REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre :SIOD21/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Système d'Information Optimisation et Décision (SIOD)**

Algorithme de colonie d'abeilles artificielles multi-objectif pour l'optimisation d'un système manufacturier reconfigurable

---

Par :

BEN TALEB Habib Errahmane

Soutenu le ../07/2021 devant le jury composé de :

	grade	Président
TORKI Fatima Zohra	grade	Rapporteur
	grade	Examinateur

Année universitaire 2020-2021

REMERCIEMENTS

Allah soit loué, qui nous a aidés à faire de cette saison scolaire un succès et à rédiger une lettre de remise des diplômes, surtout après les circonstances difficiles qui se sont produites dans le monde il y a un an, et en Algérie en particulier, qui est l'épidémie du virus Corona.

A l'occasion de la fin de l'année scolaire en cours, nous remercions sincèrement toutes les personnes qui nous ont soutenus dans la réalisation de ce travail, et nous nous spécialisons en cela grâce à l'Encadreur Mm TORKI Fatima Zohra

Les jurys pour leurs efforts et leur soin apporté à notre travail. Aux Enseignants de notre université et département informatique.

Enfin, nous adresse nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours soutenue et encouragée au cours de la réalisation de ce mémoire.

DEDICACES

Je dédie ce travail

A mes chers parents, qui ont toujours étaient là pour m'encourager. Quoi que je fasse ou que je dise, je ne saurai vous remercier comme il se do1it.

A mon cher frère Yakoub et mes chères sœurs :

Asma,Meriem et mon bon frère Seif sans leur soutien indéfectible et sans limite ce travail n'aura jamais pu voir le jour.

Je dédie aussi ce travail à Mes chères amis,Zinou,Oussama,Sofian,Haithem,Haider,Issam, Amine,Wassim, Aymen et mes collègues et tous ceux qui m'estiment.

Bentaleb Habib Errahmane

Résumé

Le Systèmes de Production Reconfigurable (RMS) est un nouveau paradigme dans les systèmes de production. Il est défini comme un système de production ayant les capacités de faciliter les changements adéquats, permettant d'ajuster ses structures et ses processus en réponse aux différents besoins, il est composé d'une collection de machines ou de stations en collaboration pour effectuer un ensemble d'opérations sur une matière première en vue d'obtenir la forme finale souhaitée du produit demandé et spécifié par le client.

Le travail élaboré dans ce mémoire vise à générer des lignes de production optimales afin d'obtenir les meilleurs arrangements des machines reconfigurables de type RMTs minimisant le coût et le temps. Tout cela pour obtenir les meilleures solutions en utilisant l'algorithme de colonies des abeilles artificielles multi-objectif (MOABC)

Abstract

Reconfigurable Production Systems (RMS) is a new paradigm in production systems. It is defined as a production system with the capacities to facilitate adequate changes, allowing to adjust its structures and processes in response to different needs, it is composed of a collection of machines or stations in collaboration to perform a set operations on a raw material in order to obtain the desired final shape of the product requested and specified by the customer.

The work elaborated in this thesis aims to generate optimal production lines in order to obtain the best arrangements of reconfigurable machines of type RMTs minimizing cost and time. All this to get the best solutions using the Multi-Objective Artificial Bee Colony Algorithm (MOABC)

ملخص

تعد أنظمة الإنتاج القابلة لإعادة التكوين (RMS) نموذجًا جديدًا في أنظمة الإنتاج. يتم تعريفه على أنه نظام إنتاج لديه القدرات على تسهيل التغييرات المناسبة ، مما يسمح بتعديل هياكله وعملياته استجابة للاحتياجات المختلفة ، ويتألف من مجموعة من الآلات أو المحطات بالتعاون لأداء مجموعة عمليات على مادة خام في من أجل الحصول على الشكل النهائي المطلوب للمنتج الذي يطلبه ويحدده العميل.

يهدف العمل المفصل في هذه الأطروحة إلى إنشاء خطوط إنتاج مثالية من أجل الحصول على أفضل الترتيبات للآلات القابلة لإعادة التشكيل من نوع RMTs لتقليل التكلفة والوقت. كل هذا للحصول على أفضل الحلول باستخدام خوارزمية مستعمرة النحل الاصطناعية متعددة الأغراض(MOABC)

.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

Dans l'industrie et avant années 70 ,la demande du marché pour les produits manufacturés était très élevée et tout ce qui était produit trouvait immédiatement un acheteur car les exigences du client étaient simples. On le sait, la production consiste à transformer des matières premières en produits finis, grâce à des systèmes de fabrication avancés. Un système manufacturier contient de l'ensemble des éléments matériels et immatériels nécessaires à la production de produits ayant la forme finale désirée du produit spécifié par le client

Plus tôt, il existait deux types principaux de systèmes manufacturiers : les Systèmes de Production Dédiés (Dedicated Manufacturing Systems– DMSs) et les Systèmes de Production Flexibles (Flexible Manufacturing Systems – FMSs) et nous en parlerons en détail dans le chapitre 1. Par la suite, un nouveau type apparut nommé les Systèmes Manufacturiers Reconfigurables (Reconfigurable Manufacturing Systems - RMS), découvert par Koren [6] où sa fonction est de combiner le haut débit des DMSs avec la grande flexibilité des FMSs tout en garantissant un haut niveau de réactivité face aux changements.

