmes.....	51
4.4. Analyse par simulation de Monte Carlo.....	55
4.5. Conclusion	60
VIII. Conclusion générale	62
IX. Annexe A : Algorithmes	63
X. Annexe B : calcul du temps total d'usinage	69
XI. Annexe C : fichier Gerber contenant les coordonnées de perçage.....	70
XII. Bibliographie	71

IV. Lite des abréviations

PCB :Printed Circuit Board (Carte à Circuit Imprimé)

CNC : Computer Numerical Control (Commande Numérique par Ordinateur)

TSP : Travelling Salesman Problem (Problème du Voyageur de Commerce)

GA : Genetic Algorithm (Algorithme Génétique)

ACO : Ant Colony Optimization (Optimisation par Colonies de Fourmis)

A* : A Star (Algorithme A étoile)

OX : Order Crossover (Croisement par Ordre)

GUI : Graphical User Interface (Interface Graphique Utilisateur)

G-code : Geometric Code (Code utilisé pour contrôler les machines CNC)

V. Liste des tableaux

Table 1 : Types de PCB utilisés selon les domaines d'app	13
Table 2: Types de machines CNC utilisées en électronique et leurs fonctions.....	17
Table 3 : Principaux composants d'une machine CNC de perçage et leur fonction.....	18
Table 4 : Paramètres de Perçage pour la Machine CNC	43
Table 5 : Distance totale parcourue par chaque algorithme sur des données réelles issues d'un fichier Gerber	52
Table 6 : Temps d'usinage estimé pour chaque algorithme sur des données réelles	53
Table 7 : Simulation Monte Carlo : statistiques sur la distance totale (10 points).....	55
Table 8 : Simulation Monte Carlo : temps d'usinage estimé (10 points)	56
Table 9 : Simulation Monte Carlo : statistiques sur la distance totale (30 points).....	57
Table 10 : Simulation Monte Carlo : temps d'usinage estimé (30 points)	58
Table 11 : Simulation Monte Carlo : statistiques sur la distance totale (50 points).....	58
Table 12 : Simulation Monte Carlo : temps d'usinage estimé (50 points)	59
Table 13 : Résumé comparatif des performances des algorithmes sur 10, 30 et 50 points (Monte Carlo)	59

VI. Liste des figures

Figure 1: Étapes de l'algorithme génétique	24
Figure 2 : Les étapes de la recherche de nourriture chez les fourmis.....	26
Figure 3 : L'Algorithme A*	29
Figure 4 : Écran de démarrage de MATLAB R2013a utilisé pour le développement de l'interface	34
Figure 5 : Interface de conception graphique GUIDE dans MATLAB	35
Figure 6 : Vue initiale de l'interface graphique MATLAB avant exécution	36
Figure 7 : Exemple d'extrait d'un fichier Gerber contenant les coordonnées de perçage	Error!
Bookmark not defined.	
Figure 8 : Lecture des coordonnées de perçage à partir d'un fichier texte en MATLAB	37
Figure 9 : Affichage des points de perçage extraits du fichier Gerber	38
Figure 10 : Code MATLAB pour l'affichage des points de perçage.....	38
Figure 11 : Parcours optimisés par les trois algorithmes (Génétique, ACO, A*)	41
Figure 12 : Distances et temps d'usinage obtenus pour chaque algorithme.....	41
Figure 13 : Convergence des algorithmes.....	42
Figure 14 : Paramètres technologiques saisis dans l'interface	43
Figure 15 : Fenêtre d'enregistrement du fichier G-code.....	45
Figure 16 : Avertissement sur la plage de vitesse d'avance (Feedrate)	46
Figure 17 : Exemple d'instruction G-code générée pour le perçage	47
Figure 18 : Confirmation de génération réussie du G-code	48
Figure 19 : Parcours optimisés à partir d'un fichier Gerber réel.....	51
Figure 20 : Graphe de convergence des distances sur cinq exécutions	54
Figure 21 : Zoom sur la convergence d'ACO et A*	54
Figure 22 : Simulation Monte Carlo : histogrammes des temps d'usinage pour 10 points	55
Figure 23 : Simulation Monte Carlo : histogrammes des temps d'usinage pour 30 points	57
Figure 24 : Simulation Monte Carlo : histogrammes des temps d'usinage pour 50 points	58

VII. Introduction générale

Dans un monde où la miniaturisation et la performance des dispositifs électroniques deviennent de plus en plus cruciales, la fabrication des cartes électroniques imprimées (PCB) joue un rôle central dans presque tous les domaines technologiques : télécommunications, aéronautique, automobile, domotique, etc. [1]

Les machines CNC (Computer Numerical Control), notamment celles utilisées pour le perçage des PCB, permettent une automatisation précise et rapide des processus de fabrication [2]. Toutefois, pour améliorer l'efficacité de ces machines, il est indispensable d'optimiser le trajet de perçage, en réduisant le temps et la distance parcourue par l'outil [3].

Parmi les méthodes d'optimisation et d'analyse utilisées, la modélisation de Monte Carlo occupe une place importante. Elle repose sur l'utilisation de techniques probabilistes pour simuler des scénarios multiples et évaluer la performance ou la robustesse d'un système [4]. Appliquée au contexte industriel, elle permet par exemple d'estimer la durée moyenne des parcours ou d'évaluer la probabilité d'atteindre une performance donnée en fonction des paramètres d'entrée.

En complément, plusieurs algorithmes inspirés de l'intelligence artificielle sont mis en œuvre pour l'optimisation des trajets de perçage, notamment l'algorithme génétique, l'algorithme de la colonie de fourmis (ACO) et l'algorithme A* .

Ce mémoire présente une étude approfondie sur l'optimisation du trajet de perçage d'une machine CNC en appliquant ces différentes approches, ainsi que le développement d'une interface MATLAB permettant la mise en œuvre et l'évaluation par simulation de ces algorithmes.

1. Chapitre 1: Introduction aux Machines CNC et Carte PCB

1.1. Introduction

Les cartes électroniques imprimées (PCB - Printed Circuit Boards) constituent l'ossature fondamentale de presque tous les systèmes électroniques modernes. Elles permettent de connecter électroniquement et mécaniquement des composants électroniques à l'aide de pistes conductrices gravées sur un substrat non conducteur [1]. Grâce à leur standardisation et leur fiabilité, les PCB sont devenues incontournables dans les domaines des télécommunications, de l'automobile, de l'aérospatiale, de la domotique et de l'électronique grand public [2].

La fabrication des PCB, bien qu'automatisée à grande échelle, implique plusieurs étapes critiques allant de la conception schématique à la gravure des couches en passant par le perçage de trous pour l'insertion ou le montage de composants traversants. C'est dans cette dernière étape que les machines CNC (Contrôle Numérique par Ordinateur) interviennent, en assurant une exécution précise et répétable du perçage, sans intervention manuelle [3].

Avec l'augmentation de la complexité des circuits et la réduction de leur taille, il devient essentiel de minimiser le temps de perçage, ce qui dépend directement du trajet suivi par l'outil de la machine CNC. L'optimisation de ce trajet s'apparente à un problème classique d'optimisation combinatoire connu sous le nom de "Problème du Voyageur de Commerce" (TSP) [4].

Ce chapitre explore dans un premier temps les différents types de cartes PCB, leurs caractéristiques et leurs domaines d'application. Ensuite, une attention particulière sera portée aux machines CNC, à leurs composants, à leur principe de fonctionnement, et enfin à leur rôle dans le processus de fabrication des circuits imprimés

1.2. Carte à Circuits Imprimés (PCB)

Une carte à circuits imprimés (PCB – Printed Circuit Board) supporte mécaniquement et connecte électriquement des composants électroniques à l'aide de pistes conductrices, de pastilles et d'autres éléments gravés à partir d'une ou plusieurs couches de cuivre, laminées sur et/ou entre des couches d'un substrat non conducteur [1].

Les composants sont généralement soudés à la surface de la carte pour garantir à la fois la connexion électrique et la fixation mécanique. Ce type de carte est omniprésent dans presque tous les produits électroniques, sauf les plus simples. On le retrouve aussi dans certains dispositifs électriques comme les boîtiers de commutation passifs [5].

Les alternatives aux PCB, telles que le câblage point à point ou l'enroulement de fils (wire wrap), étaient autrefois populaires, mais sont aujourd'hui rarement utilisées, car elles sont moins fiables et moins adaptées à la production en série [5].

Bien que les PCB nécessitent un effort initial de conception du schéma électrique, leur fabrication et assemblage sont largement automatisables. Des logiciels de CAO spécialisés permettent d'automatiser une grande partie de cette tâche, rendant la production de masse plus rapide et moins coûteuse que les méthodes de câblage traditionnelles [1].

Les PCB peuvent être classés selon leur structure :

- PCB simple face : une seule couche de cuivre.
- PCB double face : deux couches de cuivre sur chaque face du substrat.
- PCB multicouches : plusieurs couches internes et externes de cuivre alternées avec des couches de substrat.

Les PCB multicouches permettent une densité de composants bien plus élevée, car les pistes peuvent circuler à l'intérieur du circuit, libérant ainsi de l'espace en surface [1]. Cependant, l'adoption de cette technologie, surtout en combinaison avec le montage en surface (SMT), a rendu la réparation et la modification sur le terrain plus difficiles, voire parfois impraticables [5]

1.3. Types de PCB selon les Applications

Les cartes électroniques imprimées (PCB) se déclinent en plusieurs types en fonction de leur complexité, de leur nombre de couches, de leur matériau et de leur usage spécifique. Voici les principaux types

1.3.1. Carte simple face (Simple Layer PCB)

C'est le type de carte le plus basique. Il est constitué d'un seul côté conducteur, généralement en cuivre, et d'un substrat isolant comme la fibre de verre (FR-4) ou le phénolique. Les composants sont montés sur un seul côté, ce qui limite la densité du circuit.

Ces cartes sont largement utilisées dans les produits électroniques de consommation bas de gamme comme les jouets, les calculatrices et les appareils électroménagers simples [1].

1.3.2. Carte double face (Double Layer PCB)

Contrairement aux cartes simples, les cartes double face possèdent des pistes conductrices sur les deux côtés du substrat. Les connexions entre les deux faces se font à l'aide de trous métallisés appelés vias.

Ce type est très utilisé dans les systèmes embarqués, les cartes Arduino et les applications industrielles moyennement complexes [5].

1.3.3. Carte multicouche (Multilayer PCB)

Les cartes multicouches comprennent trois couches conductrices ou plus, séparées par des isolants. Elles permettent une densité de composants bien plus importante tout en réduisant la taille globale du circuit.

On les trouve dans les ordinateurs portables, les smartphones, les équipements de télécommunications, les dispositifs médicaux et les systèmes militaires [1][5].

1.3.4. Carte à haute fréquence (High-Frequency PCB)

Conçues pour transmettre des signaux à haute fréquence (au-delà de 1 GHz), ces cartes utilisent des matériaux spéciaux comme le téflon (PTFE) pour minimiser les pertes de signal.

Elles sont indispensables dans les systèmes radar, les satellites, les dispositifs RF et les communications sans fil [5].

1.3.5. Carte flexible (Flexible PCB)

Fabriquées à partir de films plastiques comme le polyimide, ces cartes peuvent être pliées ou enroulées sans casser. Elles sont utilisées dans les appareils portables, les caméras, les imprimantes et les implants médicaux.

Grâce à leur légèreté et leur souplesse, elles s'intègrent dans des environnements où l'espace est restreint ou en mouvement [1].

1.3.6. Carte rigide-flex (Rigid-Flex PCB)

Combinaison d'une ou plusieurs cartes rigides et flexibles intégrées ensemble. Elles offrent des avantages mécaniques et une fiabilité supérieure, souvent utilisées dans l'aérospatial, les drones et l'électronique médicale avancée [5].

❖ Le tableau ci-dessous présente les différents types de cartes PCB selon leur domaine d'application, leurs caractéristiques spécifiques ainsi que les références associées

Application	Type de PCB	Particularité principale	Référence
Téléviseurs	Double couche, Multicouche	Circuits vidéo/audio complexes, traitement du signal haute définition	[5]
Ordinateurs personnels	Multicouche	Transmission rapide des données, miniaturisation, réduction des interférences	[1]
Industrie (robots, capteurs)	Multicouche, Haute fréquence	Résistance aux températures extrêmes et vibrations, stabilité électromagnétique élevée	[5], [8]
Téléphones portables, montres, etc.	Flexible, Ultrafin, HDI (High Density)	Ultra miniaturisation, intégration dans des volumes restreints, flexibilité mécanique	[1], [7]
Aéronautique, Défense	Rigide-flex, Haute fiabilité	Résistance aux chocs et vibrations, durée de vie prolongée, performance critique	[6]
Dispositifs médicaux implantables	Flexible biocompatible, multicouche	Biocompatibilité, fiabilité critique, dimensions extrêmement réduites	[7]
Réseaux 5G, Radar automobile	PCB Haute fréquence (PTFE, Rogers)	Transmission de signaux à haute fréquence, très faible perte de signal	[5], [8]

Table 1 : Types de PCB utilisés selon les domaines d'app

1.4. Conception et Fabrication de Cartes PCB

La conception et la fabrication des cartes PCB représentent une étape critique dans le développement de tout système électronique. Aujourd'hui, grâce à la mondialisation et à la disponibilité de services en ligne, il est devenu plus facile que jamais de concevoir, prototyper et produire des circuits imprimés, même pour de petites séries [1].

Cependant, il est essentiel de comprendre que la réussite d'un produit électronique ne repose pas uniquement sur la fabrication. La qualité de la conception initiale joue un rôle déterminant. De nombreux fabricants à bas coût, souvent localisés en Asie, proposent uniquement des services de production pure, c'est-à-dire qu'ils fabriquent exactement selon les fichiers Gerber fournis, sans aucune vérification d'erreurs potentielles dans le design [5]. Cette approche comporte un risque : en cas d'erreur de conception, celle-ci ne sera détectée qu'après fabrication, entraînant des pertes de temps, d'argent et des retards de mise sur le marché.

D'un autre côté, certaines entreprises proposent des services de conception et fabrication intégrés, souvent appelés "conception clé en main". Ces services accompagnent le client depuis l'étape du schéma électrique, jusqu'à la génération des fichiers de production, en passant par la vérification des règles de conception (DRC – Design Rule Check), le placement des composants et le routage des pistes [1][9].

- Pour la conception des PCB, on utilise des logiciels de CAO (Conception Assistée par Ordinateur) tels que : Altium Designer ,Eagle ,KiCad ,Proteus ,OrCAD

Ces outils permettent la génération automatique de fichiers Gerber, de plans d'assemblage, de nomenclatures (BOM), et facilitent également les vérifications électriques avant fabrication [9].

1.5. Étapes du Processus de Fabrication d'un PCB

La fabrication d'une carte PCB suit une série d'étapes techniques rigoureuses, allant de la conception numérique jusqu'au test final. Voici les principales phases du processus industriel standard .

