REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DE MATHEMATIQUES ET D 'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'études

Présenté pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Informatique Décisionnelle et Optimisation

Par : Demane Dounia et Khezzari Hadil.

Intitulé

Algorithme génétique pour l'équilibrage de chaîne de montage

Soutenu devant le jury composé de :

Dr. DABA Ali	**Université de M'sila**	**Président**
Dr. HEMMAK Allaoua	**Université de M'sila**	**Encadreur**
Mr. YAGOUBI Rached	**Université de M'sila**	**Examinateur**

Année universitaire : 2022/2023

Dédicace

Je souhaite offrir ce modeste travail à ceux qui m'ont conduit à ce niveau de science et qui ont passé des nuits blanches pour moi,

قال تعالى

وَاخْفِضْ لَهُمَا جَنَاحَ الذُّلِّ مِنَ الرَّحْمَةِ وَقُلْ رَبِّ ارْحَمْهُمَا كَمَا رَبَّيَانِي صَغِيرًا ﴿٢٤﴾

سورة الإسراء 24

Mes Chers Parents,

Ma Chère mère, qui m'a porté dans son ventre pendant neuf mois et a passé des nuits blanches pour notre bien.

Mon Cher père, qui a tout sacrifié pour nous.

Mes Frères Islam et Mohamed et Sœurs Ahlam, Fatima, Hadjer, ceux qui apportent tant de joie à la vie. Les petits-enfants de notre petite famille, mes amis, que Dieu les préserve de tout mal et de tout péché.

À tout celui qui m'a fait plaisir avec un mot tout au long de la rédaction de mon mémoire.

À tous ceux qui ont supervisé mon éducation depuis le début de mes études jusqu'à aujourd'hui.

DOUNIA

Dédicace

قال تعالى {وأن ليس للإنسان الا ما سعى}

Grace à Dieu, qui nous a permis de continuer notre chemin. Maintenant je voudrais remercier chacun de ma mère, mon père mes frères et mes amis et je remercie tout particulièrement le professeur responsable de nous ***Dr .Hammak Allaoua****, merci à tous.*

HADIL

Remerciements

Je tiens à exprimer mes sincères remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Je tiens tout d'abord à remercier mon directeur de mémoire docteur ***Hammak Allaoua*** *pour son encadrement, sa disponibilité et ses conseils avisés. Sa rigueur et son expérience m'ont été précieuses tout au long de cette aventure universitaire.*

Je remercie également mes enseignants pour leur savoir-faire et leur passion pour leur discipline, qui m'ont transmis des connaissances fondamentales pour la réussite de ce projet.

Mes remerciements vont également à l'ensemble des membres du jury de soutenance pour avoir accepté de lire.

TABLE DES MATIERES:

Liste des Figures

Liste des Tableaux

Liste des Algorithmes

INTRODUCTION GENERALE

L'algorithme génétique est une méthode puissante et efficace utilisée pour résoudre les problèmes complexes de génétique des populations. Il s'inspire des processus naturels de l'évolution et de la sélection naturelle. Dans le domaine de la génétique, l'algorithme génétique permet d'explorer de vastes espaces de recherche pour trouver des solutions optimales ou proches de l'optimal.

Les problèmes d'assemblage constituent un défi important dans de nombreux secteurs industriels. Ils impliquent la coordination et l'optimisation de différentes étapes d'assemblage pour atteindre des objectifs spécifiques tels que la réduction des coûts et l'amélioration de la productivité. Les algorithmes génétiques se sont révélés être une approche efficace pour résoudre ces problèmes complexes. En utilisant des techniques inspirées de l'évolution naturelle, les algorithmes génétiques peuvent trouver des solutions optimales ou proches de l'optimal en tenant compte des contraintes spécifiques. Ils sont largement utilisés dans des domaines tels que l'automobile, l'électronique et la logistique pour optimiser les lignes d'assemblage, planifier la production et gérer les ressources. Grâce à leur flexibilité et à leur adaptabilité, les algorithmes génétiques offrent des perspectives prometteuses pour améliorer l'efficacité et la performance des processus d'assemblage dans divers secteurs industriels.

Dans le domaine des problèmes de lignes d'assemblage, l'utilisation d'algorithmes génétiques s'est avérée être une approche prometteuse pour l'optimisation. Les algorithmes génétiques, inspirés par la sélection naturelle, peuvent rapidement et efficacement trouver des solutions de qualité, même si elles ne sont pas nécessairement exactes. En appliquant ces techniques à l'optimisation des lignes d'assemblage, nous pouvons équilibrer les charges de travail entre les postes et déterminer la séquence optimale des tâches, réduisant ainsi les temps de cycle et augmentant la productivité.

De plus, l'utilisation d'algorithmes génétiques parallèles peut accélérer encore davantage la recherche de solutions optimales en exploitant le potentiel de calcul distribué. En explorant cette approche, nous espérons apporter des améliorations significatives à la conception et à l'efficacité des lignes d'assemblage, offrant ainsi de nouvelles opportunités pour des processus de fabrication optimisés et rentables.

Le problème des lignes d'assemblage SALB-2 est un défi complexe qui nécessite une répartition équilibrée des tâches sur une ligne de production. Pour résoudre ce problème, l'utilisation d'algorithmes génétiques se révèle être une approche efficace.

Les algorithmes génétiques permettent de générer et d'évaluer différentes combinaisons de séquences de tâches, en utilisant des opérations de reproduction, de mutation et de croisement. Cela permet d'explorer un large espace de solutions potentielles et d'identifier des configurations optimales qui minimisent le temps de cycle et maximisent l'efficacité de la ligne d'assemblage. En utilisant cette approche, les entreprises peuvent améliorer leur productivité, réduire les temps d'attente et optimiser l'utilisation des ressources, ce qui contribue à une meilleure performance globale de leurs opérations d'assemblage

Ce mémoire est composé de six chapitres dont on présente une brève description dans les Paragraphes suivants :

Dans le premier chapitre nous allons définir le problème d'assemblage et leur modelé et ensuite définir le problème classique d'équilibrage et le général de ce mémoire et l'état de l'art. Nous présentons des articles de recherche sur notre sujet, et résumons ces articles.

Dans le deuxième chapitre, nous expliquons ce que sont les algorithmes génétiques, leurs principes, ainsi que leur fonctionnement, et les opérateurs des algorithmes génétiques.

Dans le troisième chapitre, on se propose de conception un algorithme génétique pour la résolution de problème le problème de l'équilibrage des lignes d'assemblage.

Dans le dernier chapitre, implémentation d'algorithme génétique sur le problème SALP-2 et le les résultats obtenus.

Nous terminerons notre travail par une conclusion générale qui expose un résumé de ce qu'a été étudié dans ce mémoire et les résultats obtenus de cette recherche.

CHAPITRE I

Les problèmes d'assemblage

I.1. Introduction

L'équilibrage de la chaîne de montage (ALB) est le processus de répartition des stations de travail le long d'une chaîne de montage pour optimiser la productivité et l'efficacité. L'objectif d'ALB est d'assurer un flux de travail fluide, de minimiser les temps d'inactivité et de s'assurer que chaque travailleur est en mesure d'accomplir ses stations dans un délai donné. ALB implique de décomposer le processus d'assemblage en stations individuelles, puis d'attribuer ces stations à différents postes de travail le long de la chaîne de montage. Les stations sont attribuées en fonction de leurs exigences en matière de temps et des compétences et capacités des travailleurs. L'objectif est de créer une charge de travail équilibrée entre les postes de travail afin que toute la chaîne de montage fonctionne à sa capacité maximale.

I.2. Définition

Les problèmes d'équilibrage de lignes d'assemblage font référence à des situations où les stations dans une ligne de production doivent être réparties entre les travailleurs de manière efficace et équitable afin d'optimiser la productivité et la qualité. L'équilibrage de ligne est essentiel pour réduire les coûts, améliorer les délais de livraison et augmenter la satisfaction des clients. Lorsque les stations dans une ligne de production ne sont pas équilibrées de manière appropriée, cela peut entraîner des problèmes tels que des retards de production, des goulots d'étranglement, des temps d'attente inutiles, des coûts excessifs et des niveaux de qualité inférieurs. L'équilibrage de la ligne est donc une station critique pour les entreprises cherchant à maximiser leur efficacité [1].

I.3. Les modèles de lignes d'assemblage

Il existe plusieurs modèles de lignes d'assemblage qui peuvent être utilisés pour améliorer l'efficacité de la production et réduire les coûts. Voici quelques exemples :

3.1 Modèle de ligne d'assemblage de Ford

Ce modèle est basé sur la standardisation des stations et l'efficacité de la chaîne de montage. Chaque travailleur est chargé d'une station spécifique et répétitive, ce qui permet une production rapide et efficace.

3.2 Modèle de ligne d'assemblage Lean

Ce modèle se concentre sur l'élimination des déchets et la maximisation de la valeur pour le client. Les travailleurs sont encouragés à améliorer continuellement le processus de production en identifiant et en éliminant les inefficacités [2].

3.3 Modèle de ligne d'assemblage à haute variation

Ce modèle est conçu pour les entreprises qui fabriquent des produits à haute variation, tels que les voitures personnalisées. Il permet une flexibilité accrue dans le processus de production et nécessite des travailleurs multi stations qui peuvent effectuer plusieurs stations différentes [2].

3.4 Modèle de ligne d'assemblage hybride

Ce modèle combine plusieurs approches différentes pour maximiser l'efficacité de la production. Il peut inclure des éléments du modèle de Ford, du modèle Lean et d'autres méthodes pour créer un processus de production personnalisé pour une entreprise spécifique [2].

I.4. Le problème de l'équilibrage des lignes d'assemblage

Le problème fondamental d'équilibrage des lignes d'assemblage consiste à agencer les activités des différents postes de travail de manière à égaliser plus ou moins la cadence de production des postes.

La ligne est dite équilibrée lorsque la somme des temps improductif est minimisée. Un équilibrage parfait est caractérisé par un temps opératoire constant à tous les postes.

Il existe dans la littérature plusieurs modèles pour l'équilibrage des lignes d'assemblage. Ghosh et Gagnon (1989) distinguent quatre catégories de problèmes relativement aux paramètres de caractérisations présentées précédemment.

Ces catégories sont les suivantes voir **(**FIGURE II.1**)**:

1. Le modèle mono-produit déterministe (single model déterministe, SMD);
2. Le modèle mono-produit probabiliste (single model stochastique, SMS);
3. Le modèle multi-produits déterministe (mixed/multi model déterministe, MMD);
4. le modèle multi-produits probabiliste (mixed/multi model stochastique, MMS).

Il existe dans la littérature autant de types de problèmes d'équilibrage que de combinaisons de types de temps considérés et de type de lignes d'assemblage [2].