Le système de fabrication reconfigurable (RMS) a été introduit pour répondre à ce nouvel environnement de fabrication orienté vers le marché. Le RMS est conçu pour répondre aux exigences d'une famille de produits en modifiant rapidement les changements de ses composants matériels et logiciels en réponse aux fluctuations rapides des demandes du marché.

Donc le RMS c'est le nouveau paradigme qui a été introduit afin de permettre aux entreprises de rester compétitives dans un environnement caractérisé par des changements fréquents et imprévisibles.

La ligne de production dépend du système manufacturier sur lequel il est généré. RMS est caractérisé par la capacité de ses machines à changer de configuration.

Si les besoins des clients sont nombreux, donc un problème peut avoir plusieurs objectifs et chaque objectif est modélisé à l'aide d'une fonction mathématique. Il essaie de rechercher simultanément une solution optimale ou un ensemble de solutions approchées pour chaque fonction objective. Ce type de problèmes s'appelle des problèmes d'optimisation multi-objectifs. Les problèmes d'optimisation constituent des challenges importants à relever au sein des communautés des ingénieurs, des scientifiques et des décideurs. Cette importance revient à la complexité des systèmes et projets qui grandis jour après jour et qui surgis dans des secteurs très divers.

Notre projet est basé sur les travaux [1], [2], [22]. Donc, notre objectif sert à générer des lignes de production dans un système manufacturier reconfigurable pour trouver les meilleurs arrangements des machines reconfigurable de type RMTs minimisant le coût et le temps Pour obtenir les solutions optimales on va utiliser et adapter l'algorithme nommé Multi Objectif Artificial Bees Colony (MOABC)

Notre mémoire consiste à présenter les chapitres suivants :

Le chapitre 1sera une description de différents types de systèmes manufacturiers, le système manufacturier reconfigurable, de ses caractéristiques, de ses composants.

Le chapitre 2 est réservé à la définition du problème d'optimisation multi-objectifs et les différentes méthodes d'optimisation correspondantes

Le chapitre 3 décrit la conception de notre application représentée en deux niveaux (la conception globale et détaillée).

Le chapitre 4 est destiné à l'évaluation du modèle proposé en conception. Nous commençons par la description des outils utilisés, ainsi que les expérimentations réalisées et les résultats obtenus.

Dans la dernière de notre mémoire une conclusion générale et des perspectives attendues.

chapitre 1 :

Optimisation Multi-objectif

1.1 Introduction

Chaque jour, les ingénieurs et les décideurs sont confrontés à des problèmes de complexité croissante, qui émergent dans divers secteurs techniques. Le problème à résoudre peut souvent être exprimé comme un problème d'optimisation. Résoudre un problème d'optimisation, pour un ensemble de données ou variables, revient à trouver la meilleure solution selon un critère d'évaluation, donné par une fonction de coût (dite fonction Objectif), que l'on souhaite minimiser ou maximiser. Chacune des variables est définie sur un domaine de recherche. Dans ce chapitre en parlant sur l'optimisation multi-objectif et les méthodes méta heuristiques.

1.2 Définition de l'Optimisation

L'optimisation est une partie de la mathématique qui s'intéresse à déterminer la solution globale parmi un ensemble de solutions candidates qui forment l'espace de recherche d'un problème envisagé, cet optimum minimise ou maximise une fonction qui s'appelle fonction objective ou bien fonction de coût. Une optimisation peut se trouver avec et sans contraintes [8].

1.3 L'optimisation multi-objectifs

L'optimisation multi-objectif peut être définie comme la recherche de la meilleure ou des meilleures solutions possibles d'un problème donné. Souvent, les problèmes d'optimisation sont des problèmes multi-objectifs. Si les objectifs sont antagonistes, alors on n'a pas une seule solution optimale mais un ensemble de solutions de compromis [9].

L'optimisation multi -objectif est un axe de recherche très important à cause de la nature multi-objectif de la plupart des problèmes réels. Les premiers travaux menés sur les problèmes multi objectifs furent réalisés au 19éme siècle sur des études en économie par Edgeworth et généralisés par Pareto [10].

1.4 Problème d'optimisation multi-objectif

Dans de nombreux problèmes d'optimisation du monde réel, il n'y a pas un seul objectif mais un ensemble de critères par rapport aux quels une solution peut être mesurée. De tels problèmes sont souvent connus sous le nom de problèmes d'optimisation multi-objectifs, et sont définis par un ensemble de fonctions objectives f_1, f_2,..., f_N sur l'espace de recherche S, dont chacune devrait idéalement être minimisée. L'approche la plus courante de l'optimisation multi-objectif est peut-être de former une nouvelle fonction objectif F qui est une somme pondérée des objectifs individuels, et de chercher à minimiser cette somme [11].

Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante Selon [12] :

Minimiser $f(\vec{x})$ (fonction à optimiser)

Avec $\vec{g}(\vec{x}) \leq 0$ (m contraintes d'inégalité)
Et $\vec{h}(\vec{x}) = 0$ (p contraintes d'égalité
On a $\vec{x} \in R^n$, $\vec{g}(\vec{x}) \in R^m$ et, $\vec{g}(\vec{x}) \in R^p$

Cet ensemble de contraintes délimite un espace restreint de recherche de la solution optimale.

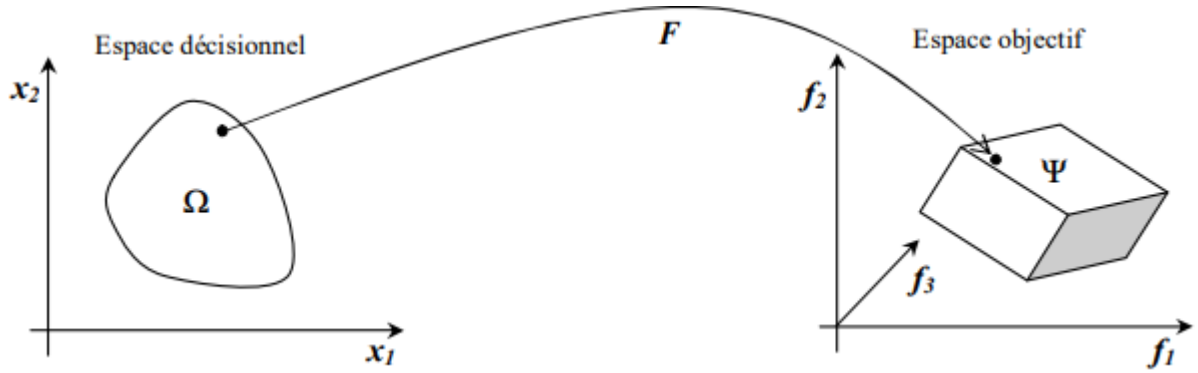


Figure 1.1 : problème d'optimisation multi-objectif (2 variables de décision et 3 fonctions objectifs)[10]

1.4.1 Principaux concepts d'optimisation

- **Fonction objective**

C'est le nom donné à la fonction f (on l'appelle encore **fonction de coût** ou **critère d'optimisation**). C'est cette fonction que l'algorithme d'optimisation va devoir "optimiser" (trouver un optimum).

- **Variables de décision**

Elles sont regroupées dans le vecteur $\vec{x}$. C'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction *f*.

- **Minimum global**

Un "point" $\vec{x}^*$ est un minimum global de la fonction *f* si on a :
$f(\vec{x}^*) < f(\vec{x})$ quel que soit $\vec{x}$ tel que $\vec{x}^* \neq \vec{x}$. Cette définition correspond au point M_3

- **Minimum local fort** :

Un "point" $\vec{x}^*$ est un minimum local fort de la fonction *f* si on a : $f(\vec{x}) < f(\vec{x})$ quel que soit $\vec{x} \in V(\vec{x}*)$ et $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un "voisinage" de $\vec{x}^*$. Cette définition correspond aux points M_2 et M_4 de la figure 1.2.

- **Minimum local faible**

Un "point" $\vec{x}^*$ est un minimum local faible de la fonction *f* si on a :

$f(\rightarrow x*) \leq f(\rightarrow x)$ quel que soit $\vec{x} \in V(\vec{x}^*)$ et $\vec{x}^* \neq \vec{x}$, où $V(\vec{x}^*)$ définit un "voisinage" de $\vec{x}^*$. Cette définition correspond au point $M1$ de la figure 1.2 [12].

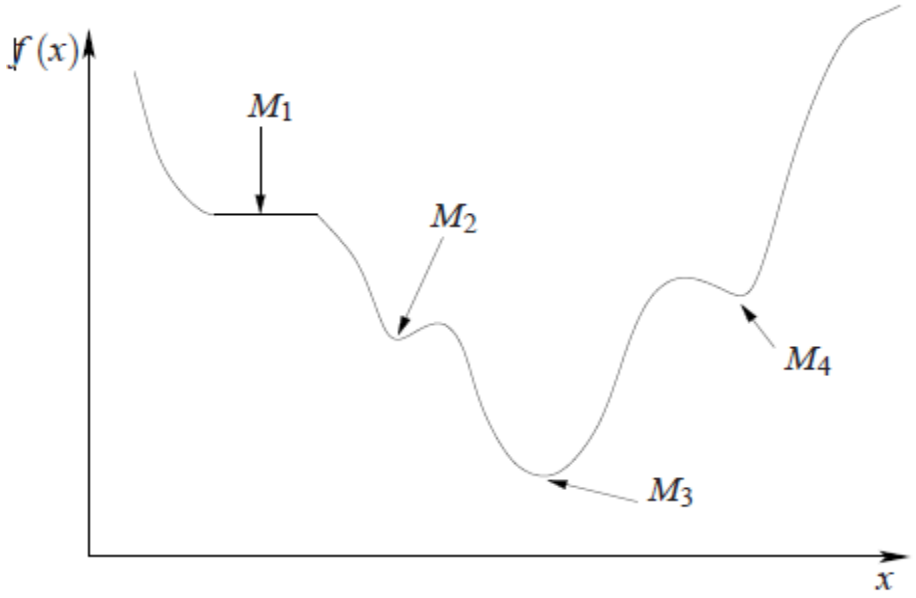


Figure1.2 : :Les différents minima [12]