Étape 1 : Conception et Exportation des fichiers

La fabrication commence par la création du schéma électronique et du routage des pistes à l'aide de logiciels de CAO spécialisés comme KiCad, Altium Designer, ou Eagle. Une fois la

conception validée, elle est exportée sous forme de fichiers Gerber, un format standard qui contient toutes les données nécessaires à la production (pistes, perçages, couches, etc.).

Avant toute fabrication, une analyse DFM (Design for Manufacturing) est réalisée par le fabricant pour vérifier la conformité du design aux contraintes techniques de production, réduisant ainsi les risques d'erreurs coûteuses[10].

Étape 2 : Création des Films Photos

Les fichiers Gerber sont utilisés pour générer des films photolithographiques servant à l'insolation des couches. Chaque couche (interne, cuivre externe, masque de soudure, sérigraphie) est représentée par un film haute précision comportant des zones opaques et transparentes [10].

Étape 3 : Imagerie des Couches Internes

Pour les PCB multicouches, les motifs de chaque couche interne sont transférés sur une feuille de cuivre à l'aide d'un photoresist (résine photosensible). Une exposition aux rayons UV durcit les zones protégées, puis une révélation chimique supprime les parties non exposées, définissant ainsi les futures pistes de cuivre [10].

Étape 4 : Gravure Chimique

Le cuivre exposé est dissous par une solution chimique, ne laissant que les pistes protégées par le photoresist. Ensuite, ce dernier est retiré, révélant les pistes finales de chaque couche [10].

Étape 5 : Alignement et Inspection Optique

Les couches internes sont alignées à l'aide de trous de repérage, puis soumises à une inspection optique automatique (AOI). Un système optique ou laser compare chaque couche au modèle numérique et détecte les éventuelles anomalies [10].

Étape 6 : Laminage et Pressage

Les couches internes, le prépreg (fibre de verre avec résine) et les couches de cuivre externes sont empilées. Un pressage à chaud fusionne l'ensemble sous pression et température contrôlées, formant un bloc multicouche rigide [10].

Étape 7 : Perçage

Des trous sont forés à des emplacements très précis pour permettre l'insertion de composants ou le passage de vias. Le perçage est réalisé à l'aide de forêts en carbure tournant à haute vitesse, parfois assistés par rayons X pour l'alignement [10].

Étape 8 : Métallisation des Trous

Une couche de cuivre très fine (1 μm) est déposée par dépôt chimique (PTH) sur toute la surface, y compris dans les trous, établissant la connexion entre les couches [10].

Étape 9 : Imagerie des Couches Externes

Les couches externes subissent le même processus d'insolation UV que les couches internes, définissant les pistes visibles du circuit final [10].

Étape 10 : Placage et Renfort des Pistes

Un placage électrolytique ajoute une couche de cuivre plus épaisse sur les pistes et les pastilles. Ensuite, une couche d'étain est déposée pour protéger le cuivre lors de l'étape suivante [10].

Étape 11 : Gravure Finale

Le cuivre excédentaire est éliminé par attaque chimique, seules les pistes protégées par l'étain subsistent. L'étain est ensuite retiré pour révéler le cuivre proprement gravé[10].

Étape 12 : Application du Vernis de Protection (Solder Mask)

Un vernis époxy (de couleur verte ou autre) est appliqué pour protéger les pistes contre l'oxydation et éviter les courts-circuits. Il est durci par UV puis cuisson [10].

Étape 13 : Finition de Surface

Pour garantir une bonne soudabilité, la surface est recouverte d'un matériau comme :

Or chimique (ENIG) ,Argent (Immersion Silver) ,Étain fondu (HAL) Ces traitements dépendent de la méthode d'assemblage prévue [10].

Étape 14 : Sérigraphie

Des indications techniques (logos, références, polarités) sont imprimées sur le PCB à l'aide d'une encre blanche via sérigraphie ou impression jet d'encre[10].

Étape 15 : Test Électrique

Un test électrique final vérifie la continuité des pistes et l'absence de court-circuit à l'aide d'un système automatisé ou par "flying probe" (sondes mobiles pour circuits complexes) [10].

1.6. Les Machines CNC et leur Importance dans la Fabrication des PCB

1.6.1. Définition et Historique des CNC

Les machines à commande numérique par ordinateur (CNC) sont des systèmes mécatroniques qui permettent l'automatisation de l'usinage via un contrôle informatique. Développées à la fin des années 1940 au MIT, les CNC ont révolutionné l'industrie en remplaçant les commandes manuelles par des instructions numériques [1].

Ces instructions sont codées sous forme de fichiers G-code, qui spécifient les mouvements de l'outil, les vitesses, les profondeurs de coupe et d'autres paramètres de traitement.

1.6.2. Fonctionnement général d'une machine CNC

Une machine CNC est composée de trois sous-systèmes majeurs :

- Le module de commande : il interprète les instructions numériques (G-code), régule les déplacements des moteurs et gère les interruptions (limites, erreurs, etc.) [2].
- Le système de translation : constitué de rails linéaires et de vis à billes qui déplacent les axes X, Y, et Z avec haute précision.
- Le système d'usinage : broche ou tête de coupe, dotée de forets ou fraises, pouvant tourner à haute vitesse (jusqu'à 100 000 RPM dans le perçage PCB).

1.6.3. Typologie des machines CNC

Les CNC se déclinent selon leur fonction ou structure :

Type de machine CNC	Utilisation
CNC à 3 axes (X, Y, Z)	Perçage, fraisage, gravure de circuits.
CNC à 5 axes	Usinage complexe 3D (non utilisé pour PCB).
CNC laser	Découpe ou marquage sans contact.
CNC de perçage automatique	Perçage rapide des vias et trous composants.
Pick-and-place CNC	Placement automatique des composants SMD.

Table 2: Types de machines CNC utilisées en électronique et leurs fonctions

1.6.4. Composants d'une Machine CNC pour le Perçage des PCB

Une machine CNC dédiée au perçage de circuits imprimés est un système mécatronique complexe, composé de divers éléments travaillant en synergie pour garantir la précision, la répétabilité et la sécurité de l'usinage[2].

Voici les principaux composants:

Composant	Fonction et description
Châssis (Structure)	Cadre rigide (souvent en aluminium ou en acier) qui supporte toute la structure. Doit être robuste pour limiter les vibrations.
Table de support	Surface plane où le PCB est fixé. Elle peut intégrer un système de vide (aspiration) ou des pinces pour l'immobiliser.
Axes de déplacement (X, Y, Z)	Assurent les mouvements : X (gauche-droite), Y (avant-arrière), Z (montée/descente). Montés sur rails linéaires de guidage.
Moteurs pas à pas / Servomoteurs	Fournissent des mouvements précis et contrôlés sur chaque axe. Les moteurs pas à pas sont courants dans les machines de prototypage.
Vis à billes / Courroies	Transmettent le mouvement des moteurs aux axes avec précision. Les vis à billes offrent peu de frottement et une meilleure rigidité.
Broche (Spindle)	Moteur rotatif qui entraîne le foret. Tourne généralement à 40 000–100 000 tr/min, permettant de percer des trous très fins (0,1 mm).
Système de changement d'outils	Peut être manuel (MTC) ou automatique (ATC). Permet de changer les forets selon le programme de perçage.
Contrôleur CNC	Carte électronique qui interprète les instructions (souvent issues d'un fichier G-code ou Gerber) et contrôle les moteurs et la broche.
Logiciel de commande	Interface PC qui permet de : charger les fichiers, ajuster les vitesses, contrôler la profondeur et lancer l'usinage.
Capteurs de fin de course	Détection des extrémités physiques de chaque axe pour éviter les collisions et assurer un référencement initial.
Système d'aspiration	Élimine les poussières produites lors du perçage pour garder la machine propre et sécurisée. Indispensable pour la santé de l'opérateur.
Bouton d'arrêt d'urgence	Permet de couper instantanément l'alimentation et d'arrêter tous les mouvements en cas de danger.

Table 3 : Principaux composants d'une machine CNC de perçage et leur fonction

1.6.5. Utilisation spécifique en perçage PCB

Lors de la fabrication de circuits imprimés, les CNC sont utilisées principalement pour :

- Percer les trous de vias (trous traversants métallisés).
- Percer les trous de montage des composants (résistances, connecteurs...).
- Usiner les contours du circuit (profilage final).

Le perçage doit être extrêmement précis pour garantir :

- La connectivité entre couches internes (dans les PCB multicouches).
- La soudabilité correcte des composants.
- L'alignement exact avec le masque de soudure et la sérigraphie.

1.6.6. Avantages et limites des CNC dans le domaine du PCB

Avantages :

- Répétabilité et précision micrométrique.
- Programmable via G-code ou interface graphique.
- Compatibilité avec fichiers Gerber.

Limites :

- Optimisation du trajet de perçage souvent négligée perte de temps [3].
- Bruit, usure des forets, énergie consommée en fonction du parcours.
- Complexité de calibration si le design change régulièrement

1.7. Le Fichier Gerber : Format Standard pour la Fabrication des PCB

Le fichier Gerber est le format industriel standard utilisé pour la communication des données de conception entre le concepteur du circuit imprimé (PCB) et le fabricant. Il contient les informations graphiques nécessaires à la fabrication de chaque couche du circuit.

1.7.1. Définition et rôle

Un fichier Gerber est un fichier vectoriel qui décrit :

- Les pistes conductrices (traces),
- Les pastilles de composants (pads),
- Les plans de masse (planes),
- Les zones à graver ou à épargner (solder mask),

- La sérigraphie (texte et symboles d'identification),
- Les couches mécaniques (profil de découpe).

Ces fichiers sont générés automatiquement à partir de logiciels de conception assistée par ordinateur (Altium, KiCad, Eagle, OrCAD, etc.) [1].

1.7.2. Types de fichiers Gerber

Chaque couche ou fonction du PCB possède son propre fichier, par exemple :

- TopCopper.GTL : couche cuivre supérieure.
- BottomCopper.GBL : couche cuivre inférieure.
- TopSolderMask.GTS : masque de soudure supérieur.
- DrillFile.TXT : fichier de perçage (Annexe C).
- BoardOutline.GKO : contour de la carte.

Ces fichiers sont souvent accompagnés d'un fichier de positionnement des composants (Pick and Place) et d'un Bill of Materials (BOM) [11].

1.7.3. Format Gerber RS-274X

Le format le plus couramment utilisé aujourd'hui est le Gerber RS-274X, qui est une évolution du format RS-274D :

- Supporte les couches multiples,
- Intègre les polices de texte, les formes et les flashes personnalisés,
- Permet une meilleure compatibilité entre logiciels et usines [1].

1.7.4. Importance dans la chaîne de fabrication

Le fichier Gerber est lu par les logiciels d'imbrication industrielle et les machines CNC pour :

- Graver les couches de cuivre,
- Percer les trous (via les fichiers Excellon),
- Aligner les couches,
- Positionner les composants automatiquement.

Une erreur dans un fichier Gerber (ex : mauvais alignement ou mauvaise échelle) peut entraîner une perte totale du lot de production. D'où l'importance de procéder à une vérification Gerber (Gerber viewer) avant fabrication [11].

1.8. Conclusion

Dans ce premier chapitre, nous avons exploré les fondements techniques nécessaires à la compréhension du processus de fabrication des cartes électroniques imprimées (PCB). Nous avons d'abord présenté les différents types de PCB, leurs structures, leurs applications industrielles et leurs spécificités technologiques.

Nous avons ensuite détaillé les étapes de conception et de fabrication d'un circuit imprimé, depuis l'exportation des fichiers Gerber jusqu'aux tests électriques finaux. Ce processus, à la fois précis et complexe, requiert une coordination étroite entre les outils logiciels de CAO, les formats de production standard (comme le Gerber), et les machines de fabrication.

Enfin, une attention particulière a été portée aux machines CNC, qui assurent des opérations de perçage avec une grande précision. Nous avons étudié leur fonctionnement, leurs composants, leurs avantages, ainsi que leurs limites, en particulier en ce qui concerne l'optimisation des trajectoires de perçage, un enjeu crucial dans les productions à grande échelle.

Ces constats soulignent la nécessité d'améliorer les performances des machines CNC par des techniques d'optimisation algorithmique, notamment dans le cas du perçage de multiples trous. Le chapitre suivant sera donc consacré à l'étude des approches d'optimisation modernes permettant de minimiser la distance totale parcourue, le temps de traitement, et les ressources utilisées, à travers des algorithmes tels que l'algorithme génétique, l'algorithme de la colonie de fourmis (ACO), et l'algorithme A* .

2. Chapitre 2 : Algorithmes d'Optimisation

2.1. Introduction

L'optimisation des processus industriels constitue aujourd'hui un enjeu majeur dans le domaine de la fabrication électronique. Dans le contexte du perçage de cartes électroniques à l'aide de machines CNC, un des objectifs essentiels est de réduire le temps d'usinage, minimiser les déplacements inutiles de la broche, et augmenter la productivité globale sans compromettre la précision.

Le perçage automatique d'une carte PCB implique souvent des centaines, voire des milliers de points à visiter. Le parcours optimal de l'outil entre ces points constitue une variante bien connue du Problème du Voyageur de Commerce (TSP - Traveling Salesman Problem), une problématique classique d'optimisation combinatoire NP-difficile [12].

Face à la complexité de ce problème, des approches heuristiques et métaheuristiques se sont imposées comme alternatives efficaces aux méthodes exactes. Ce chapitre s'intéresse particulièrement à trois familles d'algorithmes inspirés de l'intelligence artificielle :

- L'algorithme génétique (GA) : basé sur les principes de l'évolution naturelle et de la sélection.
- L'algorithme de la colonie de fourmis (ACO) : inspiré du comportement collectif des insectes sociaux.
- L'algorithme A* : une méthode de recherche de chemin utilisée pour trouver des trajets optimaux sur des graphes pondérés.

Chaque algorithme sera présenté en détail, accompagné d'une explication mathématique de son fonctionnement, d'un exemple d'application au problème de perçage, et d'une analyse comparative en termes de convergence, qualité de solution, et temps d'exécution.

2.2. Optimisation Combinatoire et Application au Trajet CNC

L'optimisation combinatoire est une branche de l'optimisation qui s'intéresse à la recherche de la meilleure solution parmi un ensemble discret et fini de possibilités [13]. Contrairement à l'optimisation continue, où les variables peuvent prendre une infinité de valeurs dans un

intervalle donné, les problèmes combinatoires manipulent des objets dénombrables comme des permutations, des arrangements ou des combinaisons.

Un problème combinatoire vise à identifier, parmi toutes les combinaisons possibles d'éléments, celle qui optimise une fonction objectif, souvent une fonction de coût, de temps ou de performance [14]. Ces problèmes apparaissent dans de nombreux domaines : logistique (TSP), planification industrielle, affectation de ressources, ou encore routage de données en informatique.