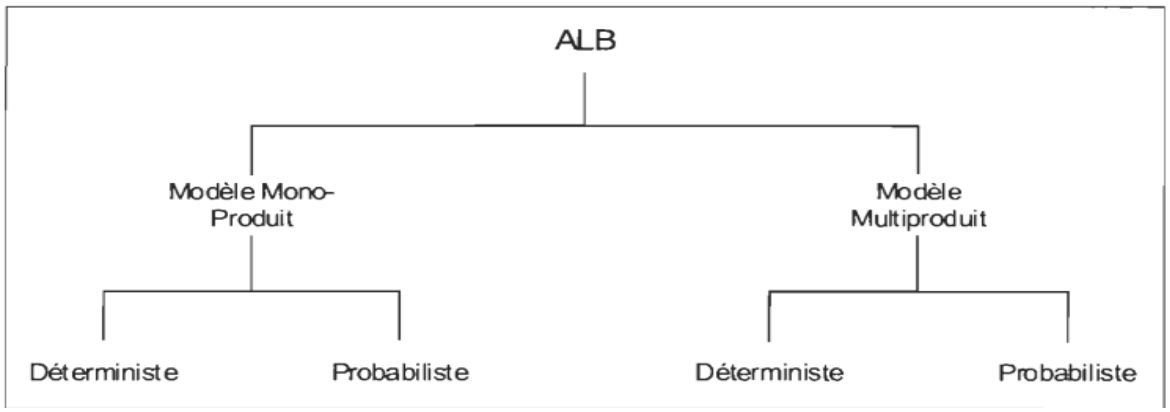

FIGURE II.2 Classification des problèmes d'équilibrage.

I.5. Définition du problème classique d'équilibrage

Deux variantes du problème sont défîmes dans la littérature:

Le SALBP-1 (problème simple d'équilibrage de ligne d'assemblage de deuxième génération) :le type I consiste à affecter les éléments de travail aux postes de façon a minimiser le nombre de postes compte tenu d'un temps de cycle fixe.

Le SALBP-2 : le type II recherche la minimisation du temps de cycle pour un nombre de postes fixe.

Baybars (1986) spécifient les hypothèses sous-jacentes au SALBP :

- Tous les paramètres relatifs à la ligne sont connus avec certitude.
- Une opération est indivisible donne ne peut être partagée entre plusieurs postes.
- Les opérations ne peuvent être exécutées dans un ordre arbitraire. L'ordre des taches est représentés par un graphe acyclique oriente.
- Toutes les opérations doivent être effectuées.
- Tous les postes sont équipes pour effectuer toutes les opérations.
- Le temps de traitement d'une opération est indépendant du poste qui l'exécute.
- Toute opération peut être exécutée dans un poste quelconque.
- La ligne d'assemblage est séquentielle.
- La ligne d'assemblage est conçue pour un produit [3].

I.6. Les procédures de résolution du SALBP-2

Bien qu'il existe de nombreuses méthodes de résolution du SALBP-l peu de techniques ont été conçues pour résoudre le SALBP-2. Cependant, une procédure de résolution du SALBP-I peut être adaptée pour solutionner le SALBP-2. A cette fin, une limite inférieure sur le temps de cycle est calculée. Elle sert de point de départ pour résoudre un problème de

type-l. Si une solution comporte un nombre de postes qui dépasse celui indiqué, la solution est rejetée. Le temps de cycle est alors augmenté d'une unité et le problème est résolu jusqu'à l'obtention d'une solution qui respecte le nombre de postes fixé. (Voir FIGURE I. 2) [3].

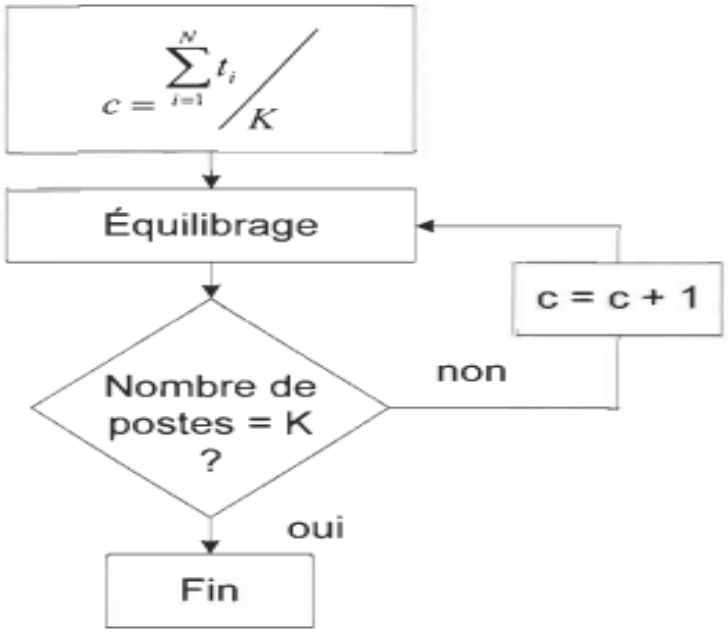

FIGURE I. 2.Procédure d'équilibrage du SALBP -2.

I.7. Les problèmes généraux d'équilibrage

Les problèmes généraux d'équilibrage de lignes d'assemblage (PGELA) sont définis à partir du problème classique par la relaxation d'une ou de plusieurs hypothèses. Les lignes de production multi-modelée par lot ou par unité en sont un exemple. Des modifications de la configuration classique en série des postes constituent également des généralisations du SALBP. A titre d'exemple, nous citerons les lignes avec des postes en parallèles (McMullen et Frazier, 1998 ; Vilarinho et Simaria, 2002 ; (Bukchin et al. 2002) et les lignes en U. Des contraintes additionnelles introduites pour reproduire des situations réelles conduisent tout aussi bien à des GALBP [3].

I.8. État de l'art

I.8.1. Équilibrage de chaînes de montage à l'aide d'algorithmes génétiques 20 décembre 1998:

Dans cet article, un algorithme génétique est développé pour résoudre le problème d'équilibrage de la chaine de montage.il a été montre que la structure chromosomique des algorithmes génétiques peut être modifiée dynamiquement pour fournir une recherche efficace dans l'espace des solution d'équilibrage de la chaine de montage .les résultats des expériences informatique dans leurs recherches ont indiqué que l'approche des algorithmes génétiques proposée est supérieurs aux méthodes d'inférence connues dans la littérature.

I.8.2. Résolution du problème d'équilibrage de la chaîne de montage à l'aide d'un algorithme génétique avec Heuristique - Population Initiale Traitée WCE 2008

Des algorithmes génétique ont été utilisés dans cette recherché afin de comparer un groupe de population initial génère aléatoirement et un group initial traite par induction, les deux groups sont testés à l'aide de l'algorithme génétique.

I.8.3. Heuristiques pour la résolution du problème Équilibrage des lignes d'assemblages April 22 - 24, 2017

Cette recherche a présenté trois méthodes indicatives de type SABP.la première concernait la réduction du temps du décès, la seconde appliquait une solution de classement des opérations selon leur ordre dans le schéma de Précédence, pour la dernière indicative il s'agit d'une méthode propose par Helgeson et Bernie1961 qui est une méthode qui permet de trier selon le poids attribue à chaque élément de l'œuvre.et j'ai présenté des résultats qui diffèrent d'une subvention à l'autre et cela nous montre l'importance du budget qui permet de réduire les couts d'investissement de fonctionnement.

I.8.4. Algorithme génétique pour l'équilibrage de chaîne de montage 28 Décembre 2017

Cette recherché a présenté une application qui a été développée pour résoudre des problèmes d'équilibrage de chaine de montage à l'aide de l'algorithme génétique il a été vérifié que la solution trouvée offrait de l'efficacité .Il a été constant que l'assemblée général a trouvé plus d'une solution possible au problème.

I.8.5. Modèles Déterministe, Stochastique et Multicritère pour l'Équilibrage de Lignes d'Assemblage

Dans ce sujet les contraintes de précédence entre les processus ainsi qu'un nombre fixe de stations sont étudiées .Ce modèle est compatible avec les problèmes de chaine de montage de type II .Ceci a été fait afin de réduire le temps de cycle de la chaine de montage de type II.

I.9. Conclusion

En conclusion, l'équilibrage de la chaîne de montage est un processus important dans la fabrication qui vise à optimiser la productivité et l'efficacité en répartissant les stations de travail le long d'une chaîne de montage. Cependant, plusieurs défis doivent être surmontés pour mettre en œuvre efficacement l'équilibrage de la chaîne de montage. L'un des défis consiste à équilibrer la charge de travail entre les postes de travail tout en tenant compte de divers facteurs tels que les compétences des travailleurs, les contraintes d'équipement et le volume de production. Un autre défi est la nécessité d'ajuster la chaîne de montage lorsque la conception du produit ou les processus de production changent. Ces ajustements peuvent être coûteux et prendre du temps, et peuvent nécessiter une planification et une analyse importantes. Malgré ces défis, l'équilibrage de la chaîne de montage peut offrir plusieurs avantages tels que des taux de production accrus, des coûts réduits, un meilleur contrôle de la qualité.

CHAPITRE II

L'ALGORITHME GÉNÉTIQUE

II.1.Introduction

Dans ce chapitre on explique c'est quoi les algorithmes génétiques, et leur principes, en Plus comment ils fonctionnent outre les opérateurs des algorithmes génétiques.

II.2. Définition

Les algorithmes génétiques (**AG**) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Leur fonctionnement est extrêmement simple. On part avec une population de solutions potentielles (*chromosomes*) initiales arbitrairement choisies. On évalue leur performance (*fitness*) relative. Sur la base de cette performance on crée une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : la sélection, le croisement et la mutation. On recommence ce cycle jusqu'à ce que l'on trouve une solution satisfaisante.
Les AG ont été initialement développées par John Holland (1975) [4].
Les principes de bases étant expliqués, voici le fonctionnement des algorithmes génétiques voir **(**FIGURE II.3**)** : [4].

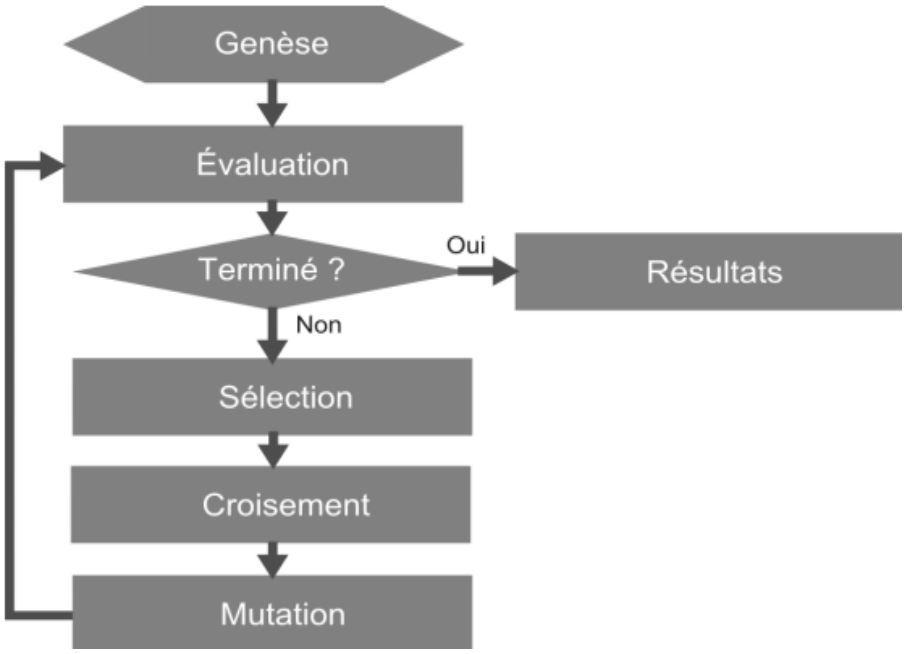

FIGURE II.4 Le fonctionnement général des algorithmes génétiques.