1.4.2 Notions préliminaires

1.4.2.1 Relation de dominance de Pareto

La résolution d'un problème d'optimisation multi-objectif (nous donne souvent une multitude de solutions appelées solutions Pareto. L'ensemble de solutions obtenues à la fin de la recherche définit la surface de compromis. C'est après avoir trouvé ces solutions que d'autres difficultés surviennent : il faut sélectionner une solution dans cet ensemble. La solution choisie doit refléter les compromis opérés par le décideur. Pour surmonter ces difficultés, il est nécessaire de faire intervenir l'humain à travers un décideur, pour le choix final de la solution à garder [13], il faut qu'il existe une relation de dominance entre la solution considérée et les autres solutions, dans le sens suivant :

On dit que le vecteur multi-objectif $\vec{x}_1$domine le vecteur $\vec{x}_2$ si :

$\vec{x}_1$ est au moins aussi bon que $\vec{x}_1$ dans tous les objectifs et $\vec{x}_1$ est strictement meilleur que $\vec{x}_2$ dans au moins un objectif [12].

La dominance au sens de Pareto permet de constituer un ensemble de solutions dites efficaces satisfaisant l'ensemble des contraintes du problème et offrant le meilleur compromis d'optimisation de l'ensemble de fonction objectif [32].

1.4.2.2 L'optimalité locale au sens de Pareto

Un vecteur $\vec{x} \in R^n$ est optimal localement au sens de Pareto s'il existe un réel d > 0 tel qu'il n'y ait pas de vecteur $\vec{x}'$ qui domine le vecteur $\vec{x}$ avec
$\vec{x}' \in R^n \cap B(\vec{x}, d)$, où $B(\vec{x}, d)$ représente une boule de centre $\vec{x}$ et de rayon [12].

Un vecteur $\vec{x}$ est donc optimal localement au sens de Pareto s'il est optimal au sens de Pareto sur une restriction de l'ensemble R^n.

1.4.2.3 L'optimalité globale au sens de Pareto

Un vecteur $\vec{x}$est optimal globalement au sens de Pareto (ou optimal au sens de Pareto) s'il n'existe pas de vecteur $\vec{x}'$ tel que $\vec{x}'$ domine le vecteur $\vec{x}$.
La différence entre cette définition et celle de l'optimalité locale tient dans le fait que l'on ne considère plus une restriction de l'ensemble R^n [12].