Dans le contexte de la fabrication électronique, le problème du trajet optimal d'une machine CNC est un exemple typique : il consiste à déterminer l'ordre optimal de perçage des trous sur un circuit imprimé (PCB), de manière à minimiser la distance totale parcourue par la broche. Ce problème devient rapidement complexe : pour n points de perçage, il existe $(n - 1)!$ combinaisons possibles. Cette croissance factorielle rend toute approche exhaustive inapplicable au-delà de quelques dizaines de points [15].

De manière générale, la plupart des problèmes d'optimisation combinatoire sont NP-difficiles, ce qui signifie qu'il est impossible de garantir une solution optimale en temps raisonnable lorsque la taille du problème augmente [15]. Pour cette raison, on a recours à des algorithmes heuristiques et surtout à des métaheuristiques, qui visent à produire des solutions proches de l'optimal dans des délais acceptables [16].

Dans ce projet, deux familles d'algorithmes métaheuristiques seront utilisées pour résoudre le problème du trajet de perçage :

- L'Algorithme de la Colonie de Fourmis (ACO), inspiré du comportement coopératif des fourmis à la recherche de nourriture.
- L'Algorithme Génétique (GA), fondé sur les principes de la sélection naturelle et de l'évolution biologique.

Les sections suivantes détailleront la logique de ces algorithmes, leurs formulations mathématiques et leur application spécifique à l'optimisation du perçage CNC.

2.3. L'Algorithme Génétique (AG)

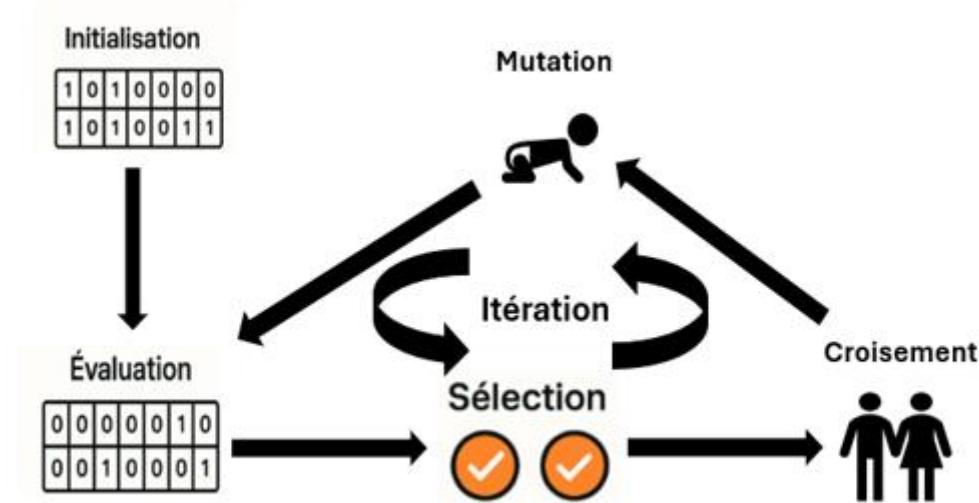


Figure 1: Étapes de l'algorithme génétique

2.3.1. Présentation générale

L'algorithme génétique (AG) est une technique d'optimisation métaheuristique inspirée du processus de sélection naturelle proposé par Charles Darwin. Il est utilisé pour résoudre des problèmes complexes où l'espace de recherche est immense et mal structuré, comme c'est le cas pour le TSP ou l'optimisation du trajet de perçage d'une machine CNC [17].

Un AG simule une évolution biologique artificielle, en manipulant une population de solutions candidates appelées individus ou chromosomes. Ces individus évoluent génération après génération sous l'effet de trois opérateurs principaux : la sélection, le croisement (crossover) et la mutation.

2.3.2. Représentation du problème

Dans notre cas, une solution candidate est représentée par une permutation des points de perçage (ex : ordre de visite des trous).

Par exemple, pour un PCB avec 6 trous, un individu peut être : [3, 1, 6, 2, 4, 5].

La fonction objectif associée (fitness function) peut être la distance totale parcourue, notée :

$$f(x) = \sum_{i=1}^{n-1} d(p_i, p_{i+1}) + d(p_n, p_1)$$

où

$d(p_i, p_{i+1})$ est la distance euclidienne entre deux points successifs [18].

2.3.3. Étapes de l'algorithme génétique

Initialisation : Génération aléatoire d'une population initiale de solutions (généralement entre 50 et 200 individus).

Évaluation (fitness) : Chaque individu est évalué selon sa distance totale. Plus la distance est courte, plus le score est élevé (fitness = 1/distance).

Sélection : On sélectionne les meilleurs individus selon des méthodes comme :

- Roulette Wheel Selection,
- Tournament Selection.

Croisement (crossover) : Combinaison de deux parents pour produire de nouveaux individus.

Méthodes classiques : PMX (Partially Mapped Crossover), OX (Order Crossover).

Mutation : Modification aléatoire d'un individu (ex : inversion de deux points) pour maintenir la diversité génétique.

Remplacement : Une nouvelle population est formée à partir des parents et des enfants.

Convergence : L'algorithme s'arrête après un nombre fixe de générations ou lorsque la population ne s'améliore plus[19].

2.3.4. Exemple de croisement (Order Crossover - OX)

Parent 1 : [1 2 3 4 5 6]

Parent 2 : [4 3 6 2 1 5]

- On choisit un segment aléatoire, ex : [3 4] dans Parent 1 → [3 4]
- On copie ces positions dans l'enfant

- Puis on complète avec les valeurs restantes de Parent 2 dans l'ordre où elles apparaissent.

2.3.5. Avantages et limites

Avantages :

- Très efficace pour les grands espaces de recherche.
- Adaptable à plusieurs types de problèmes (multi-objectifs, contraintes).
- Facilement parallélisable.

Limites :

- Convergence parfois lente.
- Risque de stagnation dans un optimum local.
- Nécessite un réglage précis des paramètres (taille de population, taux de mutation...).

2.3.6. Application à l'optimisation CNC

Dans ce projet, l'algorithme génétique est utilisé pour :

- Générer un ordre optimal de perçage.
- Réduire la distance totale et le temps d'usinage.
- Comparer ses performances à d'autres méthodes (ACO, A*).

2.4. L'Algorithme de la Colonie de Fourmis (ACO)

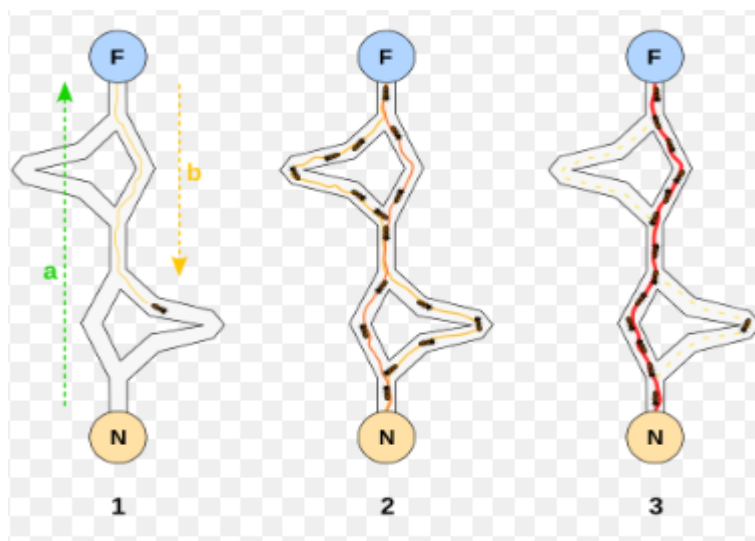


Figure 2 : Les étapes de la recherche de nourriture chez les fourmis.

2.4.1. Introduction et inspiration naturelle

L'algorithme ACO (Ant Colony Optimization) est une métaheuristique inspirée du comportement coopératif des fourmis pour rechercher de la nourriture [20]. En milieu naturel, les fourmis déposent des phéromones sur leur chemin. Plus un chemin est fréquenté et court, plus il est renforcé par les phéromones, ce qui oriente les autres fourmis vers ce chemin.

Ce principe d'intelligence collective a été modélisé par Marco Dorigo dans les années 1990 pour résoudre des problèmes combinatoires complexes, en particulier le TSP, ce qui en fait une méthode adaptée à l'optimisation du trajet de perçage sur un PCB.

2.4.2. Principes de base de l'ACO

L'algorithme repose sur une population d'agents (fourmis virtuelles) qui explorent différentes solutions en parallèle. Chaque fourmi construit un chemin (solution candidate) en se basant sur deux facteurs [21]:

- ✓ Quantité de phéromone τ (tau) : indique la préférence collective.
- ✓ Attractivité heuristique η (eta) : généralement inverse de la distance entre deux points.

Le choix de la prochaine position est probabiliste, selon la formule :

$$P_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \text{non visité}} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}, & \text{si } j \text{ non visité} \\ 0, & \text{sinon} \end{cases}$$

- τ_{ij} : intensité de la phéromone sur l'arête (i,j)
- $\eta_{ij} = \frac{1}{d_{ij}}$: visibilité (inverse de la distance)
- α : influence des phéromones
- β : influence de la visibilité

2.4.2.1. Mise à jour des phéromones :

Après que toutes les fourmis ont construit leur solution, les phéromones sont mises à jour selon

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k$$

- ρ : taux d'évaporation ($0 < \rho < 1$)
- $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{si la fourmi } k \text{ a utilisé l'arête } (i, j) \\ 0, & \text{sinon} \end{cases}$
- Q : constante de dépôt
- L_k : longueur totale du chemin de la fourmi k

2.4.3. Étapes de l'algorithme ACO

- Initialisation des paramètres et des niveaux de phéromones.
- Chaque fourmi construit un chemin en visitant tous les points selon la règle de probabilité.
- Calcul du coût (longueur) de chaque chemin.
- Mise à jour globale des phéromones.
- Répétition sur un nombre de cycles fixé ou jusqu'à convergence.

2.4.4. Avantages et limites

Avantages :

- Métaheuristique robuste et parallèle.
- Bon équilibre entre exploration et exploitation.
- Adaptée à des environnements dynamiques (modification des points).

Limites :

- Sensible aux paramètres α , β , ρ .
- Temps de convergence parfois élevé sans ajustements.

2.4.5. Application au trajet de perçage CNC

Dans le contexte du perçage automatique de PCB :

- Chaque point représente un trou à percer.
- Les fourmis explorent les ordres de perçage possibles.
- L'algorithme tend vers une séquence minimisant la distance totale parcourue par la broche CNC.

- L'ACO est particulièrement efficace pour éviter les optima locaux et explorer intelligemment l'espace de recherche.

2.5. L'Algorithme A* (A Star)

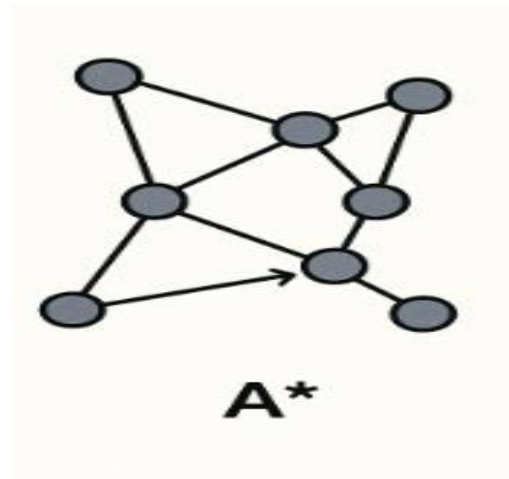


Figure 3 : L'Algorithme A*

2.5.1. Présentation générale

L'algorithme A* est une méthode de recherche informée qui permet de trouver le plus court chemin entre deux points dans un graphe pondéré. Il est largement utilisé dans les systèmes de navigation, l'intelligence artificielle des jeux vidéo, et l'optimisation logistique [22].

A* combine deux éléments clés :

- Le coût réel pour atteindre un nœud ($g(n)$),
- Une fonction heuristique estimant le coût restant pour atteindre le but ($h(n)$).

La fonction totale à minimiser est :

$$f(n) = g(n) + h(n)$$

où :

- $g(n)$: coût cumulé depuis le point de départ,
- $h(n)$: estimation heuristique du coût jusqu'à l'arrivée,

- $f(n)$: coût total estimé du chemin passant par le nœud n .

2.5.2. Fonctionnement de l'algorithme

1. On initialise une file ouverte (open list) avec le nœud de départ.
2. On sélectionne le nœud ayant la plus petite valeur de $f(n)$.
3. On explore ses voisins et on met à jour leurs valeurs de $g(n)$ et $h(n)$.
4. Si le but est atteint, le chemin est reconstruit.
5. Sinon, le processus se répète jusqu'à ce que la file ouverte soit vide ou que la cible soit atteinte.

2.5.3. Heuristique admissible

L'heuristique $h(n)$ doit être :

- Admissible : elle ne surestime jamais le coût réel.
- Consistante : elle respecte l'inégalité triangulaire.

Dans le cas du perçage CNC, on peut utiliser : $h(n) = \min(d(n, k))$ avec $k \in T$

où T est l'ensemble des trous non encore visités, et $d(n, k)$ est la distance euclidienne [23].

2.5.4. Avantages et limites

Avantages :

- Rapide pour les petits graphes.
- Reproductible et déterministe.
- Garantit le plus court chemin si $h(n)$ est admissible.

Limites :

- Ne s'adapte pas bien à des problèmes globaux comme le TSP.
- Moins efficace pour l'exploration combinatoire.
- Nécessite une fonction heuristique bien définie.

2.5.5. Application à l'optimisation du trajet de perçage

A* est rarement utilisé pour le TSP complet, car il n'est pas conçu pour explorer toutes les permutations possibles. Cependant, il peut être appliqué de manière greedy (gloutonne) pour :

- Trouver un chemin optimisé localement entre chaque point.
- Construire progressivement une séquence de perçage sous-optimale mais efficace.

Dans ce projet, A* est utilisé comme comparatif aux algorithmes GA et ACO, notamment pour sa rapidité à générer des solutions simples et déterministes .

2.6. Modélisation de Monte Carlo

2.6.1. Définition générale

La modélisation de Monte Carlo est une méthode de simulation probabiliste qui repose sur l'échantillonnage aléatoire massif pour évaluer le comportement d'un système complexe. Cette méthode est utilisée lorsqu'un système comporte trop de variables ou d'incertitudes pour être modélisé analytiquement [24].

Le principe consiste à exécuter un grand nombre de simulations aléatoires afin d'estimer des indicateurs globaux (temps moyen, variance, probabilité d'occurrence, etc.). Elle est couramment utilisée en finance, physique des particules, gestion des risques, et de plus en plus en ingénierie industrielle [25].

2.6.2. Intérêt pour l'optimisation du perçage CNC

Dans le contexte de ce projet, la méthode de Monte Carlo peut être utilisée pour :

- Tester la robustesse d'un algorithme d'optimisation (AG, ACO, A*) face à des variations aléatoires
- dans la position des points (simulation d'erreur de fabrication),
- dans les paramètres de la machine (vitesse, accélération, tolérances).
- Évaluer la performance moyenne d'une stratégie de perçage sur un grand nombre de PCB générés aléatoirement.
- Explorer la sensibilité du système à certaines configurations (par exemple : densité de points, distribution spatiale).