II.3.Principes des algorithmes génétiques

Les algorithmes génétiques utilisent la théorie de Darwin sur l'évolution des espèces. Elle repose sur trois principes : le principe de variation, le principe d'adaptation et le principe d'hérédité. Les algorithmes génétiques utilisent la théorie de Darwin sur l'évolution des espèces. [5]

Nous pouvons utiliser des algorithmes génétiques en raison de la terminologie utilisée en génétique, qui comprend des concepts tels que population, individu, chromosome et gène.

La population est l'ensemble des solutions envisageables.

L'individu représente une solution.

Le Chromosome est un composant de la solution.

Le Gène est une caractéristique, une particularité [5]

3.1 Principes des algorithmes génétiques

Il y a 3 principes pour les algorithmes génétiques pour l'évolution des espèces à partir l'utilisation de la théorie de Darwin sont :

3.1.1 Principe de variation

La population est constituée d'individus distincts. Approximativement ces différences Le processus de sélection dépendra de ce facteur, qui est très important [5]**.**

3.1.2 Principe d'adaptation

Les personnes qui sont plus en phase avec leur environnement sont plus susceptibles de mûrir. Par conséquent, ceux qui sont plus capables de survivre pourront procréer davantage [5].

3.1.3 Principe d'hérédité

Pour être transmis à ses descendants, les traits d'un individu doivent être héréditaires. Grâce à l'utilisation de ce mécanisme, l'espèce pourra développer des traits bénéfiques à sa survie [5].

II.4. Les opérateurs d'algorithmes génétiques

Les opérateurs d'algorithmes génétiques sont des outils qui permettent de modifier la population d'individus d'une génération à l'autre afin de favoriser la recherche de solutions optimales. Les opérateurs les plus couramment utilisés sont :

4. 1 L'opérateur de sélection

La sélection consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche de converger vers l'optimum global. Cet opérateur est l'application du principe d'adaptation de la théorie de Darwin [6].

Diverses méthodes de sélection existent. Les plus utilisés sont listés ci-dessous :

4.1.1 Sélection par roulette

Cette méthode est souvent utilisée en raison de sa simplicité et de son efficacité pour les problèmes de taille modérée [7].

4.1.2 Sélection par tournoi

Cette méthode est utile pour les problèmes où les individus ont des scores de fitness très proches [8].

4.1.3 Sélection par rang

Cette méthode est souvent utilisée pour les problèmes où il y a une grande variabilité dans les scores de fitness des individus [9].

4.2 L'opérateur de croisement

Le croisement, ou enjambement, crossing-over, est le résultat obtenue lorsque deux chromosomes partage leurs particularités.

Celui-ci permet le brassage génétique de la population et l'application du *principe d'hérédité de* la théorie de Darwin [6] ,Il y a deux méthodes pour le croisement.

4.2.1 Simple croisement (un seul point)

Le croisement simple à un seul point est une opération utilisée dans les algorithmes génétiques pour créer de nouvelles solutions en combinant des caractéristiques génétiques de deux parents. Voir **(**FIGURE II.4**)** [10]**.**

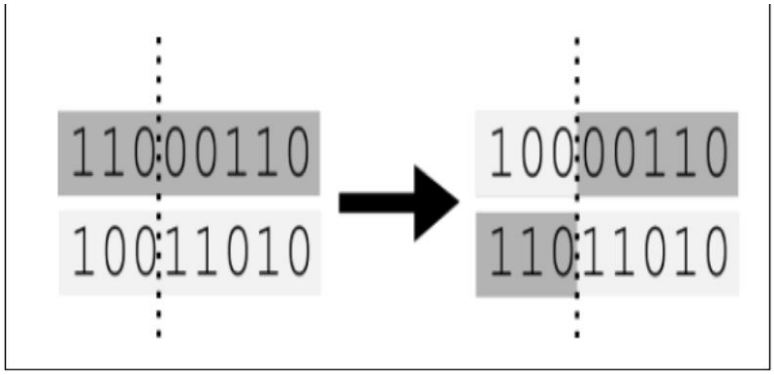

FIGURE II. 5 Croisement sur un seul point.

4.2.2 Double croisement (deux points)

Le croisement à deux points pour combiner les caractéristiques génétiques de deux parents. On sélectionne deux chromosomes parents et on choisit deux points de croisement aléatoires le long de ces chromosomes. Voir **(**FIGURE II.5**)** [10].

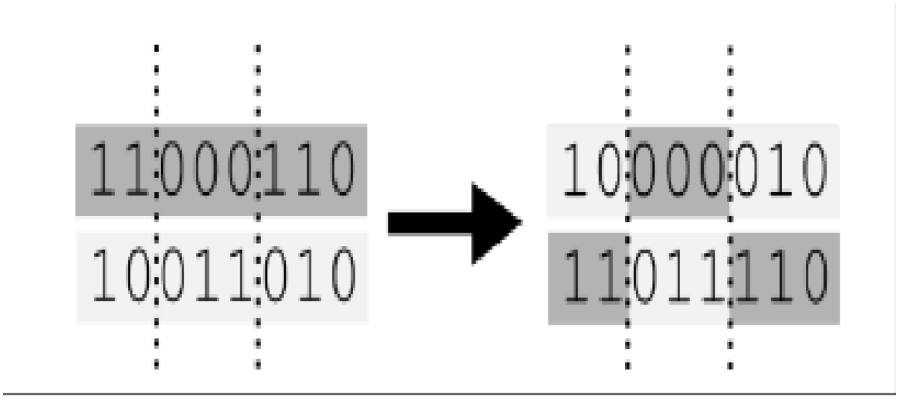

FIGURE II.6 Croisement sur deux points.

4.3 L'opérateur de mutation

La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu. Cet opérateur est l'application du principe de variation de la théorie de Darwin et permet, par la même occasion, d'éviter une convergence prématurée de l'algorithme vers un extremum local [36]. Voir **(**FIGURE II.7) [10].

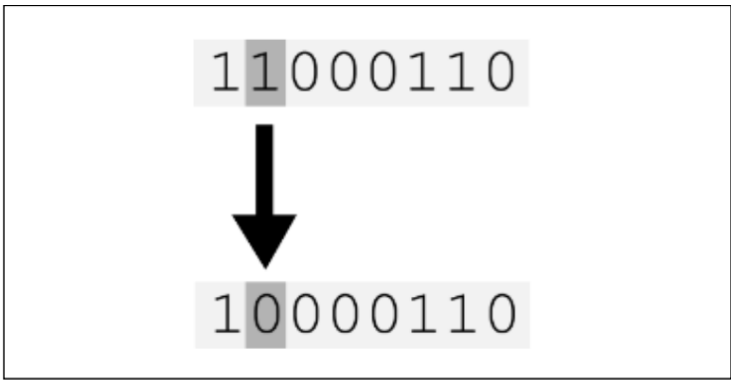

FIGURE II. 8 Représentation d’opérateur de la mutation.

Quelques explications :

- La genèse est l'étape de la création d'une population aléatoire. C'est le point de départ de notre algorithme
- L'évaluation est l'analyse des individus pour analyser si une solution est disponible. Pour ceci, nous utilisons une fonction de coût, ou d'erreur, afin de définir le score d'adaptation des individus lors du processus de sélection.
- Nous effectuons une boucle tant que l'évaluation estime que la solution n'est pas optimale [4].

II.5.Conclusion

En conclusion, les algorithmes génétiques sont des techniques d'optimisation inspirées de la biologie évolutive. Ils sont particulièrement adaptés pour résoudre des problèmes complexes pour lesquels les méthodes exactes sont difficiles ou impossibles à appliquer. Les opérateurs de sélection, de croisement et de mutation permettent de créer une population de candidats de qualité croissante au fil des itérations. Les algorithmes génétiques sont donc une approche efficace pour résoudre une grande variété de problèmes d'optimisation.

CHAPITRE III
CONCEPTION DE L'AG POUR LE PROBLEME

III.1. Introduction

Dans ce chapitre un aperçu de la façon dont vous pouvez utiliser un algorithme génétique pour résoudre le problème d'équilibrage de charge.

III.2.Problématique

Le problème comme un ensemble de stations et un ensemble de ressources, où chaque station a un certain coût de calcul et chaque ressource à une certaine capacité. L'objectif est de répartir les stations sur les ressources de manière à minimiser le déséquilibre de charge.

III.3.La formulation mathématique

La formulation mathématique de SALBP-2 (Problème de Balancement de Ligne D'Assemblage de Deuxième Génération) peut varier en fonction des contraintes et des objectifs spécifiques pris en compte. Cependant, voici une formulation générale qui capture l'essence du problème [11].

Variables :

- x_{ij}- Variable binaire indiquant si la station *i* est assignée à la station de travail *j* (1 si assignée, 0 sinon)

Paramètre :

- *N* - Nombre de stations.
- *M* - Nombre de stations de travail.
- T_i - Temps de traitement de la station *i.*
- d_{ij} - Contrainte de précédence entre les stations *i* et *j.*

Objectif :

Minimiser le temps de cycle *C* en respectant les contraintes suivantes :

$$Min\ C$$

1. Contrainte d'assignation :

$\sum_{j=1}^{M} x_{ij} = 1$ Pour tout *i* (Chaque station est assignée à exactement une station de travail)

2. Contrainte de précédence :

   $x_{ij} \leq x_{kj}$ Pour tout *i, j, k* tels que d_{ij} =1, (Si une contrainte de précédence existe entre les stations *i* et *j*, elles doivent être assignées à la même station de travail ou à des stations de travail dans le même ordre)

3. Contrainte de temps de cycle :

   $C \geq \sum_{i=1}^{N} \sum_{j=1}^{M} t_i . x_{ij}$ (Le temps de cycle doit être supérieur ou égal à la somme des temps de traitement des stations assignées aux stations de travail).

4. Contrainte de capacité de la station de travail :

$\sum_{i=1}^{N} t_i . x_{ij} \leq T_j$ Pour tout *j* (La somme des temps de traitement des stations assignées à chaque station de travail ne doit pas dépasser sa capacité maximaleT_j).