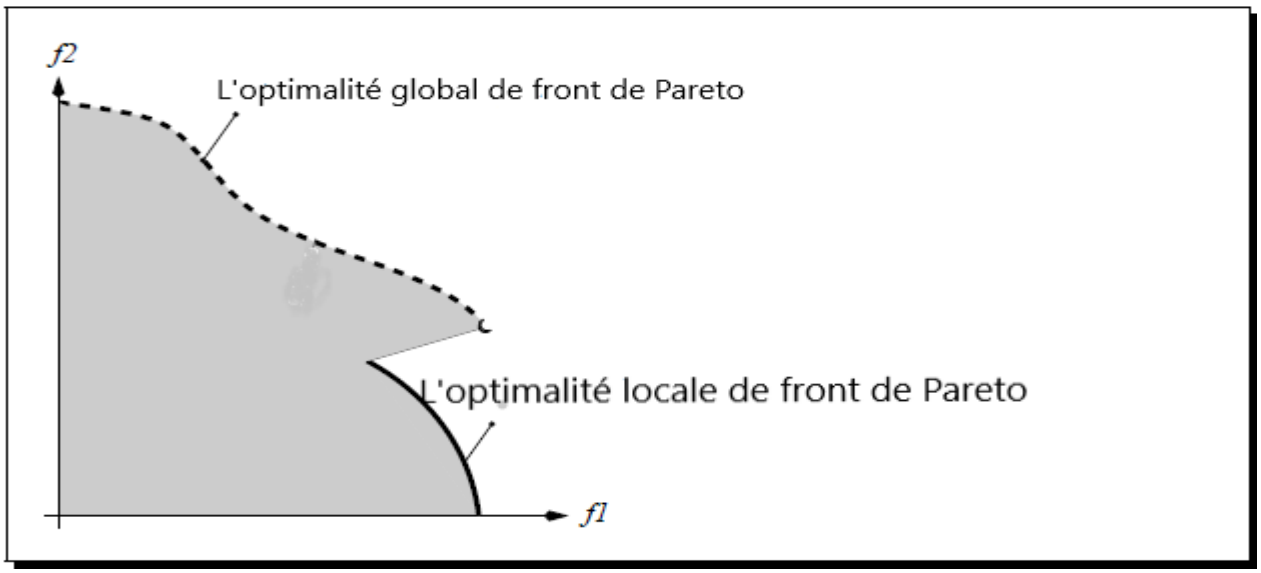


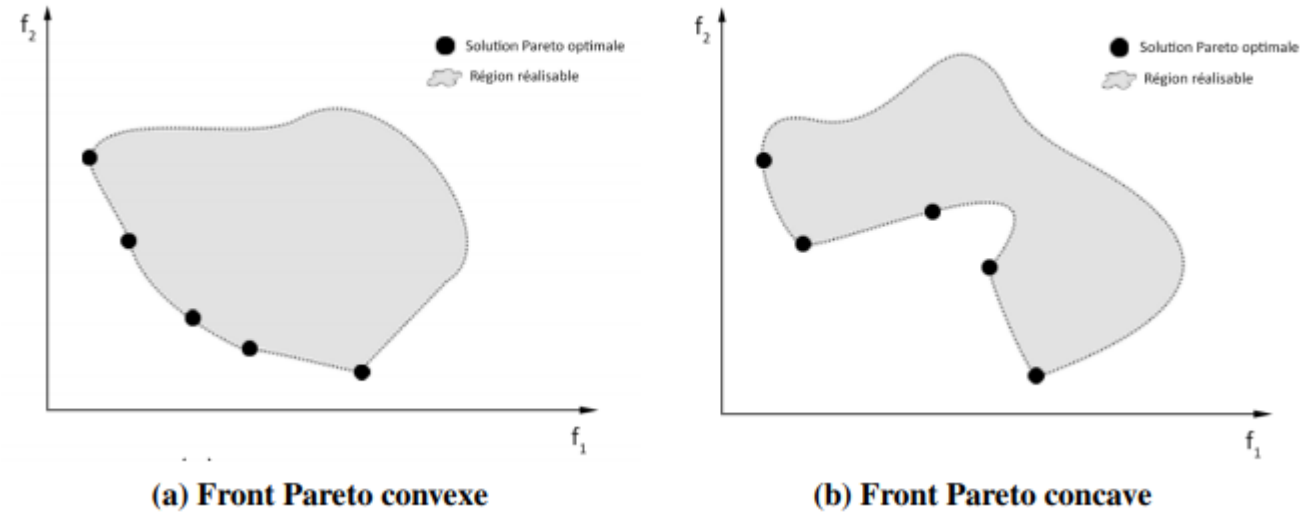
Figure1.3 :Illustration d'ensembles de solutions localement optimales et d'ensembles de solutions globalement optimales en objectif espace [14].

1.4.2.4 Caractéristiques du front Pareto.

La structure de front Pareto peut être convexe ou non-convexe (concave) et continue ou discontinue. Pour un problème à deux objectifs, un front Pareto est dit convexe si, dans l'espace objectif, tout segment qui lie deux solutions Pareto optimales est inclus dans la région des solutions réalisables. Cette définition peut être étendue à plus de deux objectifs en augmentant le nombre de dimensions [15].

- **La convexité**

Certaines méthodes d'optimisation multi-objectif nécessitent de respecter certaines hypothèses. Le plus souvent, la méthode réclame de travailler sur un espace *S* des valeurs de $\vec{f}$ qui soit convexe.
Un ensemble *S* est convexe si, étant donnés deux points distincts quelconques de cet ensemble, le segment qui relie ces deux points est contenu dans l'ensemble *S* [12].



(a) Front Pareto convexe **(b) Front Pareto concave**

Figure1.4 : Exemples de front convexe et concave d'un problème bi-objectif [15]

1.4.3 Méthodes de résolution multi objectifs traditionnelles

Les méthodes classiques de génération de l'ensemble Pareto-optimal regroupent les objectifs en une seule fonction objectif paramétrée par analogie avec la prise de décision avant la recherche.

1.4.3.1 Somme pondérée (Weighted Sum)

$$maximiser\ y = f(x) = w1.f1(x) + w1.f2(x) + \cdots + wk.fk(x))$$

$$x \in X_f$$

Les wi sont appelés poids et sans perte de généralité, normalisés tels que $\sum wi = 1$. La résolution du problème d'optimisation ci-dessus pour un certain nombre de combinaisons de poids différentes donne un ensemble de solutions.

À condition qu'un algorithme d'optimisation exact soit utilisé et que tous les poids soient positifs, cette méthode ne générera que des solutions Pareto-optimales qui peuvent être facilement affichées. Supposons qu'un vecteur de décision réalisable a maximise f pour une combinaison de poids donnée et ne soit pas optimal de Pareto. Alors, il y a une solution b qui domine a, c'est-à-dire

sans perte de généralité $f1(b) > f2(a)$ et $fi(b) \geq fi(a)$ pour $i = 2, \ldots, k$. Par conséquent, $f(b) > f(a)$, ce qui est en contradiction avec l'hypothèse que $f(a)$ est maximum [14].