2.6.3. Procédure de simulation Monte Carlo

- Définir l'espace de simulation : nombre de points, plage de coordonnées, paramètres CNC.
- Générer aléatoirement un grand nombre d'instances (ex : 1000 PCB de 20 trous chacun).
- Appliquer l'algorithme d'optimisation à chaque instance.
- Mesurer des indicateurs : distance moyenne parcourue, écart-type, taux de convergence.
- Analyser statistiquement les résultats pour évaluer la stabilité et la performance de l'algorithme.

2.6.4. Exemple simplifié

Supposons qu'on génère 500 configurations aléatoires de PCB à 30 points. Sur ces 500 simulations :

- L'AG donne une moyenne de 185 mm parcourus, avec un écart-type de 12 mm.
 - L'ACO atteint 190 mm de moyenne, mais avec un écart-type plus faible.
 - A* varie fortement et échoue à optimiser dans 30 % des cas.
- La méthode de Monte Carlo permet ici de comparer les performances globales et la robustesse des algorithmes.

2.6.5. Limites

- Temps de calcul élevé pour des milliers de simulations.
- Qualité dépendante du modèle de génération aléatoire.
- Ne donne pas de solution exacte, mais une estimation statistique.

2.7. Conclusion

Ce chapitre a permis de poser les fondements théoriques nécessaires à l'optimisation du trajet de perçage dans les systèmes de fabrication assistée par ordinateur, en particulier les machines CNC appliquées aux cartes PCB. En modélisant ce problème comme une instance du Problème du Voyageur de Commerce (TSP), nous avons mis en évidence sa nature combinatoire et sa

complexité NP-difficile, rendant inadaptées les approches exactes pour des tailles de problème réalistes.

Afin d'y répondre, nous avons étudié trois approches métaheuristiques :

- L'algorithme génétique, tiré de la théorie de l'évolution, efficace pour explorer un grand espace de solutions.
- L'algorithme de la colonie de fourmis, basé sur une optimisation distribuée par renforcement de traces.
- L'algorithme A*, plus simple, mais utile dans une perspective de construction de trajectoires partielles optimisées.

Nous avons également introduit la méthode de Monte Carlo comme outil de simulation utile pour évaluer la performance globale des algorithmes à travers des scénarios aléatoires, apportant une dimension statistique à l'analyse.

Les connaissances acquises dans ce chapitre serviront de base à la mise en œuvre pratique des algorithmes dans une interface MATLAB GUI, qui fera l'objet du chapitre suivant.

3. Chapitre 3: Fenêtre graphique MATLAB et simulation

3.1. Introduction

Ce chapitre présente la réalisation pratique de l'interface graphique (GUI) développée sous MATLAB dans le cadre de l'optimisation du trajet de perçage pour une machine CNC. L'interface permet de charger un fichier de points (extraits d'un fichier Gerber), d'appliquer différents algorithmes d'optimisation (Génétique, ACO et A*), et d'afficher les résultats sous forme graphique et numérique. Elle constitue le lien entre l'utilisateur et les algorithmes d'optimisation.

3.2. Outils et environnement de développement

Le développement de l'interface a été effectué à l'aide du logiciel MATLAB R2013a, une version stable et compatible avec la plupart des systèmes CNC utilisés dans les laboratoires. MATLAB est un environnement de calcul numérique qui offre des outils puissants pour la visualisation, le traitement de données et l'implémentation d'algorithmes.

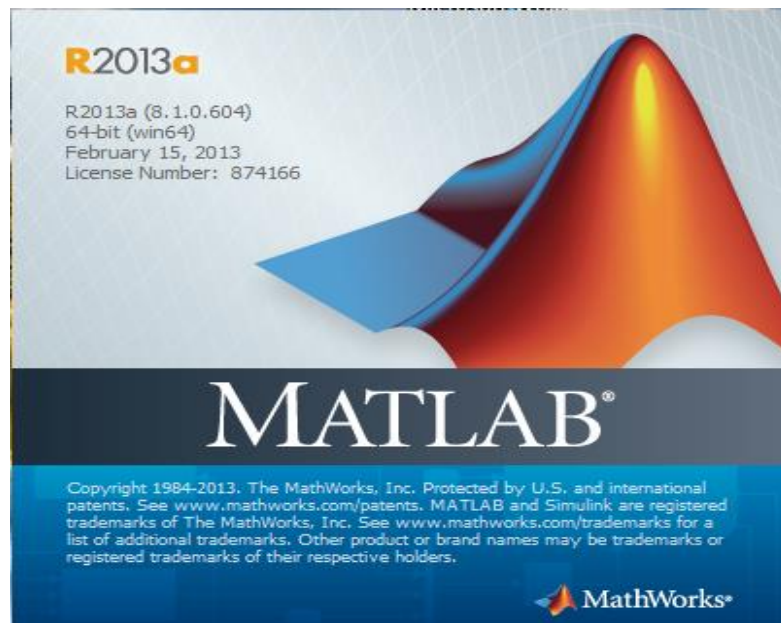


Figure 4 : Écran de démarrage de MATLAB R2013a utilisé pour le développement de l'interface

Pour la conception de l'interface utilisateur, nous avons utilisé GUIDE (Graphical User Interface Development Environment), un module intégré de MATLAB. GUIDE permet de créer des interfaces graphiques interactives de manière intuitive à travers un éditeur visuel. Il offre les composants standards tels que :

- Push Button (bouton-poussoir) : pour déclencher une action (ex. : lancer un algorithme).
- Edit Text (champ de texte) : pour afficher ou saisir des données numériques.
- Axes : zone d'affichage graphique pour les tracés 2D.
- Static Text (texte statique) : pour indiquer des libellés et résultats.
- Popup Menu et Panels : pour organiser les options de sélection et structurer l'interface.

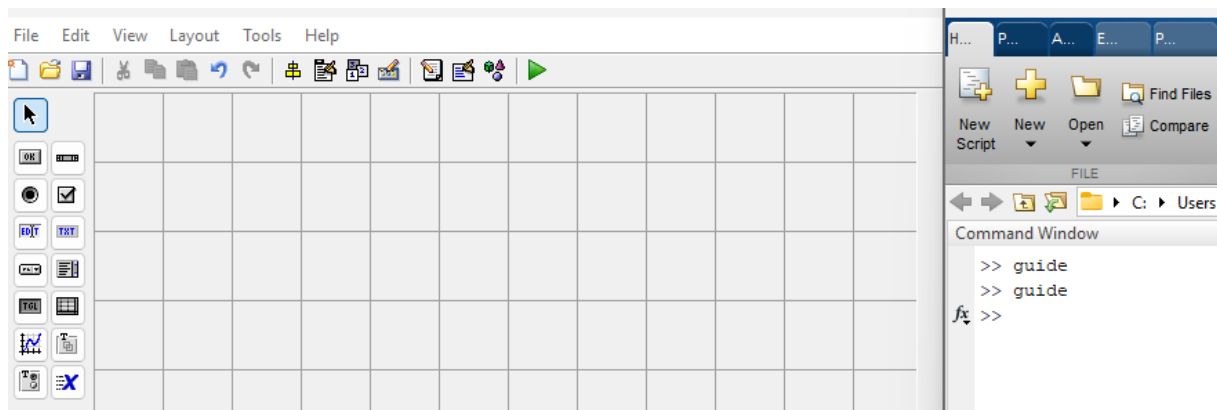


Figure 5 : Interface de conception graphique GUIDE dans MATLAB

Les scripts associés sont automatiquement liés à chaque élément graphique via des callbacks (fonctions de rappel). Ces callbacks sont écrits en langage MATLAB (.m) et permettent de gérer l'interaction entre l'utilisateur et les algorithmes intégrés dans l'interface.

3.3. Structure générale de l'interface

L'interface graphique a été conçue de manière ergonomique et structurée afin de faciliter l'utilisation par l'opérateur. Elle est divisée en plusieurs sections fonctionnelles bien distinctes, comme illustré dans l'image ci-dessous :

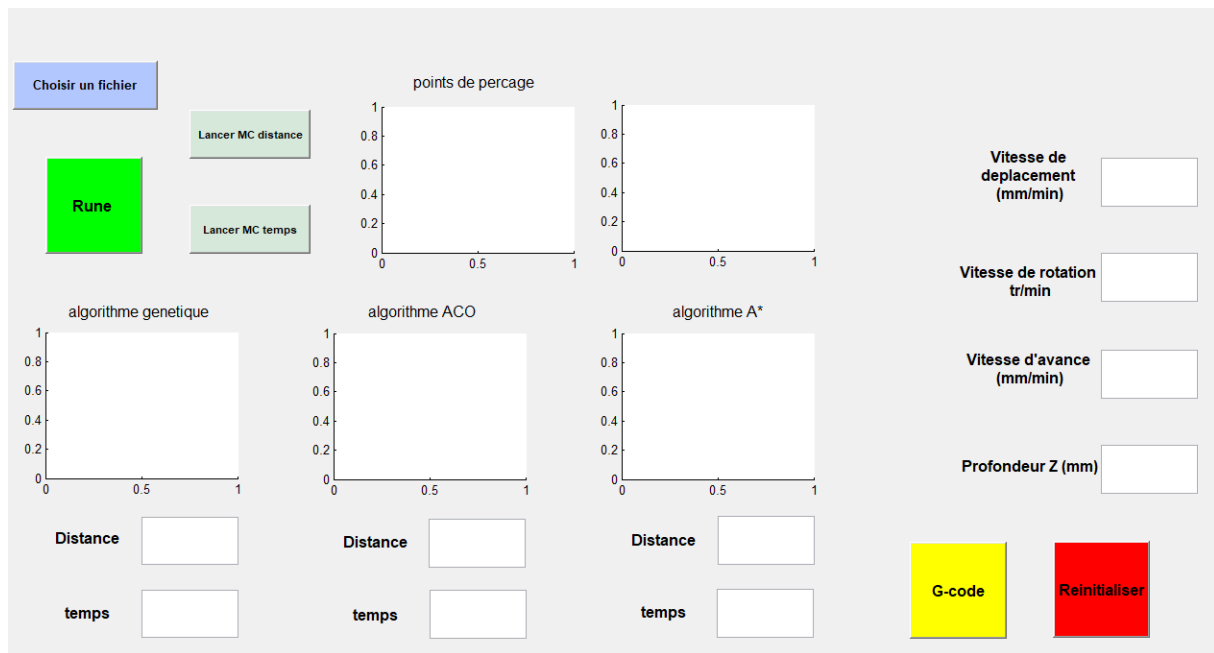


Figure 6 : Vue initiale de l'interface graphique MATLAB avant exécution

Les composants principaux de l'interface sont :

- **Zone de chargement du fichier** : en haut à gauche, le bouton "Choisir un fichier" permet d'importer un fichier texte contenant les coordonnées des points de perçage.
- **Bouton principal "Rune"** : exécute le traitement principal une fois que tous les paramètres sont chargés et configurés.
- **Boutons Monte Carlo** : "Lancer MC distance" et "Lancer MC temps" permettent de lancer des simulations statistiques afin de visualiser la variation des résultats sur plusieurs exécutions aléatoires.
- **Zone d'affichage des points** : les trois graphiques en haut au centre permettent de visualiser les points de perçage ainsi que les différents trajets générés.
- **Zones d'exécution des algorithmes** : trois zones d'affichage avec graphiques et champs de résultat (distance, temps) pour chaque algorithme (Génétique, ACO, A*).
- **Champs de saisie des paramètres** : sur la partie droite de l'interface, l'utilisateur peut entrer les valeurs nécessaires à la simulation :
 - Vitesse de déplacement (mm/min)
 - Vitesse de rotation (tr/min)
 - Vitesse d'avance (mm/min)
 - Profondeur de perçage Z (mm)

- **Boutons de sortie :** en bas à droite, le bouton "G-code" (jaune) permet d'exporter le trajet optimisé sous forme de fichier G-code. Le bouton "Réinitialiser" (rouge) permet de remettre l'interface à zéro.

Cette structure modulaire permet à l'utilisateur de suivre un flux de travail logique allant du chargement des données à l'obtention du résultat final optimisé.

3.4. Chargement et traitement des données

L'une des premières étapes essentielles dans le processus d'optimisation du parcours de perçage consiste à charger les coordonnées des trous à percer à partir d'un fichier externe. Cette étape est gérée à travers un bouton intitulé « Choisir un fichier » dans l'interface MATLAB développée.

3.4.1. Format des fichiers acceptés

L'interface accepte des fichiers contenant des coordonnées au format texte, (de fichiers .txt structurés manuellement ou exportés depuis un logiciel de CAO).

Les données sont généralement représentées sous la forme de paires (X,Y), exprimées en millimètres. Chaque ligne du fichier contient les coordonnées d'un trou (Annexe C)

Un traitement préalable peut être appliqué pour extraire uniquement les valeurs numériques, en supprimant les préfixes « X: » et « Y: », grâce à des expressions régulières ou des fonctions de découpage de chaînes dans MATLAB (regexp, str2double, etc.).

```
fid = fopen(nomFichier, 'r');  
data = textscan(fid, '%f %f');  
fclose(fid);
```

Figure 7 : Lecture des coordonnées de perçage à partir d'un fichier texte en MATLAB

Les vecteurs de coordonnées X et Y sont alors extraits et stockés dans la mémoire de l'application, pour être utilisés dans les étapes suivantes d'optimisation.

3.4.2. Affichage des points de perçage

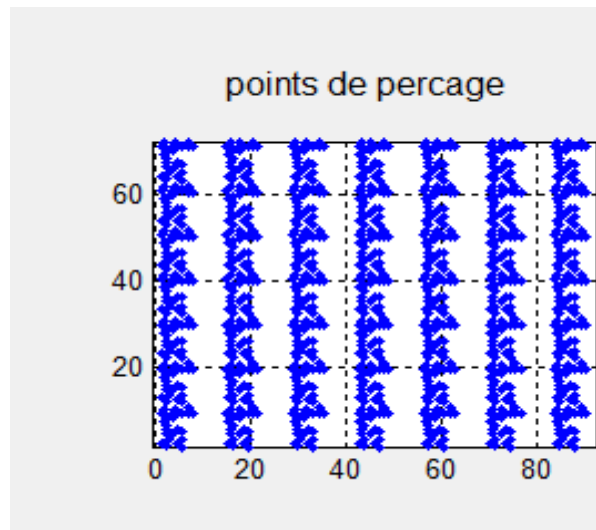


Figure 8 : Affichage des points de perçage extraits du fichier Gerber

Une fois les coordonnées extraites, un graphe est généré automatiquement dans l'axe central supérieur intitulé « points de perçage », sous forme d'un nuage de points bleus représentant les emplacements physiques des trous sur la carte PCB.