L'objectif est de trouver les valeurs des variables de décisionx_{ij} qui minimisent le temps de cycle C tout en respectant les contraintes. La résolution de cette formulation peut aider à déterminer une affectation optimale des stations aux stations de travail dans le cadre du SALBP-2 [12].

III.4.La complexité

La complexité du problème SALBP-2 dans l'algorithme génétique dépend de plusieurs facteurs, notamment la taille du problème (le nombre de stations et de stations de travail), le nombre de générations, la taille de la population et les opérations génétiques utilisées (sélection, croisement, mutation).

En général, la complexité de l'algorithme génétique pour le problème SALBP-2 peut être approximée comme suit :

La génération initiale de la population nécessite O (P*N) opérations, où P est la taille de la population et N est le nombre de stations.

Les opérations de sélection peuvent avoir une complexité de O (P) en fonction de la méthode de sélection utilisée. Cela peut varier en utilisant des méthodes telles que la sélection proportionnelle à la fitness ou la sélection par tournoi.

Le croisement des individus sélectionnés nécessite généralement O(P) opérations, car chaque paire d'individus peut être croisée pour générer deux descendants.

Les opérations de mutation peuvent être effectuées sur un ou plusieurs individus et nécessitent généralement O (P*N) opérations, où P est la taille de la population et N est le nombre de stations. Cependant, la probabilité de mutation peut également affecter la complexité, car elle détermine le nombre d'individus affectés par la mutation.

L'évaluation de la fitness pour chaque individu peut nécessiter O (N*M) opérations, où N est le nombre de stations et M est le nombre de stations de travail. Cela implique le calcul du temps de cycle pour chaque assignation de stations.

La convergence de l'algorithme génétique dépend du nombre de générations. Par conséquent, la complexité totale peut être exprimée par O(G*(P*N+N*M)) ou bien O(G*N*(P+M)) où G est le nombre de générations.

Il convient de noter que ces estimations sont approximatives et peuvent varier en fonction de la mise en œuvre spécifique de l'algorithme génétique et des opérations utilisées [12].

III.5.Approche utilisée

5.1 Les Algorithmes Génétique

Un algorithme génétique (AG) est une méthode d'optimisation basée sur des concepts inspirés de la théorie de l'évolution biologique. Il est utilisé pour résoudre des problèmes d'optimisation et de recherche, en trouvant des solutions de meilleure qualité au fil du temps. [11].

Exemple :

Supposons que nous ayons 8 tâches (T1, T2, ..., T8) à assigner à deux postes de travail (P1 et P2). Chaque tâche a un temps de traitement (Tp) associé et des contraintes de précédence. Les temps de traitement des tâches sont les suivants :

T1: 4 unités de temps

T2: 6 unités de temps

T3: 3 unités de temps

T4: 7 unités de temps

T5: 5 unités de temps

T6: 2 unités de temps

T7: 3 unités de temps

T8: 4 unités de temps

Les contraintes de précédence sont les suivantes :

T1 doit être exécutée avant T4

T2 doit être exécutée avant T5

T3 doit être exécutée avant T6

T4 doit être exécutée avant T7

T5 doit être exécutée avant T8

L'objectif est de trouver une séquence d'assignation des tâches à P1 et P2 qui minimise le temps total de production tout en respectant les contraintes de précédence.

5.1.1 Initialisation

Générer une population initiale de solutions potentielles appelées individus. Chaque individu représente une solution possible au problème donné et est généralement représenté sous forme d'une chaîne binaire ou d'un vecteur de valeurs.

Exemple :

Pour l'initialisation dans cet exemple de SALBP-2, nous pouvons générer une population initiale d'individus représentant différentes séquences d'assignation des tâches aux postes de travail. Voici un exemple d'initialisation avec une population de 5 individus :

Individu 1: [T1, T2, T3, T4, T5, T6, T7, T8]

Individu 2: [T1, T3, T2, T6, T4, T5, T7, T8]

Individu 3: [T2, T1, T3, T5, T4, T6, T7, T8]

Individu 4: [T3, T1, T2, T5, T4, T6, T7, T8]

Individu 5: [T1, T2, T3, T5, T6, T7, T4, T8]

Dans chaque individu, les tâches sont assignées aux postes de travail dans l'ordre indiqué. Par exemple, l'individu 1 a la séquence [T1, T2, T3, T4, T5, T6, T7, T8], ce qui signifie que la tâche T1 est assignée au poste de travail P1, puis la tâche T2 est assignée à P2, la tâche T3 à P1, et ainsi de suite.

5.1.2 Évaluation de la population

Pour évaluer la fitness dans cet exemple de SALBP-2, nous devons calculer le temps total de production pour chaque individu de la population en tenant compte des contraintes de précédence.

Voici les étapes pour évaluer la fitness :

Initialiser le temps total de production à 0.

Pour chaque individu dans la population :

a. Pour chaque tâche dans la séquence de l'individu :

i. Trouver le poste de travail auquel la tâche est assignée.

ii. Ajouter le temps de traitement de la tâche au temps total de production du poste de travail correspondant.

b. Vérifier si les contraintes de précédence sont respectées :

i. Pour chaque contrainte de précédence (par exemple, T1 doit être exécutée avant T4) :

- Si la tâche précédente n'a pas encore été exécutée, ajouter un coût de pénalité (par exemple, une valeur élevée) au temps total de production.

La fitness de l'individu est égale à l'inverse du temps total de production. Plus le temps total de production est faible, meilleure est la fitness.

Prenons l'exemple de l'individu 1 : [T1, T2, T3, T4, T5, T6, T7, T8]

Supposons que le temps de traitement pour chaque tâche soit le suivant :

T1: 4 unités de temps

T2: 6 unités de temps

T3: 3 unités de temps

T4: 7 unités de temps

T5: 5 unités de temps

T6: 2 unités de temps

T7: 3 unités de temps

T8: 4 unités de temps

Et supposons que les contraintes de précédence soient respectées.

Le temps total de production pour cet individu serait :

Poste de travail P1 : 4 + 3 + 5 + 3 = 15 unités de temps

Poste de travail P2 : 6 + 7 + 2 + 4 = 19 unités de temps

Donc, le temps total de production pour cet individu est de 15 + 19 = 34 unités de temps.

Si nous utilisons une fonction d'évaluation qui prend l'inverse du temps total de production, la fitness pour cet individu serait 1/34.

Vous pouvez répéter ces étapes pour chaque individu de la population afin de calculer la fitness de chacun d'entre eux

5.1.3 Sélection

Sélectionner les individus les plus performants pour la reproduction en fonction de leur aptitude.

Les individus de haute qualité ont une probabilité plus élevée d'être sélectionnés, mais une diversité est maintenue pour éviter la convergence prématurée vers une solution su optimale.

Sélection de la roulette.

Pour chaque solution, calculez la probabilité de sélection en divisant son inverse de temps de cycle par la somme totale de fitness.

Générez un nombre aléatoire entre 0 et 1.Parcourez la population et sélectionnez une solution dont la probabilité de sélection est supérieure au nombre aléatoire généré. Répétez cette étape jusqu'à ce que le nombre souhaité de solutions parent soit sélectionné.

La sélection par roulette, également connue sous le nom de sélection proportionnelle, est une méthode couramment utilisée pour sélectionner les individus dans un algorithme génétique. Dans cet exemple de SALBP-2, voici comment vous pouvez effectuer la sélection par roulette :

Calculer la fitness pour chaque individu de la population en utilisant la méthode d'évaluation de la fitness que nous avons discutée précédemment.

Calculer la somme totale des valeurs de fitness pour tous les individus de la population.

Calculer la probabilité de sélection pour chaque individu en divisant sa valeur de fitness par la somme totale des valeurs de fitness. Cela normalise les valeurs de fitness pour obtenir une distribution de probabilité.

Générer un nombre aléatoire compris entre 0 et 1.

Parcourir les individus de la population et accumuler les probabilités de sélection jusqu'à ce que la somme atteigne ou dépasse le nombre aléatoire généré**.**

L'individu dont la probabilité de sélection a été atteinte ou dépassée est sélectionné.

Répéter les étapes 4 à 6 jusqu'à ce que vous ayez sélectionné le nombre souhaité d'individus pour la reproduction.

Voici un exemple illustrant la sélection par roulette pour une population de 5 individus avec leurs valeurs de fitness respectives :

Individu 1 : Fitness = 1/34

Individu 2 : Fitness = 1/42

Individu 3 : Fitness = 1/39

Individu 4 : Fitness = 1/37

Individu 5 : Fitness = 1/35

Supposons que la somme totale des valeurs de fitness soit de 0,0609.

Maintenant, générez un nombre aléatoire entre 0 et 1, par exemple, 0,486.

À partir de là, effectuez la sélection par roulette en accumulant les probabilités de sélection :

Accumulation des probabilités :

Individu 1 : 0,0323

Individu 2 : 0,0671

Individu 3 : 0,0995

Individu 4 : 0,1323

Individu 5 : 0,1659

Dans cet exemple, l'individu 3 a une probabilité de sélection de 0,0995, qui est la première probabilité à dépasser ou à égaler le nombre aléatoire généré. Par conséquent, l'individu 3 est sélectionné pour la reproduction.

Vous pouvez répéter ces étapes pour sélectionner le nombre souhaité d'individus pour la reproduction. La sélection par roulette favorise les individus ayant une fitness plus élevée, car leur probabilité de sélection est proportionnellement plus grande

.5.1.4 Croisement (Crossover)

Pour effectuer la croisement dans cet exemple de SALBP-2, vous pouvez utiliser un opérateur de croisement basé sur la séquence d'assignation des tâches aux postes de travail. Voici une approche couramment utilisée, appelée "croisement à un point" :

Sélectionnez deux parents de la population, qui serviront de base pour la création de nouveaux individus par croisement.

Choisissez un point de croisement aléatoire entre 1 et le nombre de tâches moins 1. Ce point de croisement déterminera où les séquences des parents seront coupées et combinées pour créer les séquences des enfants.

Prenez la partie de la séquence du premier parent avant le point de croisement, et concaténez-la avec la partie de la séquence du second parent après le point de croisement. Cela forme la séquence du premier enfant.

De même, prenez la partie de la séquence du second parent avant le point de croisement et concaténez-la avec la partie de la séquence du premier parent après le point de croisement. Cela forme la séquence du deuxième enfant.

Appliquez des opérations de mutation, si nécessaire, pour maintenir la diversité de la population. Par exemple, vous pouvez permuter aléatoirement deux tâches dans la séquence d'un enfant.

Répétez les étapes 1 à 5 jusqu'à ce que vous ayez créé le nombre souhaité d'enfants pour la prochaine génération.