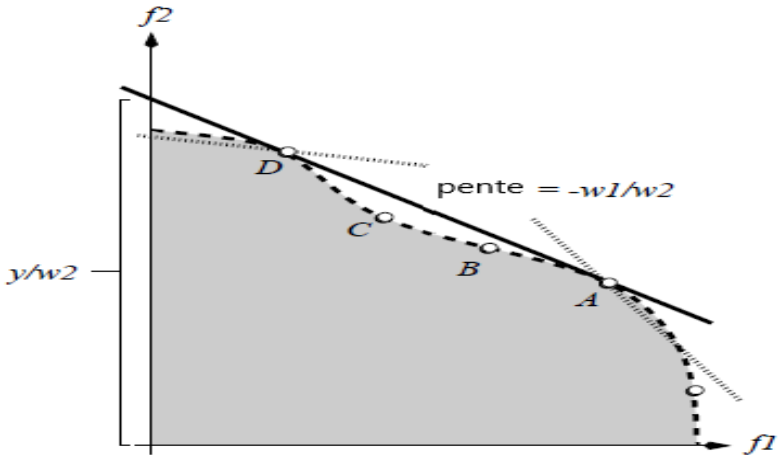


Figure 1.5 :Interprétation graphique de la méthode de pondération [14].

1.4.3.2 ε–contrainte

La méthode des contraintes ε proposée par Chankong et Haimes en 1983, le décideur choisit un objectif parmi n à minimiser ; les objectifs restants doivent être inférieurs ou égaux à des valeurs cibles données. En termes mathématiques, si on laisse $f2\ (x)$ la fonction objectif choisie pour être minimisée, on a le problème $P\ (\varepsilon 2)$ suivant :

$$\begin{gathered} min\ f2(x) \\ fi(x)\ \leq \varepsilon i, \forall i\ \in\ \{1, \ldots, n\}\backslash\{j\} \\ x\ \in\ S. \end{gathered}$$

Un avantage de la méthode des contraintes ε est qu'elle permet d'atteindre des points efficaces dans une courbe de Pareto non convexe.

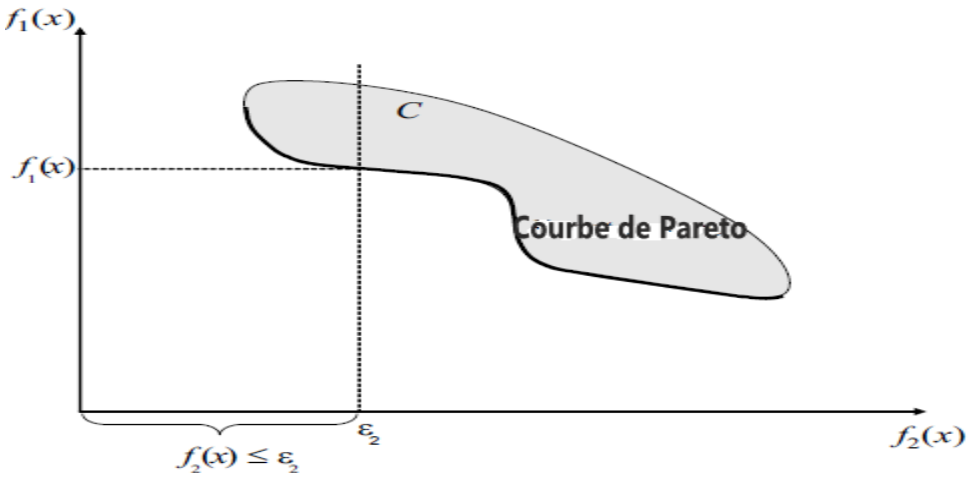


Figure 1.6 : Représentation géométrique de l'approche des contraintes ε dans le cas de la courbe de Pareto non convexe [16].

Cette approche a l'avantage dans les problèmes non convexes, mais présente plusieurs inconvénients à savoir :

- La formulation des préférences utilisateur est délicate et nécessite une connaissance approfondie du problème de départ.
- Les contraintes rajoutées compliquent la résolution du problème.

1.5 Méthodes Métaheuristiques

Le mot métaheuristique est dérivé de la composition de deux mots grecs :

- Heuristique qui vient du verbe "heuriskein" et qui signifie "trouver"
- Méta qui est un suffixe signifiant "au-delà", dans un niveau supérieur.

Les Métaheuristiques sont des stratégies algorithmiques de haut niveau utilisées pour guider les algorithmes dans leur recherche dans l'espace des solutions possibles de la valeur optimale (dans le cas mono-objectif) ou de l'ensemble des valeurs optimales (dans le cas multicritère) [22].

Les métaheuristiques sont un ensemble d'algorithmes d'optimisation visant à résoudre les problèmes d'optimisation difficiles. Elles sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas du recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou de l'optimisation par essaims particulaires (cas des algorithmes de colonies d'abeilles). Les métaheuristiques ont montré une grande capacité d'adaptation à plusieurs problèmes dans divers domaines [23].