L'affichage se fait via une commande de type :

```
plot(X, Y, 'b.');
```

```
axis equal;
```

```
title('Points de perçage');
```

Figure 9 : Code MATLAB pour l'affichage des points de perçage

Cette visualisation constitue la base de référence pour l'optimisation des parcours, qui s'appliquera sur ces mêmes points dans les sections ultérieures.

3.5. Implémentation des algorithmes

Dans le cadre de ce projet, trois algorithmes d'optimisation ont été implémentés pour résoudre le problème du trajet de perçage (équivalent au problème du voyageur de commerce) :

- l'algorithme génétique (GA),
- l'optimisation par colonie de fourmis (ACO),
- et l'algorithme A*.

Ces algorithmes ont été intégrés dans l'interface MATLAB développée et appliqués aux coordonnées des trous extraites des fichiers d'entrée. Leur objectif est de fournir un ordre de parcours optimisé, minimisant la distance totale (ou le temps de perçage simulé).

Les codes complets sont disponibles dans les annexes A.1 à A.3.

3.5.1. Algorithme génétique (GA)

L'algorithme génétique repose sur une approche stochastique inspirée des processus évolutifs. Il génère une population initiale de parcours aléatoires, puis applique successivement des opérateurs de sélection, de croisement (crossover) et de mutation sur plusieurs générations.

Les principaux paramètres sont :

- Taille de population : 50,
- Nombre de générations : 100,
- Taux de mutation : 5 %.

Chaque individu représente un ordre de parcours, et sa "valeur" est déterminée par la distance totale du chemin (calculée à l'aide d'une matrice de distances euclidiennes). L'algorithme converge progressivement vers un parcours optimal ou quasi-optimal.

Le code détaillé est présenté en Annexe A.1.

3.5.2. Algorithme ACO (Ant Colony Optimization)

L'algorithme ACO s'inspire du comportement des colonies de fourmis dans la nature, où des phéromones sont déposées pour marquer les meilleurs chemins vers une source de nourriture.

Dans cette implémentation :

- Plusieurs fourmis construisent des solutions en parallèle,
- Les probabilités de choix dépendent de la quantité de phéromone τ et de la visibilité (l'inverse de la distance) η ,
- Un mécanisme d'évaporation est appliqué pour éviter la stagnation.

Paramètres principaux :

- Nombre de fourmis : 30,
- Nombre d'itérations : 100,
- $\alpha=1$, $\beta=5$, $\rho=0.5$, $Q=100$.

Après chaque itération, les phéromones sont mises à jour proportionnellement à la qualité des solutions produites (chemins les plus courts renforcés).

L'implémentation complète de cet algorithme est présentée en Annexe A.2.

3.5.3. Algorithme A*

L'algorithme A* est une méthode déterministe de recherche de chemin, souvent utilisée dans les systèmes embarqués et les jeux vidéo. Il a été ici adapté pour trouver un ordre efficace de visite des points selon une heuristique simple : la distance euclidienne directe entre les points.

Son avantage principal est sa rapidité d'exécution, mais il ne garantit pas toujours une solution optimale, surtout pour des parcours nécessitant un retour au point initial (comme dans un TSP classique).

Le code correspondant est présenté en Annexe A.3.

3.5.4. Intégration dans l'interface

Chaque algorithme peut être lancé indépendamment depuis l'interface utilisateur MATLAB à l'aide du bouton principal « Rune ». Les résultats sont affichés en temps réel :

- le parcours calculé est tracé graphiquement,
- la distance totale est affichée,
- le temps estimé de perçage est calculé dynamiquement à partir des vitesses définies dans l'interface.

Cette structure permet une comparaison visuelle et numérique directe entre les différentes méthodes.

3.6. Visualisation des résultats

L'une des fonctionnalités clés de l'interface développée est la visualisation immédiate et intuitive des résultats produits par les différents algorithmes d'optimisation. Cela inclut non seulement le tracé graphique des parcours optimisés, mais aussi l'affichage des indicateurs numériques tels que la distance totale, le temps de perçage estimé, et les courbes de convergence.

3.6.1. Affichage des parcours optimisés

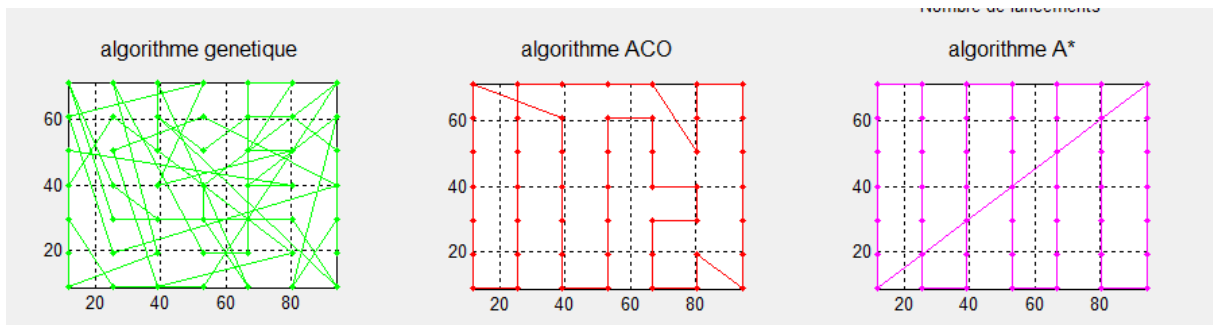


Figure 10 : Parcours optimisés par les trois algorithmes (Génétique, ACO, A*)

Pour chaque algorithme exécuté (Génétique, ACO, A*), l'interface MATLAB affiche dynamiquement le chemin calculé reliant les différents points de perçage. Ces parcours sont tracés dans des axes graphiques séparés :

- Axe gauche pour l'algorithme génétique,
- Axe central pour l'algorithme ACO,
- Axe droit pour l'algorithme A*.

Chaque point du parcours est relié par une ligne, avec des cercles (ou marqueurs) indiquant les positions de perçage. Cette représentation permet à l'utilisateur de comparer visuellement la qualité des trajets générés.

3.6.2. Indicateurs numériques affichés

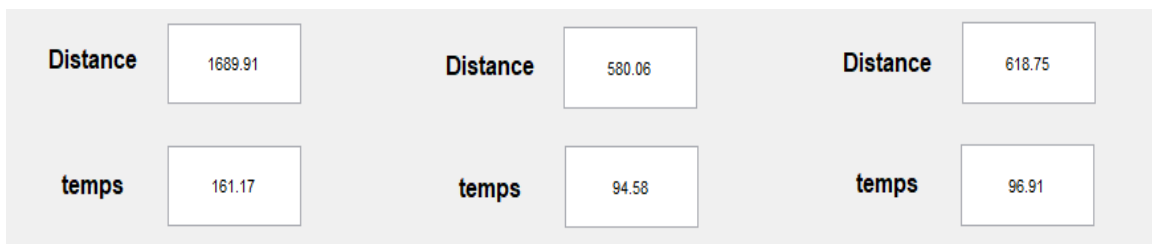


Figure 11 : Distances et temps d'usage obtenus pour chaque algorithme

Pour chaque méthode, deux valeurs importantes sont affichées dans l'interface :

- Distance totale du parcours (en mm) : calculée à partir de la somme des distances entre tous les points successifs du chemin, y compris le retour au point de départ (circuit fermé).

- Temps estimé d'usinage (en secondes) : calculé en fonction de la distance parcourue et des vitesses définies dans l'interface (avance, déplacement, rotation). Cela représente une simulation réaliste du temps que prendrait le trajet sur une machine CNC.

Ces valeurs sont mises à jour en temps réel après chaque exécution d'algorithme.

3.6.3. Graphe de convergence

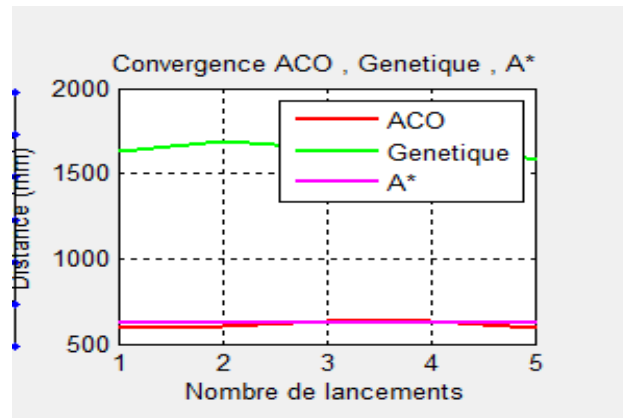


Figure 12 : Convergence des algorithmes

Dans la partie supérieure de l'interface, un graphique de convergence affiche l'évolution de la distance obtenue par chaque algorithme au fil de plusieurs lancements successifs. Ce graphique compare :

- la stabilité des résultats,
- la tendance à converger vers une solution optimale,
- la variabilité entre différentes exécutions.

Ce type de représentation est particulièrement utile pour visualiser la performance comparative entre ACO, GA et A* sur des instances réelles (comme les points extraits d'un fichier Gerber).

3.6.4. Interface centrée sur l'utilisateur

L'ensemble de ces visualisations est conçu pour fournir à l'utilisateur :

- une compréhension rapide des résultats,
- une capacité à interpréter visuellement la performance de chaque méthode,
- et une possibilité d'exporter le parcours le plus efficace pour l'utiliser dans un environnement de production CNC.

3.7. Simulation de l'usinage

L'une des fonctionnalités essentielles de l'interface MATLAB développée est la simulation du temps d'usinage correspondant au parcours généré par chaque algorithme. Cela permet d'évaluer non seulement l'efficacité des algorithmes sur le plan théorique (distance minimale), mais également leur pertinence en contexte réel de fabrication sur une machine CNC.

3.7.1. Données utilisées pour la simulation

Figure 13 : Paramètres technologiques saisis dans l'interface

Pour estimer la durée totale de l'opération de perçage, plusieurs paramètres sont pris en compte, tous configurables via l'interface graphique :

Paramètre	Description	Unité
Vitesse d'avance	Vitesse de perçage dans la matière	mm/min
Vitesse de déplacement	Vitesse de déplacement "à vide"	mm/min
Vitesse de rotation	Vitesse de rotation de la fraise	tr/min (indicatif)
Profondeur Z	Profondeur de descente pour chaque trou	mm

Table 4 : Paramètres de Perçage pour la Machine CNC

Ces paramètres sont saisis par l'utilisateur avant exécution de l'algorithme. Ils influencent directement le temps estimé.

3.7.2. Calcul du temps total

Le temps total d'usinage affiché dans l'interface est calculé de manière détaillée selon deux composantes principales :

3.7.2.1. Temps de déplacement sur le plan XY

Pour chaque algorithme, le temps de déplacement horizontal est donné par :

$$T_{xy} = D_totale / V_déplacement$$

où :

- D _totale : est la distance totale parcourue dans le plan XY,
- V _déplacement : est la vitesse de déplacement (mm/min).

3.7.2.2. Temps de descente/remontée en Z pour chaque trou

Pour chaque trou, la machine effectue une descente, une remontée, et une pause de 1 seconde.

Le temps Z est donné par :

$$T_z \text{ par trou} = (2 \times \text{profondeur} / V_avance) + (1 / 60)$$

Le temps total Z est donc :

$$T_z \text{ total} = nb_trous \times T_z \text{ par trou}$$

3.7.2.3. Temps d'usinage total

Le temps global est obtenu par la somme :

$$T_total = T_{xy} + T_z$$

Puis il est converti en secondes pour l'affichage dans l'interface :

$$T_affiché = T_total \times 60$$

Remarque : Ce modèle permet une estimation réaliste du temps de fabrication basé sur les paramètres définis par l'utilisateur. Il prend en compte à la fois les déplacements rapides et les cycles d'usinage verticaux associés à chaque trou.

Le script MATLAB correspondant à ce calcul est présenté dans l'annexe B.

3.7.3. Affichage du résultat

Une fois le parcours généré, le temps d'usinage est affiché automatiquement dans l'interface, pour chaque algorithme, sous la mention « temps ». Ce temps est exprimé en secondes, et tient compte de :

- la distance totale du parcours,
- la combinaison des vitesses définies,
- des temps de descente par trou.

Cette estimation donne à l'utilisateur une indication réaliste de la performance globale de la stratégie de perçage générée.

3.8. Exportation du G-code

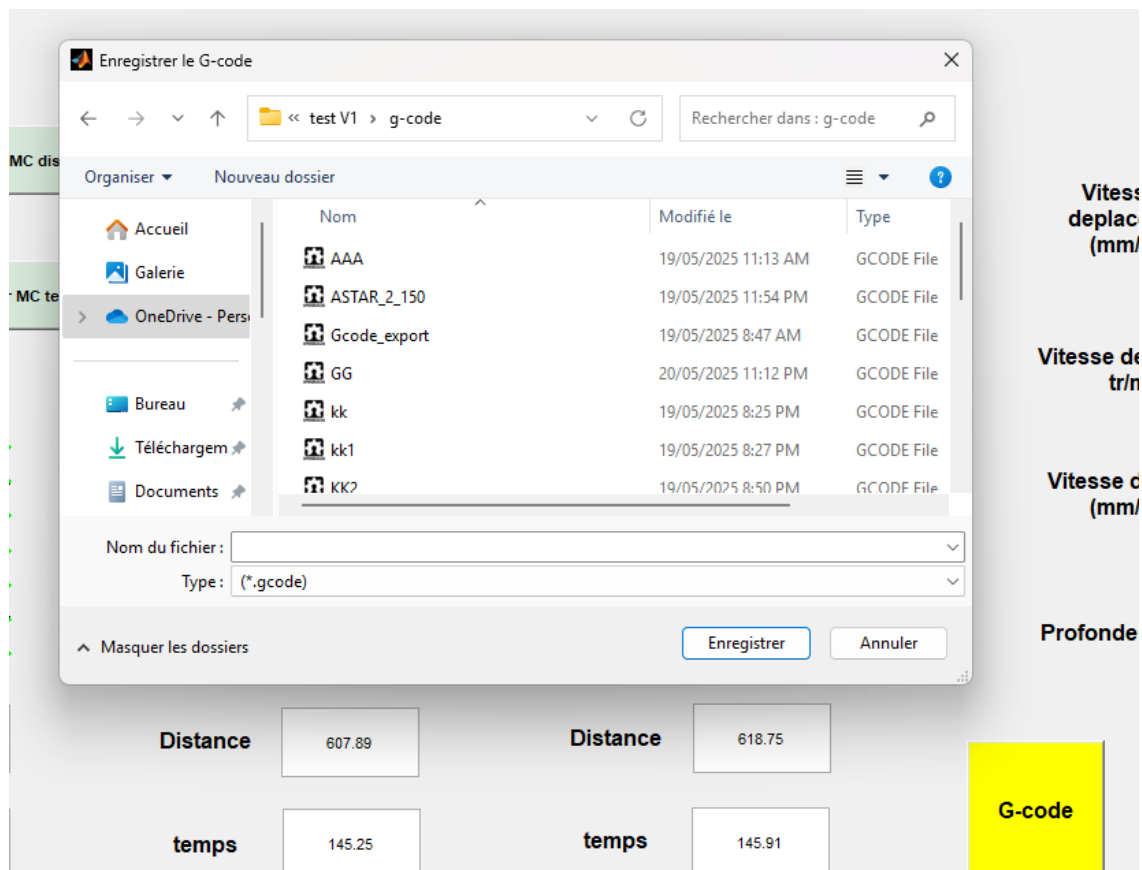


Figure 14 : Fenêtre d'enregistrement du fichier G-code

L'interface MATLAB développée permet à l'utilisateur d'exporter le parcours de perçage optimisé sous forme d'un fichier G-code, directement exploitable par une machine CNC. Cette fonctionnalité rend l'application non seulement analytique mais également prête à l'usage industriel.