Voici un exemple illustrant le croisement à un point avec deux parents :

Parent 1: [T1, T2, T3, T4, T5, T6, T7, T8]

Parent 2: [T1, T3, T2, T6, T4, T5, T7, T8]

Supposons que le point de croisement aléatoire soit à l'indice 4.

En utilisant le point de croisement, nous effectuons le croisement pour créer les enfants :

Enfant 1: [T1, T2, T3, T4, T4, T5, T7, T8]

Enfant 2: [T1, T3, T2, T6, T5, T6, T7, T8]

Ensuite, vous pouvez appliquer des opérations de mutation, si nécessaire, pour maintenir la diversité de la population.

Répétez ces étapes pour créer le nombre souhaité d'enfants pour la prochaine génération. Le croisement permet de combiner les caractéristiques des parents et d'explorer l'espace des solutions afin de trouver de meilleures séquences d'assignation des tâches

.**5.1.5** Mutation

Pour effectuer la mutation dans cet exemple de SALBP-2, vous pouvez utiliser un opérateur de mutation qui perturbe aléatoirement les séquences d'assignation des tâches. Voici une approche couramment utilisée pour la mutation :

Parcourez chaque individu de la population nouvellement créée par croisement.

Pour chaque individu, générez un nombre aléatoire entre 0 et 1. Si ce nombre est inférieur à un seuil de mutation prédéfini, procédez à la mutation de cet individu.

Pour effectuer la mutation, sélectionnez deux positions aléatoires dans la séquence d'assignation des tâches de l'individu.

Permutez les tâches situées aux positions sélectionnées. Cela peut être réalisé en échangeant les positions des tâches.

Répétez les étapes 2 à 4 pour tous les individus de la population qui doivent subir une mutation.

Par exemple, considérons l'individu suivant après le croisement :

Individu : [T1, T2, T3, T4, T4, T5, T7, T8]

Appliquons la mutation à cet individu. Supposons que le seuil de mutation soit fixé à 0,1.

Générez un nombre aléatoire entre 0 et 1, par exemple, 0,073. Comme ce nombre est inférieur au seuil de mutation, nous procédons à la mutation.

Sélectionnez deux positions aléatoires dans la séquence, par exemple, les positions 2 et 6. Les tâches T2 et T5 sont sélectionnées pour la permutation.

Après la permutation, l'individu muté serait :

Individu muté : [T1, T5, T3, T4, T4, T2, T7, T8]

Répétez ces étapes pour tous les individus qui doivent subir une mutation. Cela permet d'introduire une exploration aléatoire dans l'espace des solutions, favorisant la recherche de nouvelles solutions potentiellement meilleures.

Il est important de choisir un seuil de mutation approprié pour contrôler la fréquence des mutations. Un seuil trop élevé peut entraîner une exploration excessive, tandis qu'un seuil trop faible peut limiter l'exploration. Expérimentez et ajustez le seuil de mutation en fonction de votre problème spécifique

.**5.1.6** Remplacement

Remplacer les individus les moins performants de la population précédente par les descendants générés.

Cette étape assure que la population évolue vers de meilleures solutions au fil du temps.

5.1.7 Critères d'arrêt

Définir des critères d'arrêt pour terminer l'algorithme génétique, tels qu'un nombre maximum d'itérations, une convergence satisfaisante ou l'obtention d'une solution optimale.

5.1.8 Répétition

Répéter les étapes de sélection, croisement, mutation et remplacement jusqu'à ce que les critères d'arrêt soient atteints.

III.6. Conclusion

Les algorithmes génétiques sont une classe d'algorithmes d'optimisation inspirés du processus de sélection naturelle et de génétique. Ils sont largement utilisés pour résoudre différents problèmes d'optimisation, y compris le problème de SALBP-2

CHAPITRE IV

IMPLANTATION ET RÉSULTAT

IV.1.Introduction

Nous fissent la présentation de notre travaille pour bien explication nous commençons par la présentation de martiale de développement apres ca nous expliquant les programme qui nous utilisons, enfin nous testons notre programme avec des paramètres déférent.

IV.2.Les éléments matériels

Nous avons utilisé comme élément matériel un ordinateurs HP qui possède comme particuliers:

Processeur**:** Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz.

Mémoire vive installée**:** 8,00 Go.

Type du système**:** Système d'exploitation 64 bits, processeur x64.

IV.3.les éléments logiciels

Plateforme utilisé est Windows 11 professionnel .le langage de développement choisit est python.

3.1 Le langage de programmation "python"

Python est un langage de programmation interprété de haut niveau qui a été conçu dans un souci de simplicité et de lisibilité. Il a été créé par Guido van Rossum et a été publié pour la première fois en 1991 [13].

3.2 L'environnement Jupyter Notebook

Jupyter Notebook est un environnement de développement interactif qui a révolutionné la façon dont les scientifiques des données, les chercheurs et les développeurs travaillent avec du code et des données. Grâce à sa nature basée sur le web voir (FIGURE IV.1), Jupyter Notebook permet d'écrire, d'exécuter et de partager du code de manière interactive, en combinant des cellules de code, des explications textuelles des visualisations et bien plus encore. Cela facilite la collaboration, lexpérimentation et la documentation des analyses de données, de la modélisation et des projets d'apprentissage automatique. Que ce soit pour l'exploration de données, la création de modèles prédictifs ou la communication de résultats, Jupyter Notebook offre un environnement flexible et puissant pour transformer des idées en réalité [14].

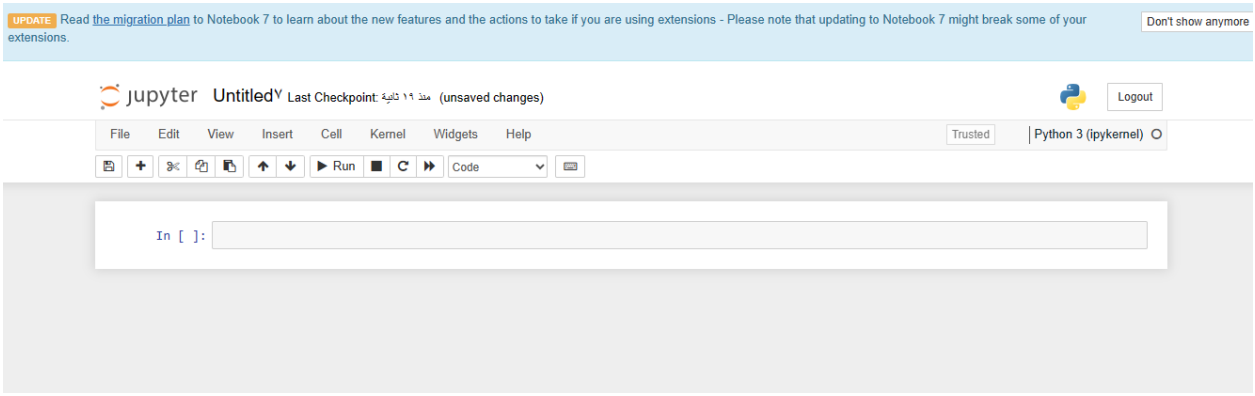

FIGURE IV 1 Interface de Jupyter.

IV.4.importation des modules nécessaires

```
: import copy
import math
from random import randint, random
from numpy.random import permutation
from functools import reduce
import numpy as np
import matplotlib.pyplot as plt
import time
```

Algorithme IV. 1 Bibliothèque nécessaires.

1.1 Importe les modules nécessaires

Copy: permet de créer des copies d'objets.

Math: fournit des fonctions mathématiques.

Randint et random from random: utilisé pour générer des nombres aléatoires.

Permutation from numpy .random: utilisé pour générer des permutations aléatoires.

Reduce from functools: utilisé pour réduire une liste à une seule valeur.

Numpy and matplotlib.pyplot as np and plt: utilisés respectivement pour les opérations numériques et de traçage.

L'algorithme de génération de population initiale comme suite :

```
def gen_indv(cross_type, mut_type, num_stations, num_operations):
    code = [randint(0, num_stations - 1) for i in range(num_operations)]
    return Individual(num_operations, num_stations, code, cross_type=cross_type, mut_type=mut_type)
```

Algorithme IV. 2 L'algorithme de génération de population initiale**.**

Gen_indv :

Cette fonction génère un individu (solution) pour l'algorithme génétique. Il prend le type de croisement, le type de mutation, le nombre de stations et le nombre d'opérations en entrée. Il crée un code (une liste d'entiers aléatoires) représentant les stations pour chaque opération.

```
def create_population(pop_size, cross_type, mut_type, num_stations, num_operations):
    pop = [gen_indv(cross_type, mut_type, num_stations, num_operations) for i in range(pop_size)]
    return pop
```

Algorithme IV. 3 L'algorithme de génération de population initiale**.**

Create _population :

Cette fonction crée une population d'individus pour l'algorithme génétique .Il prend la taille de la population, le type de croisement, le type de mutation, le nombre de stations et le nombre d'opérations en entrée .Il utilise la fonction gen_indv pour générer des individus et crée une liste d'individus .Il renvoie la liste de la population.

Initialisation

```
def engine(k, num_operations, graph, times, num_stations=10,
           pop_size=100, iterations=100,
           perc_elitism=0.1, perc_mat=0.1, sel_type='roulette', cross_type='SP',
           mutation_rate=0.05, mut_type='random'
           ):


    population = rank_population(k, graph, times,
                                 create_population(pop_size, cross_type, mut_type, num_stations, num_operations),
                                 0)

    best = []
    mean = []

    select = eval('select_by_' + sel_type)

    for i in range(iterations):

        best.append(population[0].fitness)
        mean.append(reduce(lambda x, y: x + y.fitness, population, 0)/pop_size)
        if population[0].gen < i - 50:
            break
```

Algorithme IV. 4 Initialiser la population**.**

La fonction commence par initialiser la population à l'aide de la fonction create_population avec la taille de population donnée, le type de croisement (cross_type), le type de mutation (mut_type) et d'autres paramètres.

```
# Elitism
new_generation = population[:int(perc_elitism*pop_size)]
```

Algorithme IV. 5 Élitisme**.**

Une partie des meilleurs individus est conservée dans la nouvelle génération.

La sélection

```
# Selection
the_chosen_ones = select(population[:int(pop_size * perc_mat)], num=(pop_size - int(perc_elitism*pop_size)))

mut = 2
```

Algorithme IV. 6 La sélection.

Les individus restants pour l'accouplement sont sélectionnés à l'aide de la méthode de sélection spécifiée et on a 3 méthodes comme suite

```
def select_by_roulette(population, num=2):
    total = reduce(lambda x, y: x + y.fitness, population, 0)
    roulette = [indiv.fitness / total for indiv in population]

    return np.random.choice(a=population, size=num, p=roulette)
```

Algorithme IV.6 .1 La sélection par roulette.

```
def select_by_tournament_(population):
    i = randint(0, len(population) - 1)
    j = randint(0, len(population) - 1)
    return population[i] if (population[i].fitness > population[j].fitness)else population[j]
def select_by_tournament(population, num=2):
    return [select_by_tournament_(population) for i in range(num)]
```

Algorithme IV.6 . 2 La sélection par tournoi.

```
def select_by_rank(population, num=2):
    total = len(population)*(1 + len(population))/2
    rank = [i/total for i in range(1, len(population) + 1)]
    return np.random.choice(a=population, size=num, p=rank)
```

Algorithme IV.6. 3 La sélection par rang.