1.5.1 Le recuit simulé

Le recuit stimulé (Simulated annealing SA) est une technique de résolution des problèmes d'optimisation proposée en 1983 par Kirkpatrick et al. Cette méthode est basée sur la technique du recuit, utilisée en métallurgie, qui consiste à chauffer un solide métallique jusqu'à sa température

de fusion, puis à le refroidir lentement jusqu'à la température ambiante, afin que les atomes aient le temps de prendre des positions qui minimisent l'énergie [1].

L'algorithme simulant ce processus consiste à perturber d'une manière aléatoire, à une certaine température, la position des atomes de manières à avoir un nouvel état d'énergie. Dans le cas d'une augmentation d'énergie, une décision probabiliste juge l'acceptation ou le rejet de cet état. Les perturbations sont alors générées, jusqu'à l'obtention d'un état d'énergie minimale. Ces étapes de l'algorithme SA on peut résumer sous forme de l'organigramme de la figure 1.3 Le pseudo-code du recuit simulé est représenté dans l'algorithme suivant : [1]

Algorithme 1.1 : Recuit simulé

init T (température initiale)
init x (point de départ)
init ΔT (température minimale)
Tantque (non(fin))
$y = Voisin\ (x)$
$\Delta C = C(y) - C(x)$
si $\Delta C < 0$ alors $y = x$
sinon si $alea(0,1) < e^{-\frac{\Delta C}{T}}$ alors $y = x$
$T = \alpha(T)$
si $T < \Delta T$ alors end(while)
repeter Tantque (while)

– On commence par choisir un point de départ au hasard (x).
– On calcule un voisin de ce point ($y = Voisin\ (x)$).
– On évalue ce point voisin et on calcule l'écart par rapport au point d'origine ($\Delta C = C(y) - C(x)$).
– Si cet écart est négatif, on prend le point y comme nouveau point de départ, s'il est positif, on peut quand même accepter le point y comme nouveau point de départ, mais avec une probabilité $e^{-\frac{\Delta C}{T}}$ (qui varie en sens inverse de la température T).
– Au fur et à mesure du déroulement de l'algorithme, on diminue la température T ($T = \alpha\ (T)$), souvent par paliers.
– On répète toutes ces étapes tant que le système n'est pas figé (par exemple, tant que la température n'a pas atteint un seuil minimal).

1.5.2 Algorithmes Génétiques

Les algorithmes génétiques ont pour base la théorie de l'évolution et les règles de la génétique qui démontrent l'habilité des espèces vivantes à s'adapter à leur environnement par la combinaison des mécanismes suivants [22],[23] :

- **Sélection**

La sélection naturelle montre que les individus les mieux adaptés à l'environnement ont tendance à survivre plus longtemps et ont donc une plus grande probabilité de se reproduire ; donc, cette phase a pour but de sélectionner les combinaisons de la population qui seront ensuite élues pour la recombinaison et la mutation. Il s'agit là de conserver la sélection des meilleures combinaisons, tout en accordant une petite chance aux moins bonnes. Il y a une multitude de méthodes de procéder à cette phase de sélection.

- **Reproduction (le croisement)**

Le croisement fait qu'un individu hérite ses caractéristiques de ses parents, de sorte que le croisement de deux individus bien adaptés à leur environnement aura tendance à créer un nouvel individu bien adapté à l'environnement ; Il existe deux méthodes de croisement : simple ou double enjambement.

- croisement en 1-point : consiste à fusionner les particularités des deux individus à partir d'un pivot afin d'obtenir un ou deux enfants.
- croisement en 2-point : repose sur le même principe, sauf qu'il y'a deux pivots.

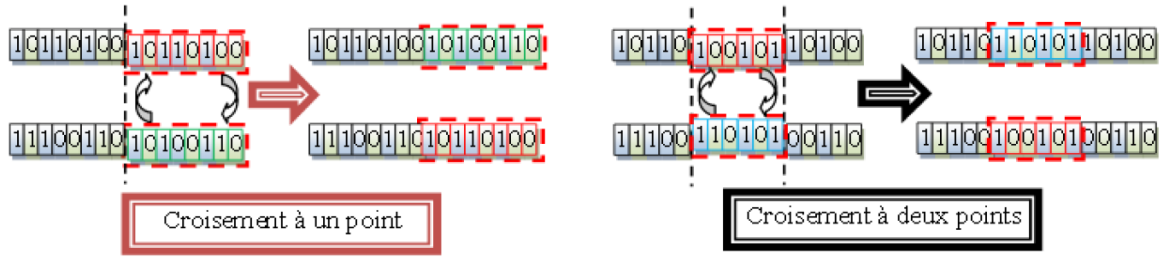


Figure 1.7: Représentation de l'opération de croisement

- **Mutation**

Elle fait que certaines caractéristiques peuvent apparaître ou disparaître de façon aléatoire, permettant ainsi d'introduire de nouvelles capacités d'adaptation à l'environnement, capacités qui pourront se propager grâce aux mécanismes de sélection et de croisement. La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité pour qu'une mutation soit effectuée sur un individu.