3.8.1. Vérification des paramètres avant génération

Avant d'écrire le fichier G-code, l'interface procède à une vérification des paramètres de coupe, notamment :

- Vitesse d'avance (feedrate),
- Profondeur Z,
- Vitesse de déplacement rapide.

Si une valeur sort des plages recommandées, une boîte de dialogue d'avertissement s'affiche.

Exemple de message affiché à l'utilisateur :

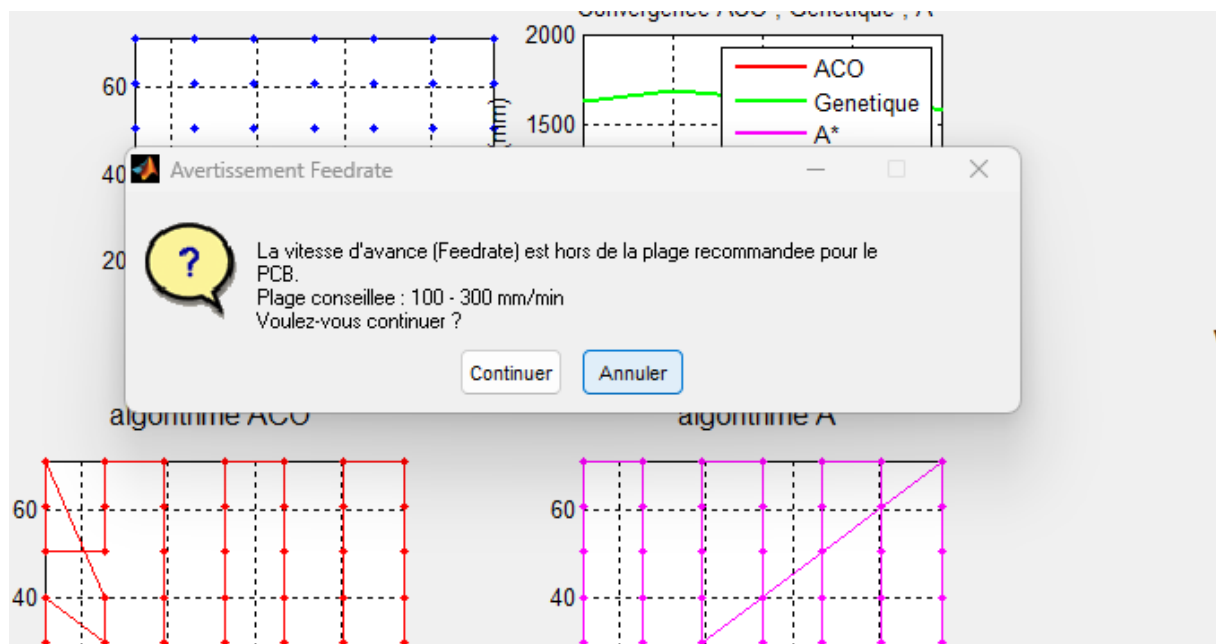


Figure 15 : Avertissement sur la plage de vitesse d'avance (Feedrate)

Ce mécanisme permet à l'utilisateur de corriger les valeurs dangereuses ou de confirmer s'il souhaite poursuivre en connaissance de cause.

3.8.2. Génération du fichier G-code

Une fois les paramètres validés, le fichier G-code est généré en respectant la syntaxe classique des commandes CNC :

G21 : unités en millimètres

G90 : positionnement absolu

G0 X Y : déplacement rapide

G1 Z-... F... : descente pour perçage

G1 Z+... F... : remontée

Chaque point du parcours est converti en instructions successives :

```
G0 X... Y...  
G1 Z-1.5 F250  
G1 Z5 F250
```

Figure 16 : Exemple d'instruction G-code générée pour le perçage

Les valeurs utilisées (Z, F, X, Y) dépendent des entrées utilisateur et du parcours optimisé sélectionné (ACO, GA, A*).

3.8.3. Sauvegarde et exportation

Le fichier G-code est sauvegardé localement dans le répertoire de l'utilisateur, avec une extension .gcode

Un message de confirmation est affiché après l'écriture réussie du fichier , Le code généré est ensuite prêt à être transféré vers une machine CNC via un logiciel d'interface tel que Candle, bCNC, ou toute autre plateforme compatible GRBL.

3.8.4. Robustesse de l'interface

L'exportation du G-code dans l'interface MATLAB est conçue pour être fiable, guidée et sécurisée. Elle repose sur plusieurs mécanismes de vérification et de sélection automatique visant à garantir la qualité et la sécurité du fichier généré.

Tout d'abord, l'utilisateur ne peut pas générer un fichier G-code sans avoir exécuté une méthode d'optimisation.

Ensuite, une vérification des paramètres critiques (vitesse d'avance, profondeur Z, etc.) est réalisée avant la génération.

Après validation des paramètres, le système procède à la sélection automatique du meilleur parcours, c'est-à-dire celui qui donne la distance ou le temps d'usinage le plus faible parmi les algorithmes exécutés (ACO, génétique ou A*).

Ce parcours optimal est ensuite utilisé pour générer le G-code, garantissant ainsi une exécution optimale sur la machine CNC. Une fenêtre de confirmation notifie à l'utilisateur le succès de l'opération et précise l'algorithme ayant généré le fichier, ainsi que son chemin d'enregistrement.

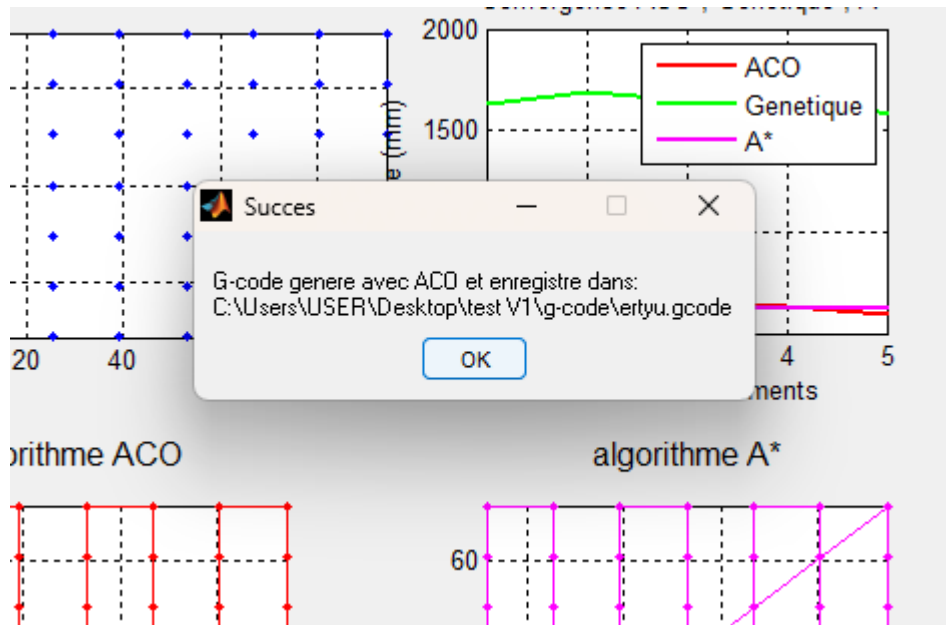


Figure 17 : Confirmation de génération réussie du G-code

Ce mécanisme permet à l'utilisateur de travailler en toute confiance, tout en minimisant les risques d'erreurs de configuration ou de sélection.

3.9. Conclusion

Dans ce chapitre, nous avons présenté en détail le développement de l'interface MATLAB permettant d'optimiser le parcours de perçage d'une machine CNC à partir de coordonnées extraites d'un fichier Gerber .

Nous avons décrit les différentes composantes fonctionnelles de cette interface, notamment :

- le chargement des points de perçage,
- l'implémentation de trois algorithmes d'optimisation (génétique, ACO, A*),
- l'affichage graphique des parcours générés,
- le calcul du temps estimé d'usinage,
- et l'exportation du G-code prêt à l'emploi pour la commande d'une machine CNC.

Des éléments de validation ont également été intégrés dans l'interface, tels que les alertes sur les paramètres critiques (vitesse d'avance, profondeur de coupe), garantissant une génération sécurisée du G-code.

Cette interface constitue un outil complet et interactif permettant non seulement d'évaluer la performance des algorithmes d'optimisation, mais également de passer à une application pratique et industrielle grâce à l'automatisation de la production du G-code.

Dans le chapitre suivant, nous procéderons à une analyse comparative fondée sur des simulations, en évaluant les performances des différents algorithmes sur divers jeux de données, et en analysant les résultats obtenus en termes de distance parcourue, de temps d'usinage et de stabilité des solutions.

4. Chapitre 4: Résultats et interprétations

4.1. Introduction

Ce chapitre présente une étude comparative basée sur des simulations des performances des trois algorithmes d'optimisation implémentés dans l'interface : l'algorithme génétique, ACO (Ant Colony Optimization) et A*.

Les tests sont effectués sur différents jeux de données, incluant des points de perçage réels extraits de fichiers Gerber.

Les critères de comparaison sont :

- la distance totale du parcours,
- le temps d'usinage estimé,
- la stabilité des algorithmes (via Monte Carlo),
- et la qualité du G-code généré.

4.2. Données de test et scénarios

Deux types de scénarios ont été utilisés :

Cas simulé : points générés aléatoirement (ex. 10, 30, 50 points).

Cas réel : points extraits depuis un fichier Gerber d'une carte PCB.

Les paramètres fixés pour la machine CNC sont :

- Vitesse de déplacement : 1000 mm/min
- Vitesse d'avance : 250 mm/min
- Vitesse de rotation : 15000 tr/min
- Profondeur Z : -1.5 mm

4.3. Analyse comparative des algorithmes

Dans le cadre de ce projet, trois algorithmes d'optimisation ont été appliqués à la résolution du problème du trajet de perçage d'une machine CNC. Il s'agit de :

- l'algorithme génétique,
- l'optimisation par colonie de fourmis (ACO),
- et l'algorithme A*.

Ces méthodes ont été testées sur les mêmes ensembles de données, et intégrées dans l'interface MATLAB développée à cet effet.

L'objectif de cette section est de présenter une comparaison approfondie de leurs performances selon plusieurs critères pertinents, à savoir :

- la distance totale parcourue,
- le temps d'usinage estimé,
- la stabilité des résultats (analyse de convergence),

Les résultats sont issus de simulations répétées sur des ensembles de points de perçage simulés et réels. Des visualisations graphiques (trajectoires optimisées, courbes de convergence, histogrammes) sont utilisées afin de permettre une évaluation à la fois qualitative et quantitative des performances de chaque algorithme.

4.3.1. Distance totale parcourue

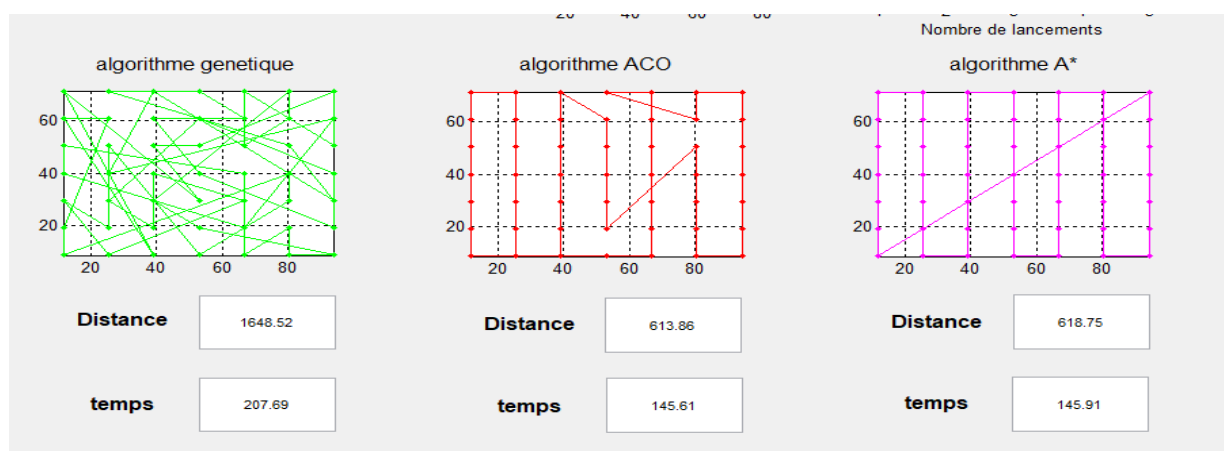


Figure 18 : Parcours optimisés à partir d'un fichier Gerber réel

L'un des critères fondamentaux pour évaluer la performance d'un algorithme d'optimisation de parcours est la distance totale du chemin généré. Plus cette distance est courte, plus le parcours est considéré comme efficace et optimisé, réduisant ainsi le temps de déplacement de la machine et l'usure mécanique.

Les résultats suivants ont été obtenus lors d'un test sur un jeu de données issu d'un fichier Gerber, contenant des coordonnées réelles de points de perçage

Algorithme	Distance totale (mm)
Génétique	1648.52
ACO	613.86
A*	618.75

Table 5 : Distance totale parcourue par chaque algorithme sur des données réelles issues d'un fichier Gerber

4.3.1.1. Analyse des résultats :

- L'algorithme ACO fournit la distance la plus courte, confirmant son efficacité sur ce jeu de données.
- L'algorithme A* donne un résultat très proche, bien qu'il soit déterministe et moins sophistiqué.
- L'algorithme génétique produit une distance beaucoup plus longue (environ 2,7 fois plus élevée que celle d'ACO), ce qui traduit une performance plus faible dans cette configuration, probablement liée à la convergence aléatoire .

4.3.1.2. Observation visuelle :

Les tracés graphiques illustrent clairement la différence :

Le parcours de l'algorithme génétique semble désordonné, avec de nombreux croisements.

Celui de l'ACO est plus structuré et optimisé.