Le croisement

Des paires d'individus sélectionnés sont croisées pour créer de nouveaux individus descendants.

L’algorithme de génération de croisement comme suite :

```
# Crossover
for j in range(0, len(the_chosen_ones), 2):

    if j == len(the_chosen_ones) - 1:
        new_generation.append(Individual(num_operations, num_stations, copy.deepcopy(the_chosen_ones[j].code),
                                         cross_type=cross_type, mut_type=mut_type))
        mut = 1
    else:
        new_generation.extend(the_chosen_ones[j].crossover(the_chosen_ones[j+1]))
```

Algorithme IV.7 Croisement**.**

Il y a trois méthodes de croisement différentes : crossover_SP, crossover_DP et crossover_UX Chaque méthode effectue un type différent de croisement entre deux individus.

1 Croisement à un point (crossover_SP)

```
def crossover_SP(self, indv):

    p = randint(0, self.operations)

    code_c1 = self.code[:p] + indv.code[p:]
    code_c2 = indv.code[:p] + self.code[p:]

    ch1 = Individual(self.operations, self.stations, code=copy.deepcopy(code_c1),
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])

    ch2 = Individual(self.operations, self.stations, code=copy.deepcopy(code_c2),
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])
    return ch1, ch2
```

Algorithme IV.7.1 Croisement à un point**.**

2 Croisement à deux points (crossover_DP)

```
def crossover_DP(self, indv):

    i = randint(0, self.operations)
    j = randint(0, self.operations)
    if i > j:
        j, i = i, j

    code_c1 = copy.deepcopy(self.code[:i] + indv.code[i:j] + self.code[j:])
    code_c2 = copy.deepcopy(indv.code[:i] + self.code[i:j] + indv.code[j:])

    ch1 = Individual(self.operations, self.stations, code=code_c1,
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])

    ch2 = Individual(self.operations, self.stations, code=code_c2,
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])
    return ch1, ch2
```

Algorithme IV.7. 2 Croisement à un point**.**

3 Croisement aléatoire (crossover_UX)

```
def crossover_UX(self, indv):

    code_c1 = [0] * self.operations
    code_c2 = [0] * self.operations

    for i in range(self.operations):
        if random() < 0.5:
            code_c1[i] = self.code[i]
            code_c2[i] = indv.code[i]
        else:
            code_c2[i] = self.code[i]
            code_c1[i] = indv.code[i]

    ch1 = Individual(self.operations, self.stations, code=copy.deepcopy(code_c1),
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])

    ch2 = Individual(self.operations, self.stations, code=copy.deepcopy(code_c2),
                     cross_type=self.crossover.__name__[-2:],
                     mut_type=self.mutate.__name__.split('_')[-1])

    return ch1, ch2
```

Algorithme IV.7.3 Croisement aléatoire**.**

Mutation

Certains des nouveaux individus de la progéniture sont mutés en fonction du taux et du type de mutation spécifiés**.** Ces fonctions permettent d'effectuer différentes opérations de mutation sur le code des individus, ce qui peut introduire de la diversité dans la population et potentiellement améliorer les solutions.

```
# Mutation
for indv in new_generation[-mut:]:
    if random() < mutation_rate:
        if mut_type == 'heur':
            indv.mutate(graph)
        else:
            indv.mutate()
```

Algorithme IV .8 Mutation**.**

Voici les différentes fonctions de mutation.

1 Mutations aléatoires (mutate_random)

```
def mutate_random(self):
    self.code[randint(0, self.operations - 1)] = randint(0, self.stations - 1)
    self.fitness = 0
```

Algorithme IV .8 1 Mutation aléatoires**.**

.

2 Mutation heuristique (mutate_heur)

```
def mutate_heur(self, graph):

    has_changed = False
    for op in range(self.operations):
        for neighbor in graph[op]:
            if self.code[neighbor] < self.code[op]:
                self.code[op] = randint(0, self.stations - 1)
                has_changed = True

    if not has_changed:
        self.mutate_random()
    else:
        self.fitness = 0
```

Algorithme IV .8 . 2 Mutation heuristique**.**

3 Mutation par échange (mutate_swap)

```
def mutate_swap(self):

    i = randint(0, self.operations - 1)
    j = randint(0, self.operations - 1)

    self.code[i], self.code[j] = self.code[j], self.code[i]
    self.fitness = 0
```

Algorithme IV .8 . 3 Mutation par échange.

4 Mutation par mélange (mutate_scramble)

```
def mutate_scramble(self):

    i = randint(0, self.operations)
    j = randint(0, self.operations)

    if i > j:
        i, j = j, i

    self.code[i:j] = copy.deepcopy(permutation(self.code[i:j]))
    self.fitness = 0
```

Algorithme IV .8 . 4 Mutation par mélange.

5 Mutation par inversion (mutate_inversion)

```
def mutate_inversion(self):

    i = randint(0, self.operations)
    j = randint(0, self.operations)

    if i > j:
        i, j = j, i

    self.code[i:j].reverse()
    self.fitness = 0
```

Algorithme IV .8 . 5 Mutation par inversion**.**

Évaluation

La fonction d'évaluation joue un rôle crucial pour guider l'algorithme génétique vers la recherche de meilleures solutions au fil du temps, en orientant la recherche vers la solution optimale ou quasi-optimale du problème à résoudre.

```
    # Evaluation
    population = rank_population(k, graph, times, new_generation, i)

if (population[0].calc_violations(graph)) > 0:
    print("SOLUTION NOT VALID: ", population[0].calc_violations(graph))

print(f"Best solution : {[i+1 for i in population[0].code]}")
print(f"Best solution reached after {population[0].gen} generations.")
print(f"Fitness of the best solution : {population[0].fitness}")

return population[0], best, mean
```

Algorithme IV.8 Évaluation.

IV.4. Résultats et discussion

Notre ensemble de données est composé du nombre d'opérations et du temps de chaque opération et pour chaque opération il Ya contrainte de présidence

Exemple 1

Le tableau présente l'opération et le temps de chaque opération et la présidence de chaque opération pour appliquer 75 opérations voir (TABLE IV.1).

Tableau IV. 1 Les donnes d'opérations.

Opération	Temps de chaque opération	précédence
1	23	2, 3,4, 5, 6, 7
2	24	15
3	25	13,24
4	26	8, 14,16
5	23	12,15
6	22	9, 10, 11, 13
7	6	-
….75	…	….

```
Best solution : [1, 1, 2, 1, 1, 1, 12, 14, 1, 1, 9, 5, 5, 5, 1, 1, 10, 1, 15, 1, 4, 10, 9, 2, 2, 1, 1, 11, 11, 13, 2, 5, 8, 2,
1, 1, 5, 12, 12, 4, 15, 1, 1, 14, 11, 5, 1, 15, 1, 2, 12, 2, 2, 14, 6, 9, 8, 7, 1, 12, 14, 2, 11, 12, 9, 1, 15, 2, 12, 5, 9, 9,
13, 12, 12]
Best solution reached after 58 generations.
Fitness of the best solution : 407
time of processing is  12.921608209609985 s
Best solution : [1, 1, 2, 2, 1, 1, 3, 10, 1, 1, 10, 3, 2, 4, 1, 2, 4, 1, 4, 1, 2, 3, 5, 2, 2, 2, 1, 4, 4, 4, 4, 6, 4, 8, 1, 2,
7, 8, 8, 4, 10, 1, 2, 6, 6, 8, 1, 9, 1, 2, 10, 4, 4, 10, 4, 7, 7, 6, 2, 2, 6, 8, 10, 10, 8, 3, 10, 3, 7, 5, 8, 10, 8, 10, 9]
Best solution reached after 62 generations.
Fitness of the best solution : 278
Best solution : [1, 1, 2, 1, 1, 1, 7, 4, 1, 1, 5, 2, 6, 10, 1, 1, 8, 3, 3, 1, 3, 7, 3, 2, 2, 3, 1, 8, 9, 9, 4, 3, 4, 9, 1, 1,
7, 7, 6, 2, 10, 1, 1, 10, 10, 2, 1, 7, 1, 1, 10, 2, 2, 10, 5, 8, 7, 7, 1, 3, 2, 6, 8, 10, 10, 3, 6, 4, 5, 6, 7, 9, 5, 10, 6]
Best solution reached after 158 generations.
Fitness of the best solution : 402
SOLUTION NOT VALID:  1
Best solution : [1, 2, 2, 1, 1, 1, 6, 3, 1, 1, 6, 2, 4, 8, 2, 1, 3, 1, 2, 10, 8, 7, 6, 2, 3, 1, 1, 3, 7, 8, 1, 1, 8, 8, 1, 2,
9, 3, 6, 3, 9, 1, 2, 10, 4, 6, 2, 8, 2, 4, 9, 4, 2, 7, 8, 4, 9, 3, 2, 7, 9, 5, 10, 5, 5, 4, 8, 4, 8, 6, 5, 7, 9, 10, 8]
Best solution reached after 61 generations.
Fitness of the best solution : 581
```

FIGURE IV. 2 Le résultat obtenu pour 75 opérations.

Les résultats indiquent les meilleures solutions obtenues (voir FIGURE IV. 3) après un certain nombre de générations dans un processus d'optimisation. Chaque solution est représentée par une séquence de nombres.

La première meilleure solution est [1, 1, 2, 1, 1, 1, 12, 14, 1, 1, 9, 5, 5, 5, 1, 1, 18, 1, 15, 1, 4, 18, 9, 2, 2, 1, 1, 11, 11, 13, 2, 5, 8, 2, 1, 1, 5, 12, 12, 4, 15, 1, 1, 14, 11, 5, 1, 15, 1, 2, 12, 2, 2, 14, 6, 9, 8, 7, 1, 12, 14, 2, 11, 12, 9, 1, 15, 2, 12, 5, 9, 9, 13, 12, 12]. Cette solution a été obtenue après 58 générations et sa valeur de fitness est de 407. La fitness est une mesure de la qualité de la solution, où une valeur plus élevée indique une meilleure solution.

La deuxième meilleure solution est [1, 1, 2, 2, 1, 1, 3, 10, 1, 1, 10, 3, 2, 4, 1, 2, 4, 1, 4, 1, 2, 3, 5, 2, 2, 2, 1, 4, 4, 4, 4, 6, 4, 8, 1, 2, 7, 8, 8, 4, 10, 1, 2, 6, 6, 8, 1, 9, 1, 2, 10, 4, 4, 18, 4, 7, 7, 6, 2, 2, 6, 8, 10, 10, 8, 1, 10, 1, 7, 5, 8, 18, 8, 18, 9]. Elle a été atteinte après 62 générations et sa valeur de fitness est de 278.

La troisième meilleure solution est [1, 1, 2, 1, 1, 1, 7, 4, 1, 1, 5, 2, 6, 10, 1, 1, 8, 3, 3, 1, 3, 7, 3, 2, 2, 3, 1, 8, 9, 9, 4, 3, 4, 9, 1, 1, 7, 7, 6, 2, 18, 1, 1, 10, 18, 2, 1, 7, 1, 1, 10, 2, 2, 10, 5, 8, 7, 7, 1, 3, 2, 6, 8, 18, 18, 3, 6, 4, 5, 6, 7, 9, 5, 18, 6]. Elle a été obtenue après 158 générations et sa valeur de fitness est de 402.

La quatrième solution n'est pas valide, mais aucune information n'est donnée sur ce qui rend cette solution invalide.

Chaque génération de l'algorithme d'optimisation génère de nouvelles solutions en utilisant des mécanismes de sélection, de recombinaison et de mutation pour améliorer progressivement les performances des solutions. Le processus se poursuit jusqu'à ce qu'une solution satisfaisante soit trouvée ou jusqu'à ce qu'un certain critère d'arrêt soit atteint.

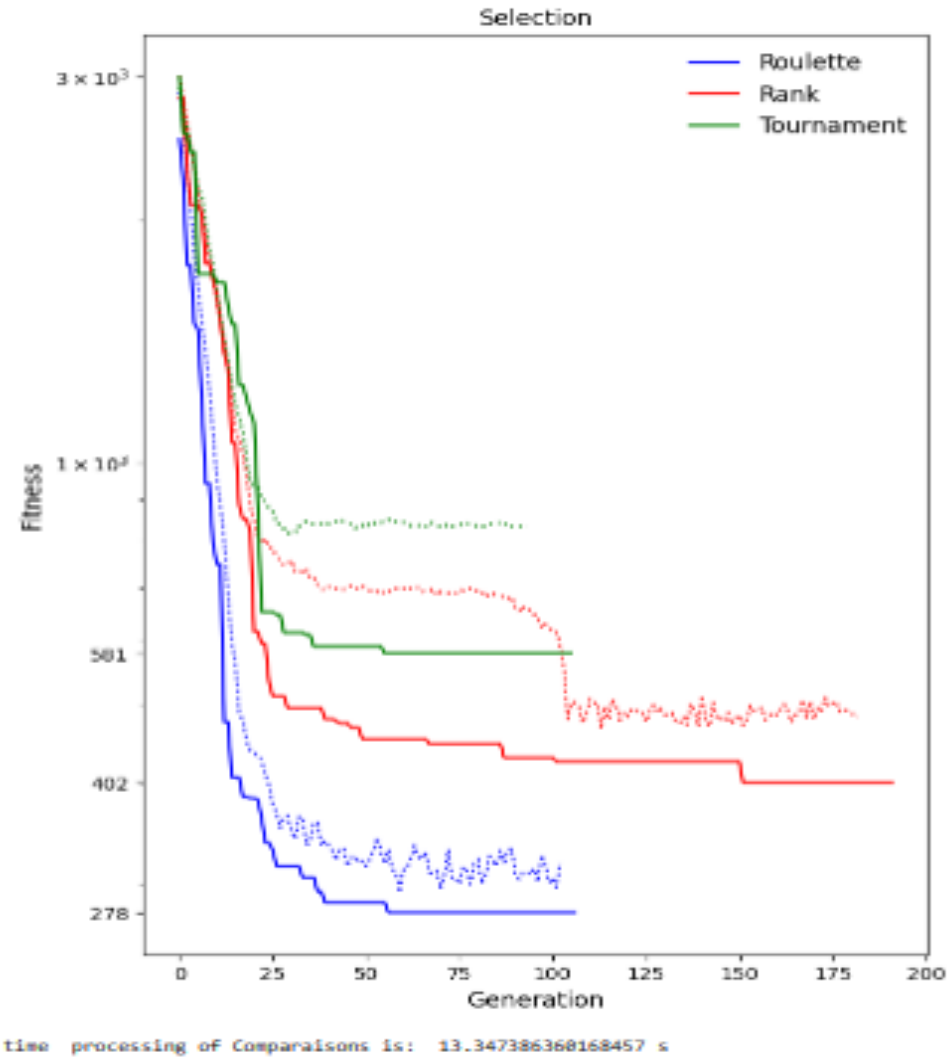

FIGURE IV.4 Comparer les méthodes de sélection.

Ce courbes représente comparaison pour les 3 méthodes de sélections voir (FIGURE IV.4). Indicative fitness et génération, on a remarqué que le plus de génération la valeur de fitness elle est devient petite .le mois de génération le plus de valeur de fitness.

Nous concluons que la méthode de sélection le plus efficace dans cette cas est Roulette pacque elle reset crée de génération.

Exemple 2

Pour notre exemple on a appliqué 100 opérations.

```
SOLUTION NOT VALID:  2
Best solution : [1, 2, 2, 2, 3, 2, 20, 12, 2, 4, 5, 9, 7, 7, 3, 4, 7, 4, 17, 20, 5, 15, 14, 3, 5, 4, 1, 10, 11, 15, 5, 5, 12, 1
1, 1, 1, 9, 6, 14, 2, 17, 2, 3, 11, 19, 9, 2, 16, 3, 3, 19, 6, 5, 15, 15, 8, 13, 10, 4, 8, 17, 3, 17, 12, 11, 4, 5, 4, 9, 17, 1
1, 17, 19, 15, 5, 19, 1, 2, 1, 1, 1, 1, 5, 4, 2, 3, 2, 8, 1, 1, 1, 1, 8, 11, 11, 11, 4, 8, 2, 7]
Best solution reached after 61 generations.
Fitness of the best solution : 2270
time of processing is  21.622650861740112 s
SOLUTION NOT VALID:  2
Best solution : [1, 1, 3, 1, 3, 2, 10, 4, 2, 2, 2, 4, 4, 6, 4, 1, 8, 2, 4, 5, 3, 7, 8, 3, 3, 2, 2, 9, 8, 8, 2, 3, 3, 9, 4, 2,
6, 5, 2, 2, 3, 5, 3, 10, 10, 3, 1, 6, 1, 3, 4, 1, 1, 6, 4, 6, 3, 6, 1, 6, 8, 3, 5, 6, 7, 1, 9, 1, 10, 6, 2, 2, 9, 10, 1, 1, 1,
1, 1, 1, 1, 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 3, 4, 6, 7, 7, 2, 2, 1, 4]
Best solution reached after 44 generations.
Fitness of the best solution : 2507
SOLUTION NOT VALID:  1
Best solution : [1, 1, 1, 1, 1, 1, 8, 8, 1, 1, 1, 2, 1, 3, 1, 1, 3, 1, 2, 1, 1, 3, 6, 1, 1, 2, 1, 9, 9, 3, 2, 2, 5, 9, 1, 1, 7,
2, 2, 2, 9, 1, 1, 6, 7, 2, 1, 4, 1, 2, 3, 2, 3, 4, 5, 7, 6, 6, 1, 2, 2, 5, 9, 5, 10, 1, 8, 2, 4, 2, 6, 9, 2, 10, 2, 5, 1, 2, 1,
1, 1, 1, 4, 3, 4, 1, 1, 2, 1, 1, 1, 1, 4, 4, 4, 4, 2, 2, 2, 8]
Best solution reached after 78 generations.
Fitness of the best solution : 2320
Best solution : [1, 1, 2, 1, 1, 1, 2, 10, 1, 1, 8, 8, 2, 4, 1, 2, 6, 1, 9, 1, 2, 8, 9, 2, 5, 2, 1, 8, 9, 10, 2, 4, 8, 8, 1, 1,
4, 10, 5, 1, 10, 1, 2, 9, 10, 2, 1, 9, 1, 2, 5, 4, 2, 7, 2, 5, 4, 6, 1, 10, 6, 6, 10, 8, 8, 1, 6, 1, 3, 6, 7, 5, 8, 6, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 4, 1, 1, 1, 1, 2, 6, 6, 6, 2, 6, 2, 5]
Best solution reached after 74 generations.
Fitness of the best solution : 1051
```

FIGURE IV.5 Le résultat obtenu pour 100 opérations.

Avec notre implémentation on a trouvé Le résultat obtenu voir (FIGURE IV.5).

Les résultats présentés consistent en plusieurs solutions générées par un processus d'optimisation. Chaque solution est représentée par une séquence de nombres.

La première solution présentée a été obtenue après 61 générations. Elle est représentée par la séquence suivante : [1, 2, 2, 2, 3, 2, 20, 12, 2, 4, 5, 9, 7, 7, 3, 4, 7, 4, 17, 28, 5, 15, 14, 3, 5, 4, 1, 10, 11, 15, 5, 5, 12, 1, 1, 1, 1, 9, 6, 14, 2, 17, 2, 3, 11, 19, 9, 2, 16, 3, 3, 19, 6, 5, 15, 15, 8, 13, 18, 4, 8, 17, 3, 17, 12, 11, 4, 5, 4, 9, 17, 1, 1, 17, 19, 15, 5, 19, 1, 2, 1, 1, 1, 1, 5, 4, 2, 3, 2, 8, 1, 1, 1, 1, 8, 11, 11, 11, 4, 8, 2, 7]. La valeur de fitness de cette solution est de 2278.

La deuxième solution a été obtenue après 44 générations. Elle est représentée par la séquence : [1, 1, 3, 1, 3, 2, 18, 4, 2, 2, 2, 4, 4, 6, 4, 1, 8, 2, 4, 5, 3, 7, 8, 3, 3, 2, 2, 9, 8, 8, 2, 3, 3, 9, 4, 2, 6, 5, 2, 2, 3, 5, 3, 18, 18, 3, 1, 6, 1, 3, 4, 1, 1, 6, 4, 6, 3, 6, 1, 6, 8, 3, 5, 6, 7, 1, 9, 1, 10, 6, 2, 2, 9, 10, 1, 1, 1, 1, 1, 1, 1, 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 3, 4, 6, 7, 7, 2, 2, 1, 4]. Sa valeur de fitness est de 2587.

La troisième solution a été obtenue après 78 générations. Elle est représentée par la séquence : [1, 1, 1, 1, 1, 1, 8, 8, 1, 1, 1, 2, 1, 3, 1, 1, 3, 1, 2, 1, 1, 3, 6, 1, 1, 2, 1, 9, 9, 3, 2, 2, 5, 9, 1, 1, 7, 2, 2, 2, 9, 1, 1, 6, 7, 2, 1, 4, 1, 2, 3, 2, 3, 4, 5, 7, 6, 6, 1, 2, 2, 5, 9, 5, 10, 1, 8, 2, 4, 2, 6, 9, 2, 18, 2, 5, 1, 2, 1, 1, 1, 1, 4, 3, 4, 1, 1, 2, 1, 1, 1, 1, 4, 4, 4, 4, 2, 2, 2, 8]. Sa valeur de fitness est de 2328.