Le parcours généré par A* est linéaire et cohérent, bien qu'il ne tienne pas forcément compte du retour au point initial dans sa version simplifiée

4.3.2. Temps d'usinage estimé

Outre la distance, un critère essentiel dans un contexte industriel est le temps total d'usinage estimé, c'est-à-dire la durée que mettrait la machine CNC pour réaliser le parcours proposé par chaque algorithme.

Ce temps est calculé à partir :

- de la distance parcourue en XY,
- des paramètres définis dans l'interface : vitesse d'avance, vitesse de déplacement rapide, profondeur Z, etc.
- et du nombre de trous, pris en compte pour simuler les cycles de descente/remontée en Z.

Les résultats suivants ont été obtenus pour les trois algorithmes :

Algorithme	Temps estimé (secondes)
Génétique	207.69
ACO	145.61
A*	145.91

Table 6 : Temps d'usinage estimé pour chaque algorithme sur des données réelles

Analyse :

Le temps d'usinage est directement corrélé à la distance : plus le chemin est court, plus la machine passe rapidement d'un trou à l'autre.

- ACO obtient ici le temps le plus court, ce qui confirme la qualité du parcours généré.
- A* arrive juste après, avec une différence négligeable (moins de 0.3 seconde).
- L'algorithme génétique, avec une distance bien plus longue, entraîne un temps d'usinage significativement plus élevé, ce qui peut être problématique en production réelle.

4.3.3. Analyse de convergence

Pour évaluer la stabilité et la robustesse des algorithmes face à la variation aléatoire (positions initiales, chemins choisis, etc.), une série de cinq exécutions indépendantes a été réalisée pour chaque méthode.

Le graphe suivant (Figure 20) illustre l'évolution de la distance totale obtenue pour chaque algorithme lors de ces différents lancements.

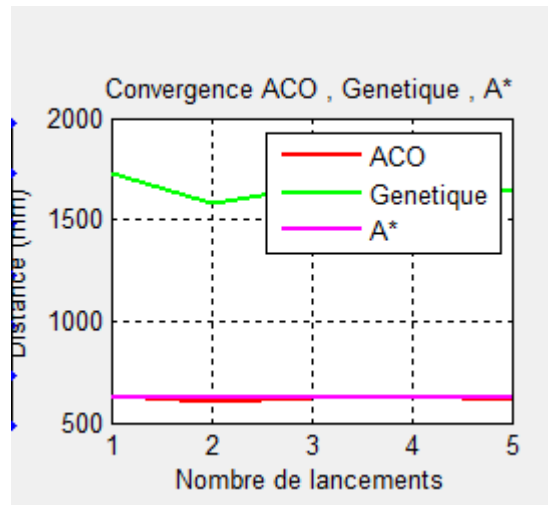


Figure 19 : Graphe de convergence des distances sur cinq exécutions

Observations :

- ACO (rouge) : montre une convergence régulière entre 600 et 650 mm avec de légères fluctuations. Cela reflète une bonne performance globale avec un comportement stable.
- A* (magenta) : reste extrêmement stable à chaque exécution (~618 mm), ce qui témoigne d'un fonctionnement déterministe fiable, même si la solution n'est pas toujours optimale.
- Génétique (vert) : montre une variabilité significative entre les exécutions. La distance passe de ~1800 mm à ~1600 mm, ce qui indique une convergence lente et une forte dépendance aux paramètres initiaux.

Pour mieux observer les différences entre ACO et A*, un zoom de la zone [500–750 mm] est présenté dans la figure suivante :

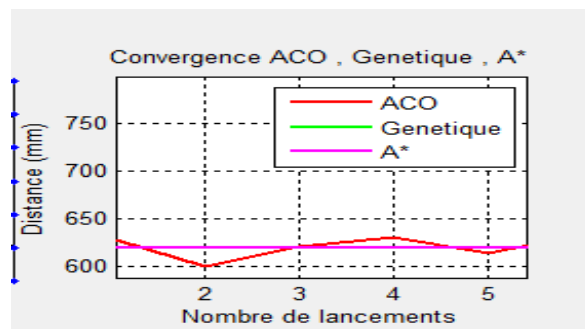


Figure 20 : Zoom sur la convergence d'ACO et A*

Ce zoom confirme que ACO est légèrement supérieur à A* en moyenne, bien que les deux restent proches dans cet exemple.

4.4. Analyse par simulation de Monte Carlo

Afin d'évaluer la robustesse et la variabilité des performances des trois algorithmes d'optimisation (génétique, ACO et A*), une simulation de Monte Carlo a été réalisée. Elle consiste à répéter l'exécution de chaque algorithme un grand nombre de fois (ici 50) sur un même ensemble de points (10 , 30 , 50 points de perçage), tout en conservant une part d'aléatoire dans l'initialisation ou la sélection.

Les résultats ont été analysés selon deux indicateurs :

- la distance totale du parcours généré,
- le temps d'usinage estimé, calculé comme dans la section 3.7.2.

4.4.1. Résultats sur 10 points

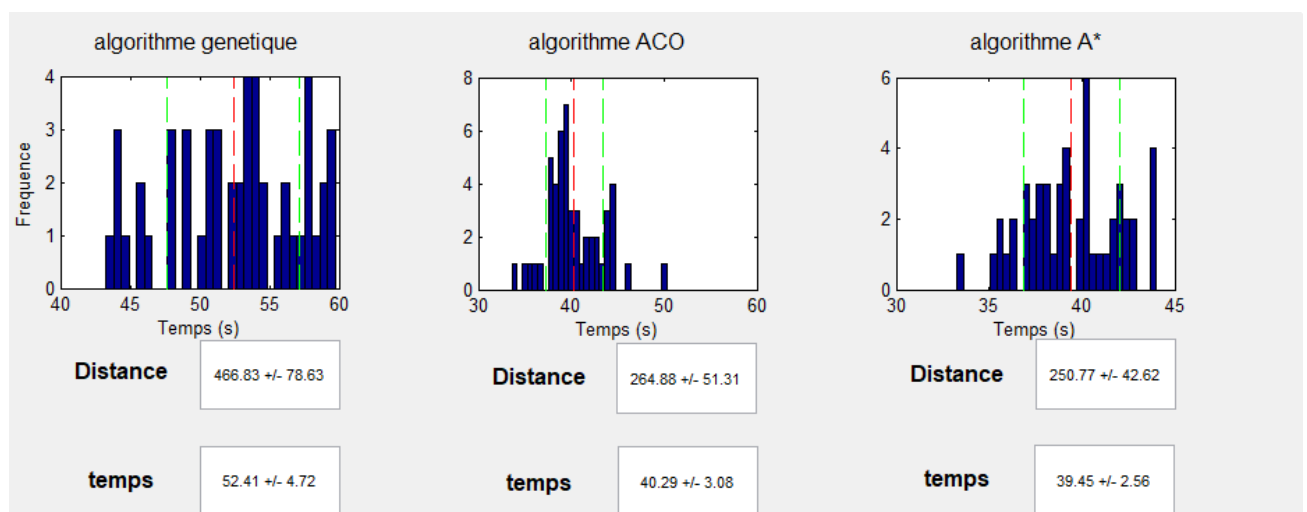


Figure 21 : Simulation Monte Carlo : histogrammes des temps d'usinage pour 10 points

4.4.1.1. Distance totale (mm)

Algorithme	Moyenne (mm)	Écart-type (mm)
Génétique	466.83	± 78.63
ACO	264.88	± 51.31
A*	250.77	± 42.62

Table 7 : Simulation Monte Carlo : statistiques sur la distance totale (10 points)

Analyse :

- L'algorithme A* montre la distance moyenne la plus faible, avec également la plus faible dispersion.
- ACO produit des distances légèrement supérieures mais reste relativement stable.
- L'algorithme génétique est le moins performant et le moins stable, avec une forte variabilité.

Ces distributions sont illustrées sous forme d'histogrammes, où l'on observe que :

- Les résultats de A* sont concentrés autour de la moyenne.
- ACO présente un étalement plus large mais mieux que le génétique.
- Génétique affiche une répartition très dispersée, ce qui traduit un manque de fiabilité.

4.4.1.2. Temps d'usinage estimé (s)

Algorithme	Moyenne (s)	Écart-type (s)
Génétique	52.41	± 4.72
ACO	40.29	± 3.08
A*	39.45	± 2.56

Table 8 : Simulation Monte Carlo : temps d'usinage estimé (10 points)

Analyse :

- A* conserve son avantage en temps, avec une estimation stable autour de 39.45 s.
- ACO est très proche, avec une moyenne légèrement supérieure mais très acceptable.
- Génétique est le plus lent et le plus variable.

Le temps d'usinage est proportionnel à la distance, ce qui confirme la cohérence du modèle de simulation mis en place.

4.4.2. Résultats sur 30 points

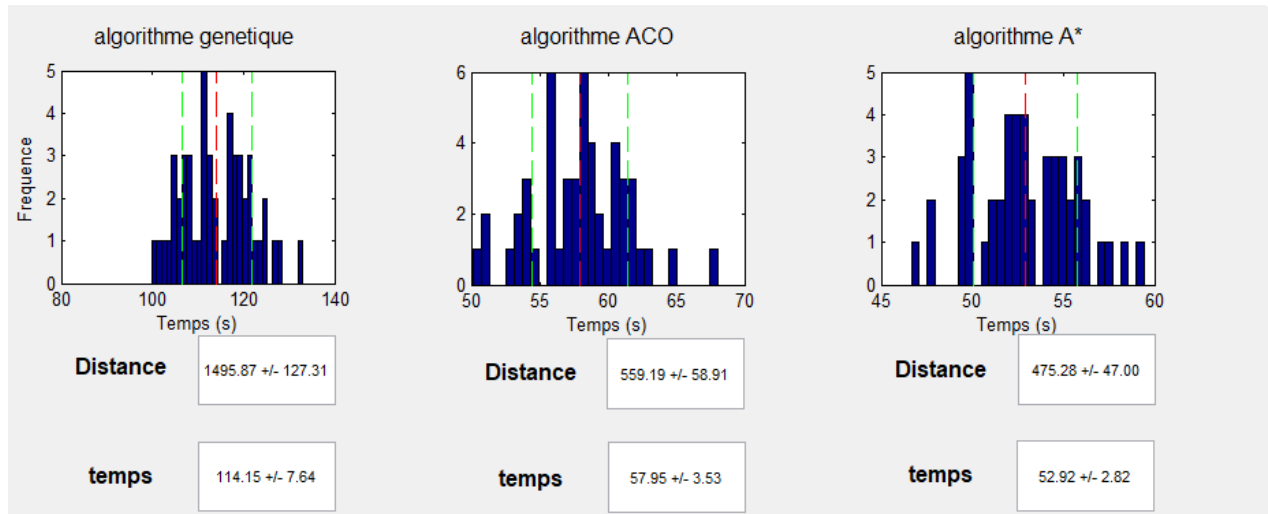


Figure 22 : Simulation Monte Carlo : histogrammes des temps d'usage pour 30 points

4.4.2.1. Distance totale (mm)

Algorithme	Moyenne (mm)	Écart-type (mm)
Génétique	1495.87	± 127.31
ACO	559.19	± 58.91
A*	475.28	± 47.00

Table 9 : Simulation Monte Carlo : statistiques sur la distance totale (30 points)

Analyse :

- L'ACO conserve de bonnes performances, mais A* le dépasse en moyenne, bien que légèrement.
- Le génétique reste le moins performant, avec un grand écart-type, ce qui indique une variabilité toujours marquée.
- L'écart entre les courbes commence à se stabiliser : les performances d'A* sont de plus en plus intéressantes à mesure que la taille du problème augmente légèrement.

4.4.2.2. Temps d'usinage estimé (s)

Algorithme	Moyenne (s)	Écart-type (s)
Génétique	114.15	± 7.64
ACO	57.95	± 3.53
A*	52.92	± 2.82

Table 10 : Simulation Monte Carlo : temps d'usinage estimé (30 points)

Analyse :

- Le temps est cohérent avec les distances : A* reste le plus rapide.
- ACO montre une bonne stabilité avec un écart-type modéré.
- Génétique reste le plus long, et aussi le plus dispersé.

4.4.3. Résultats sur 50 points

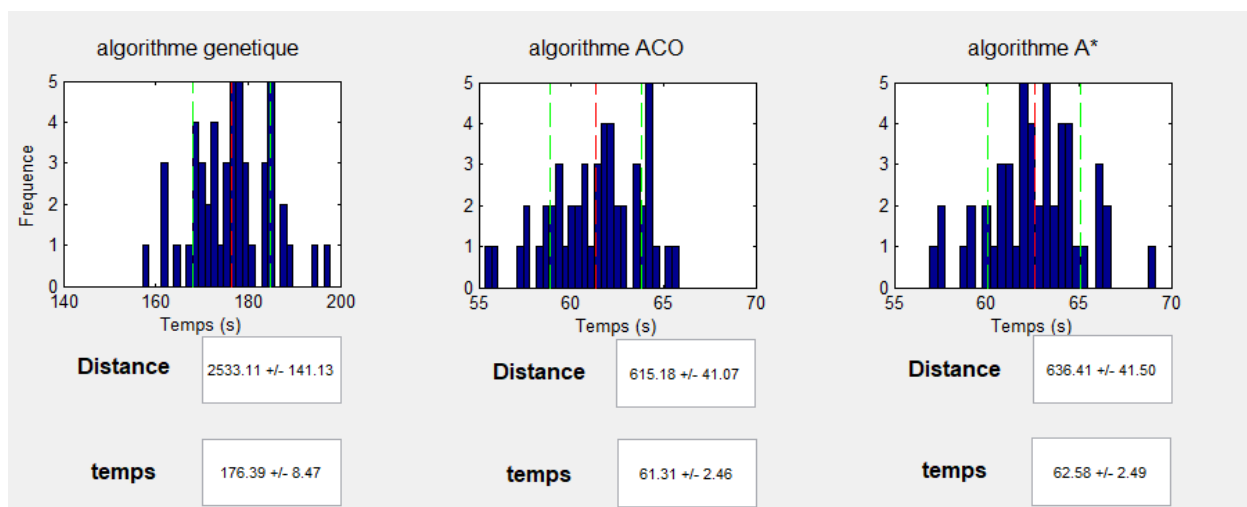


Figure 23 : Simulation Monte Carlo : histogrammes des temps d'usinage pour 50 points

4.4.3.1. Distance totale (mm)

Algorithme	Moyenne (mm)	Écart-type (mm)
Génétique	2533.11	± 141.13
ACO	615.18	± 41.07
A*	636.41	± 41.50

Table 11 : Simulation Monte Carlo : statistiques sur la distance totale (50 points)

Analyse :

- À 50 points, ACO prend un net avantage, surpassant A* en moyenne et en stabilité.
- A* reste stable, mais sa solution n'est plus la plus optimale.
- Le génétique souffre davantage de l'augmentation de la complexité : forte distance et forte dispersion.

4.4.3.2. Temps d'usinage estimé (s)

Algorithme	Moyenne (s)	Écart-type (s)
Génétique	176.39	± 8.47
ACO	61.31	± 2.46
A*	62.58	± 2.49

Table 12 : Simulation Monte Carlo : temps d'usinage estimé (50 points)

Analyse :

- Les valeurs suivent fidèlement les distances.
- ACO devient l'algorithme le plus rapide et le plus constant pour ce niveau de complexité.
- A* garde un comportement stable mais moins performant à grande échelle.

4.4.4. Synthèse comparative des trois tailles (10, 30, 50 points)

Taille	Algorithme	Moy. Distance (mm)	Écart-type (mm)	Moy. Temps (s)	Écart-type (s)
10 pts	Génétique	466.83	± 78.63	52.41	± 4.72
	ACO	264.88	± 51.31	40.29	± 3.08
	A*	250.77	± 42.62	39.45	± 2.56
30 pts	Génétique	1495.87	± 127.31	114.15	± 7.64
	ACO	559.19	± 58.91	57.95	± 3.53
	A*	475.28	± 47.00	52.92	± 2.82
50 pts	Génétique	2533.11	± 141.13	176.39	± 8.47
	ACO	615.18	± 41.07	61.31	± 2.46
	A*	636.41	± 41.50	62.58	± 2.49

Table 13 : Résumé comparatif des performances des algorithmes sur 10, 30 et 50 points (Monte Carlo)

4.4.5. Conclusion globale de la simulation Monte Carlo

La simulation Monte Carlo appliquée à différents niveaux de complexité montre que :

- ACO devient progressivement le meilleur choix en termes de performance globale dès que le nombre de points augmente (30 puis 50).
- A* reste très stable, rapide, mais peut produire des solutions sous-optimales dans des cas plus complexes.
- L'algorithme génétique, malgré sa puissance théorique, souffre de variabilité et d'instabilité, en particulier sans réglage précis.

Ainsi, pour des applications industrielles impliquant un grand nombre de points, ACO représente la meilleure solution pour optimiser le trajet de perçage.

4.5. Conclusion

Dans ce chapitre, nous avons procédé à une évaluation comparative par simulation des trois algorithmes d'optimisation intégrés dans l'interface MATLAB : l'algorithme génétique, l'optimisation par colonie de fourmis (ACO) et l'algorithme A*.

Les tests ont été menés sur plusieurs ensembles de points de perçage (10, 30 et 50), et ont permis d'analyser les performances selon plusieurs critères : distance totale, temps d'usinage estimé, stabilité des résultats, et robustesse face à la variation.

Les principaux enseignements sont les suivants :

- ACO s'est révélé être l'algorithme le plus équilibré et performant, combinant bonne optimisation des distances, stabilité raisonnable, et robustesse, en particulier sur des jeux de données plus complexes.
- A*, bien qu'extrêmement rapide et stable, atteint rapidement ses limites en termes de qualité de solution, notamment lorsque le nombre de points augmente.
- L'algorithme génétique, bien que prometteur en théorie, a montré une forte variabilité et une sensibilité aux paramètres, ce qui le rend moins fiable sans ajustement fin.

Les simulations de type Monte Carlo ont permis de confirmer ces tendances de manière statistique, en mettant en évidence la dispersion des résultats et la capacité des algorithmes à fournir des solutions cohérentes et efficaces.

Ces résultats renforcent l'intérêt d'utiliser des méthodes comme ACO pour l'optimisation de trajets en fabrication CNC, tout en soulignant l'importance de mener des analyses approfondies lors du choix d'un algorithme selon les contraintes du problème.

VIII. Conclusion générale

Ce mémoire a porté sur la conception, l'implémentation et l'évaluation par simulation d'un système d'optimisation du trajet de perçage pour une machine CNC, reposant sur trois algorithmes d'intelligence artificielle : l'algorithme génétique, l'optimisation par colonie de fourmis (ACO) et l'algorithme A*.

La première partie du travail a consisté à développer une interface graphique MATLAB intuitive et fonctionnelle. Cette interface permet de charger des points de perçage à partir de fichiers Gerber, de lancer différents algorithmes d'optimisation, de visualiser les parcours générés, d'estimer le temps d'usinage, et de produire automatiquement le G-code nécessaire au pilotage d'une machine CNC.

La seconde partie a permis de comparer les performances des algorithmes sur des jeux de données variés. Les résultats des simulations ont révélé :

- que l'algorithme ACO est le plus performant et robuste, notamment sur des parcours complexes,
- que l'algorithme A* est rapide et stable mais offre des solutions moins optimisées pour les grands ensembles de points,
- et que l'algorithme génétique, bien que prometteur, montre une sensibilité importante aux paramètres et une variabilité des résultats.

L'analyse statistique via des simulations Monte Carlo a permis de renforcer ces conclusions en apportant une vision quantitative de la qualité et de la cohérence des solutions obtenues.

Ainsi, ce projet démontre que l'intégration d'algorithmes d'optimisation dans un outil interactif peut améliorer significativement l'efficacité, la précision et l'automatisation des opérations de perçage dans les systèmes industriels à commande numérique. Ces travaux ouvrent également des perspectives intéressantes pour des extensions futures, notamment l'intégration d'algorithmes hybrides ou l'optimisation multi-objectif appliquée à d'autres procédés de fabrication.

IX. Annexe A : Algorithmes

A.1 – Algorithme Génétique (Genetic Algorithm) , code MATLAB

```

function best_path = algorithme_genetique(X, Y)
    n = length(X);
    coords = [X(:), Y(:)];
    dist = squareform(pdist(coords));

    population_size = 50;
    generations = 100;
    mutation_rate = 0.05;

    % Initialisation population
    population = zeros(population_size, n);
    for i = 1:population_size
        population(i,:) = randperm(n);
    end

    best_path = population(1,:);
    best_cost = calcul_distance(best_path, dist);

    for g = 1:generations
        % Évaluation
        costs = zeros(population_size, 1);
        for i = 1:population_size
            costs(i) = calcul_distance(population(i,:), dist);
        end

        % Sélection (tournoi)
        [~, idx] = sort(costs);
    end

```

```

[~, idx] = sort(costs);
elite = population(idx(1), :);

% Croisement + Mutation
new_population = elite;
for i = 2:population_size
    p1 = population(idx(randi(10))), :);
    p2 = population(idx(randi(10))), :);
    child = crossover_OX(p1, p2);

    if rand < mutation_rate
        child = mutation_swap(child);
    end
    new_population(i,:) = child;
end
population = new_population;

% Mise à jour du meilleur
current_best = population(1,:);
current_cost = calcul_distance(current_best, dist);
if current_cost < best_cost
    best_path = current_best;
    best_cost = current_cost;
end
end
end

```

Annexe A

```
end
```

```
function cost = calcul_distance(path, dist)
    cost = 0;
    for i = 1:(length(path)-1)
        cost = cost + dist(path(i), path(i+1));
    end
    cost = cost + dist(path(end), path(1)); % retour au départ
end
```

```
function child = crossover_OX(p1, p2)
    n = length(p1);
    idx = sort(randperm(n, 2));
    start = idx(1); stop = idx(2);
    child = zeros(1, n);
    child(start:stop) = p1(start:stop);

    ptr = 1;
    for i = 1:n
        if ~ismember(p2(i), child)
            while child(ptr) ~= 0
                ptr = ptr + 1;
            end
            child(ptr) = p2(i);
        end
    end
end
end
end
```

```
function mutated = mutation_swap(path)
    i = randi(length(path));
    j = randi(length(path));
    mutated = path;
    mutated([i, j]) = mutated([j, i]);
end
```

A.2 – Algorithme ACO (Ant Colony Optimization) , code MATLAB

```

function best_path = algorithme_ACO_reel(X, Y)
    n = length(X);
    coords = [X(:), Y(:)];
    dist_matrix = squareform(pdist(coords));

    n_ants = 30;
    n_iter = 100;
    alpha = 1;
    beta = 5;
    rho = 0.5;
    Q = 100;

    pheromone = ones(n, n);
    visibility = 1 ./ (dist_matrix + eye(n));

    best_distance = Inf;
    best_path = [];

    for iter = 1:n_iter
        paths = zeros(n_ants, n);
        distances = zeros(n_ants, 1);

        for k = 1:n_ants
            visited = false(1, n);
            path = zeros(1, n);
            path(1) = randi(n);
            visited(path(1)) = true;

```

```

path(1) = randi(n);
visited(path(1)) = true;

for step = 2:n
    i = path(step - 1);
    prob = (pheromone(i,:) .^ alpha) .* (visibility(i,:) .^ beta);
    prob(visited) = 0;
    prob = prob / sum(prob);
    next = randsample(1:n, 1, true, prob);
    path(step) = next;
    visited(next) = true;
end

paths(k, :) = path;
distances(k) = calcul_distance(path, dist_matrix);
end

[min_dist, idx] = min(distances);
if min_dist < best_distance
    best_distance = min_dist;
    best_path = paths(idx, :);
end

% Mise à jour des phéromones
pheromone = (1 - rho) * pheromone;
for k = 1:n_ants
    for s = 1:n-1
        for s = 1:n-1
            i = paths(k, s);
            j = paths(k, s+1);
            pheromone(i, j) = pheromone(i, j) + Q / distances(k);
            pheromone(j, i) = pheromone(j, i) + Q / distances(k);
        end
    end
end
end

function d = calcul_distance(path, dist_matrix)
    d = sum(dist_matrix(sub2ind(size(dist_matrix), path, circshift(path, -1))));
end

```

A.3 – Algorithme A* , code MATLAB

```
function path = algorithme_astar(X, Y)
    % Basé sur un plus court chemin heuristique entre les points
    coords = [X(:), Y(:)];
    n = length(X);
    visited = false(1, n);
    path = zeros(1, n);

    current = 1;
    path(1) = current;
    visited(current) = true;

    for i = 2:n
        dist = vecnorm(coords - coords(current, :), 2, 2);
        dist(visited) = Inf;
        [~, next] = min(dist);
        path(i) = next;
        visited(next) = true;
        current = next;
    end
end
```

X. Annexe B : calcul du temps total d'usinage

B. -Code MATLAB pour le calcul du temps total d'usinage (XY + Z)

```
% === Calcul temps de déplacement XY ===
temps_gen_xy = handles.genetic_distance / v_deplacement;
temps_aco_xy = handles.aco_distance / v_deplacement;
temps_astar_xy = handles.astar_distance / v_deplacement;

% === Calcul temps Z (descente + monte + pause) ===
% Pour chaque trou : descente Z + monte Z + 0.5 sec de pause
temps_z_par_trou = (2 * z_depth / v_feedrate) + (1.5 / 60); % en minutes
temps_total_z = nb_trous * temps_z_par_trou;

% === Temps total = XY + Z
temps_gen_total = temps_gen_xy + temps_total_z;
temps_aco_total = temps_aco_xy + temps_total_z;
temps_astar_total = temps_astar_xy + temps_total_z;

% === Affichage en secondes
set(handles.editTempsGen, 'String', sprintf('%.2f', temps_gen_total * 60));
set(handles.editTempsACO, 'String', sprintf('%.2f', temps_aco_total * 60));
set(handles.editTempsAstar, 'String', sprintf('%.2f', temps_astar_total * 60));
```

XI. Annexe C : fichier Gerber contenant les coordonnées de perçage

C.- Exemple d'extrait d'un fichier Gerber contenant les coordonnées de perçage

```
X+011699Y+009122  
X+011699Y+019437  
X+011699Y+029752|  
X+011699Y+040066  
X+011699Y+050381  
X+011699Y+060696  
X+011699Y+071011  
X+025439Y+009122  
X+025439Y+019437  
X+025439Y+029752  
X+025439Y+040066  
X+025439Y+050381  
X+025439Y+060696  
X+025439Y+071011  
X+039179Y+009122  
X+039179Y+019437  
X+039179Y+029752  
X+039179Y+040066  
X+039179Y+050381  
X+039179Y+060696  
X+039179Y+071011  
X+052920Y+009122  
X+052920Y+019437  
X+052920Y+029752  
X+052920Y+040066  
X+052920Y+050381  
X+052920Y+060696  
X+052920Y+071011  
X+066660Y+009122  
X+066660Y+019437  
X+066660Y+029752  
X+066660Y+040066  
X+066660Y+050381  
X+066660Y+060696
```

XII. Bibliographie

- [1] Rashid, M. (2020). Introduction to Printed Circuit Boards. Springer.
- [2] Altintas, Y. (2012). Manufacturing Automation: Metal Cutting Mechanics, Machine Tool Vibrations, and CNC Design. Cambridge University Press.
- [3] Lee, K., & Kim, J. (2016). "Toolpath optimization for CNC drilling using TSP techniques", Journal of Manufacturing Systems.
- [4] Kroese, D. P., Brereton, T., Taimre, T., & Botev, Z. I. (2014). "Why the Monte Carlo method is so important today", Wiley Interdisciplinary Reviews: Computational Statistics.
- [5] Sharma, R. (2019). "Applications and Advancements in PCB Technology", Electronics For You Magazine.
- [6] Harper, C. A. (2006). High Reliability PCB Design for Aerospace and Defense Applications. McGraw-Hill.
- [7] Kim, J. (2017). "Flexible and Biocompatible PCB Technologies in Wearable and Implantable Devices", IEEE Transactions on Biomedical Circuits and Systems
- [8] Chen, W., & Zhang, Q. (2018). "High-Frequency PCB Design for 5G and Automotive Radar", IEEE Microwave Magazine
- [9] Madsen, J. (2018). Printed Circuit Board Design Using CAD Tools. IEEE Press .
- [10] MR.TOLINA TABO . PCB DRILLING AND TRACING CNC MACHINE.
- [11] Putz, B. (2018). PCB Design for Real-World Manufacturing: A Guide to Gerber Files. IPC Press.
- [12] Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (2007). The Traveling Salesman Problem: A Computational Study. Princeton University Press.
- [13] Korte, B., & Vygen, J. (2018). Combinatorial Optimization: Theory and Algorithms. Springer.
- [14] Papadimitriou, C. H., & Steiglitz, K. (1998). Combinatorial Optimization: Algorithms and Complexity. Dover Publications.
- [15] Garey, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman.
- [16] Talbi, E.-G. (2009). Metaheuristics: From Design to Implementation. Wiley.

Bibliographie

- [17] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.
- [18] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- [19] Reeves, C. R., & Rowe, J. E. (2002). *Genetic Algorithms: Principles and Perspectives*. Springer.
- [20] Dorigo, M., & Di Caro, G. (1999). "Ant Colony Optimization: A New Meta-Heuristic", *Proceedings of the Congress on Evolutionary Computation (CEC)*.
- [21] Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.
- [22] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions on Systems Science and Cybernetics*.
- [23] Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
- [24] Kroese, D. P., Taimre, T., & Botev, Z. I. (2011). *Handbook of Monte Carlo Methods*. Wiley.
- [25] Metropolis, N., & Ulam, S. (1949). "The Monte Carlo Method", *Journal of the American Statistical Association*.