La quatrième solution a été obtenue après 74 générations. Elle est représentée par la séquence : [1, 1, 2, 1, 1, 1, 2, 18, 1, 1, 8, 8, 2, 4, 1, 2, 6, 1, 9, 1, 2, 8, 9, 2, 5, 2, 1, 8, 9, 10, 2, 4, 8, 8, 1, 1, 4, 18, 5, 1, 18, 1, 2, 9, 18, 2, 1, 9, 1, 2, 5, 4, 2, 7, 2, 5, 4, 6, 1, 10, 6, 6, 10, 8, 8, 1, 6, 1, 3, 7, 5, 8, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 4, 1, 1, 1, 1, 2, 6, 6, 6, 2, 6, 2, 5]. Sa valeur de fitness est de 1851.

Cependant, certaines des solutions générées sont marquées comme "INVALIDES". Cela signifie qu'elles ne répondent pas aux critères spécifiques du problème ou qu'elles ont été évaluées comme étant moins optimales par rapport aux autres solutions valides.

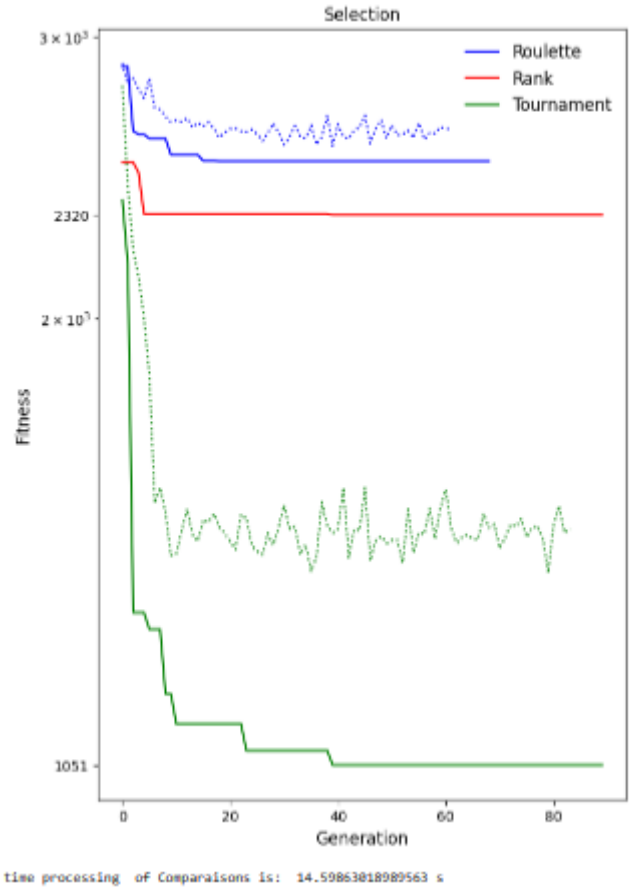

FIGURE IV. 6 Comparer les méthodes de sélection 2.

Ce courbes représente comparaison pour les 3 méthodes de sélections voir (FIGURE IV. 6). Indicative fitness et génération, on a remarqué que le plus de génération la valeur de fitness de chaque méthode elle est devient petite la méthode de roulette est Ranke devient stable après 5 génération et la valeur de fitness est stable mais la méthode de roulet la valeur de fitness est décroissante. Le mois de 5 génération le plus des valeurs de fitness pour les 3 méthodes.

Nous concluons que la méthode de sélection le plus efficace dans ce cas est tourmente .

Comparaison

Tableau IV. 2 Temps de trématent.

Temps de traitement	Temps de traitement de la comparaison	Donnes
12,91 s	13,34 s	Pour 75 opérations
21,66 s	14,59 s	Pour 100 opérations

IV.5.Conclusion

Dans ce chapitre, en premier temps nous avons présenté les environnements matériel et logiciel et les Données utilisés, ensuite de test les résultats obtenues sur des exemples, et en fin nous avons présenté et étudier l'effet de l'ensemble des paramètres de l'algorithme génétique sur les résultats retournés par cet algorithme.

CONCLUSION GÈNÈRALE

En conclusion, notre étude a démontré l'efficacité des algorithmes génétiques dans la résolution du problème SALP-2. Nous avons pu obtenir des résultats prometteurs en termes d'optimisation des objectifs multiples, tels que la minimisation du temps de cycle, tout en tenant compte des contraintes de préférence des travailleurs.

L'utilisation de l'algorithme génétique nous a permis d'explorer efficacement l'espace de recherche complexe du SALP-2 et de trouver des solutions de haute qualité. Les opérations de sélection, de croisement et de mutation ont contribué à la diversification et à l'amélioration progressive des solutions au fil des générations.

Cependant, il convient de noter que l'algorithme génétique n'est pas exempt de limitations. La performance de l'algorithme dépend fortement des paramètres choisis, tels que la taille de la population, les taux de croisement et de mutation, ainsi que des choix de conception spécifiques pour représenter les individus et les gènes.

Dans ce mémoire, nous avons rencontré plusieurs difficultés lors de notre étude sur le problème SALP-2 et l'utilisation de l'algorithme génétique. Voici quelques-unes des difficultés rencontrées :

- Collecte des informations pour cette étude
- Complexité du problème.
- Recherche de paramètres optimaux.
- Modélisation des contraintes de préférence.
- Temps de calcul.
- Évaluation des résultats.

Malgré ces difficultés, nous avons pu surmonter ces obstacles grâce à une approche méthodique, des itérations et des ajustements constants. Les défis rencontrés ont également contribué à l'enrichissement de notre compréhension du problème et à l'identification des possibilités d'amélioration future.

Des recherches futures peuvent se concentrer sur l'amélioration des techniques d'algorithme génétique pour le SALP-2, en explorant différentes stratégies de sélection, en

introduisant des opérateurs de recombinaison plus avancés, ou en combinant l'algorithme génétique avec d'autres approches d'optimisation.

En définitive, notre étude montre que les algorithmes génétiques offrent une approche prometteuse pour résoudre le problème SALP-2 et ouvrent la voie à de nouvelles opportunités de recherche dans le domaine de l'optimisation des chaînes de montage.

Résume

Dans ce travail, nous avons résolu le problème complexe des chaînes de montage, plus précisément le problème SALP-2, en utilisant des algorithmes génétiques. Ce problème est connu pour être NP complet, ce qui signifie qu'il est difficile de trouver une solution optimale en un temps raisonnable. Les algorithmes génétiques que nous avons appliqués nous ont permis d'explorer efficacement l'espace des solutions et de trouver des résultats prometteurs. Bien que les solutions obtenues ne soient pas garanties d'être optimales, elles étaient proches de l'optimum et satisfaisantes dans un délai raisonnable. Cette approche représente une avancée significative dans la résolution du problème des chaînes de montage et ouvre des perspectives intéressantes pour des améliorations futures.

Mots clés : algorithmes génétiques, problème de chaîne de montage, salp-2.

ملخص

في هذا العمل، قمنا بحل المشكلة المعقدة لخطوط التجميع، وتحديدًا مشكلة SALP-2، باستخدام الخوارزميات الجينية. تُعرف هذه المشكلة بأنهاNP-complete ، مما يعني أن العثور على حل مثالي في فترة زمنية معقولة يمثل تحديًا. سمحت لنا الخوارزميات الجينية التي طبقناها باستكشاف مساحة الحل بكفاءة والحصول على نتائج واعدة. على الرغم من أن الحلول التي تم الحصول عليها ليست مضمونة لتكون مثالية، إلا أنها كانت قريبة من المستوى الأمثل والمرضي ضمن إطار زمني معقول. يمثل هذا النهج تقدمًا كبيرًا في حل مشكلة خط التجميع ويفتح آفاقًا مثيرة للاهتمام للتحسينات المستقبلية.

.

الكلمات المفتاحية: الخوارزميات الجينية، مشكلة خط التجميع، salp-2.

Abstract

In this work, we solved the complex problem of assembly lines, specifically the SALP-2 problem, using genetic algorithms. This problem is known to be NP-complete, which means that finding an optimal solution in a reasonable amount of time is challenging. The genetic algorithms we applied allowed us to efficiently explore the solution space and obtain promising results. Although the solutions obtained are not guaranteed to be optimal, they were close to the optimum and satisfactory within a reasonable time frame. This approach represents a significant advancement in solving the assembly line problem and opens up interesting prospects for future improvements.

Keywords: genetic algorithms, assembly line problem, salp-2.

Bibliographies

[1] T. F. S. Scholz-Reiter"" ،Line balancing and production leveling.2019 "،"

[2] J. D. C. J. R. Dixon" " ،Assembly Line Balancing "،"Third Edition, ed. by G. Salvendy, Wiley..2018 ،

[3] TH.VICTOR" ،DEVELOPPEMENT D'UN OUTIL INFORMATIQUE POUR L'EQUILIBRAGE DE LIGNE DANS LE CONTEXTE DE PRODUCTION MIXTE "،p. page 10.2008 ،.

[4] M. e. T. Vallée " ،Présentation des algorithmes génétiques et de leurs applications en économie 2004 04 "،feb.

[5] D. Goldberg " ،Algorithmes génétiques dans la recherche تأليف "،*l'optimisation et l'apprentissage automatique*.1989 ،

[6] "Wikipédia .[متصل] .2022 03 25 "،Available : http://igm.univmlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html. ,

[7] D. E. Goldberg " ،Simple genetic algorithms and the minimal deceptive problem تأليف "،.*In Proceedings of the 1st International Conference on Genetic Algorithms* ،pp. (pp. 74-88)..

[8] S. W. Mahfoud" ،Crowding and niching in genetic algorithms. PhD thesis "،,University of Illinois at Urbana-Champaign.1995 ،.

[9] J. E. Baker" ،Adaptive selection methods for genetic algorithms ،1985 "،.pp. (pp. 101-111).

[10] M.Yildizoglu " ،Présentation des algorithmes génétiques et de leurs applications en économie "، universiter de brodeux.2004 ،

[11] D. E. Goldberg " ،Genetic Algorithms in Search تأليف "،*Optimization, and Machine Learning* ، . Addison-Wesley Professional.(1989 ،.

[12] I. Baybars " ،.A survey of exact algorithms for the simple assembly line balancing problem "، Management Science ،,aout 1986.[متصل] .

[13] G. van Rossum" ،Python Reference Manual. Consulté à l'adresse .[متصل] .2023 06 01 "،. Available: htttp/www.python.com.

[14] T. R.-K. B. P. F. G. B. B. M. F. J. K. K. H. G. J. C. S. .. I. P. A. D. A. S. W. C. Kluyver" ،Jupyter Notebooks "،Jupyter Notebooks - un format de publication pour des workflows in .2016 ، .[متصل]