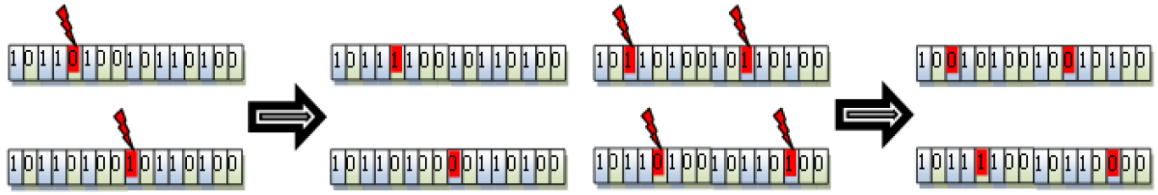


Figure1.8: Représentation de l'opération de mutation

Les algorithmes génétiques reprennent ces mécanismes pour définir une méta-heuristique. L'idée est de faire évoluer une population de combinaisons, par sélection, croisement et mutation, la capacité d'adaptation d'une combinaison étant ici évaluée par la fonction objective à optimiser. On peut résumer ce principe général sous forme d'un organigramme illustré dans la figure 1.9.

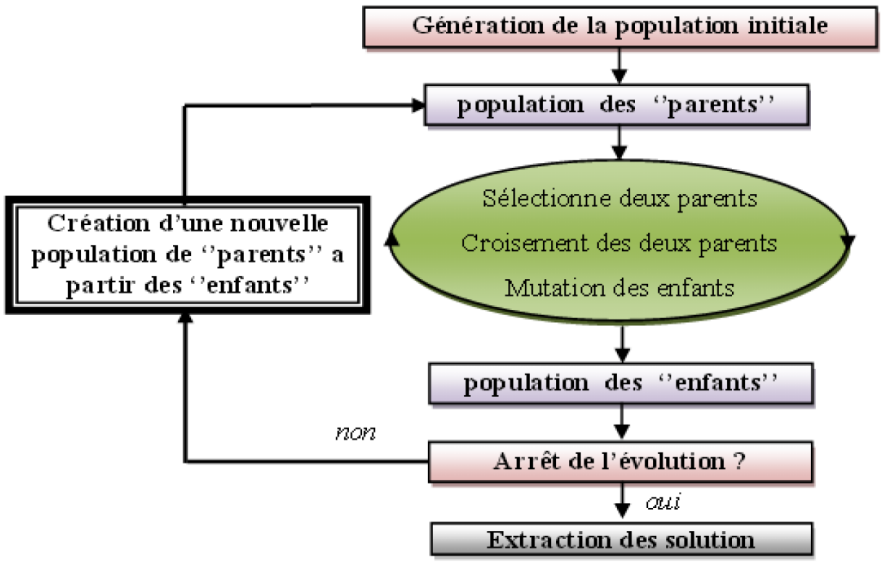


Figure 1.9: Organigramme de l'algorithme génétique [23].

1.5.3 Optimisation par Colonie d'abeilles

L'optimisation par colonie d'abeilles est une famille très récente des métaheuristiques. Son principe est basé sur le comportement des abeilles réelles dans la vie. Cette approche de résolution fait l'objet de notre étude [27].

1.5.3.1 Les abeilles en nature

Les abeilles possèdent des propriétés assez différentes de celles des autres espèces d'insectes. Elles vivent en colonies, en construisant leurs nids dans des troncs d'arbres ou d'autres espaces clos similaires [28]. Généralement, une colonie d'abeilles contient une femelle reproductrice appelée reine, quelques centaines de mâles connus sous le nom de faux-bourdon, et de 10.000 à 80.000 femelles stériles qui s'appellent les ouvrières. Après accouplement avec plusieurs faux-bourdons, la reine reproduit beaucoup de jeunes abeilles appelées les couvées [27].

1.5.3.2 Recherche de nourriture chez les abeilles

Les scientifiques ont effectué beaucoup de recherches pour déterminer comment l'ordre est maintenu dans la ruche où vivent des dizaines de milliers d'abeilles. Un grand nombre d'études académiques ont été aussi effectuées à cette fin. Un éminent expert et professeur à l'Université de Munich, le zoologiste autrichien Karl Von Frisch, a consacré un livre à la communication des abeilles, "The dance language and orientation of bees" (Le langage de la danse et l'orientation des abeilles) [29].

1.5.3.3 L'algorithme d'abeilles (Bees Algorithm BA)

Algorithme d'abeilles (Bee Algorithm ou BA) a été introduit en 2005 cet algorithme a le pouvoir de simuler le comportement alimentaire des abeilles par le biais d'une recherche de voisinage combinée avec une recherche aléatoire en se basant sur une population initiale d'abeilles qui est divisée en deux groupes: **les scouts** qui sont responsables de l'exploration de l'espace de recherche alors que **les recrues** sont chargés d'exploiter cet espace pour localiser des solutions via une recherche locale.[8]

L'idée de base de l'algorithme des abeilles est issue de l'observation du comportement d'un seul type d'abeilles, à savoir les ouvrières, pendant la récolte du nectar des fleurs. Le comportement de recherche de nourriture est illustré dans la figure -10, ci-dessous [28].

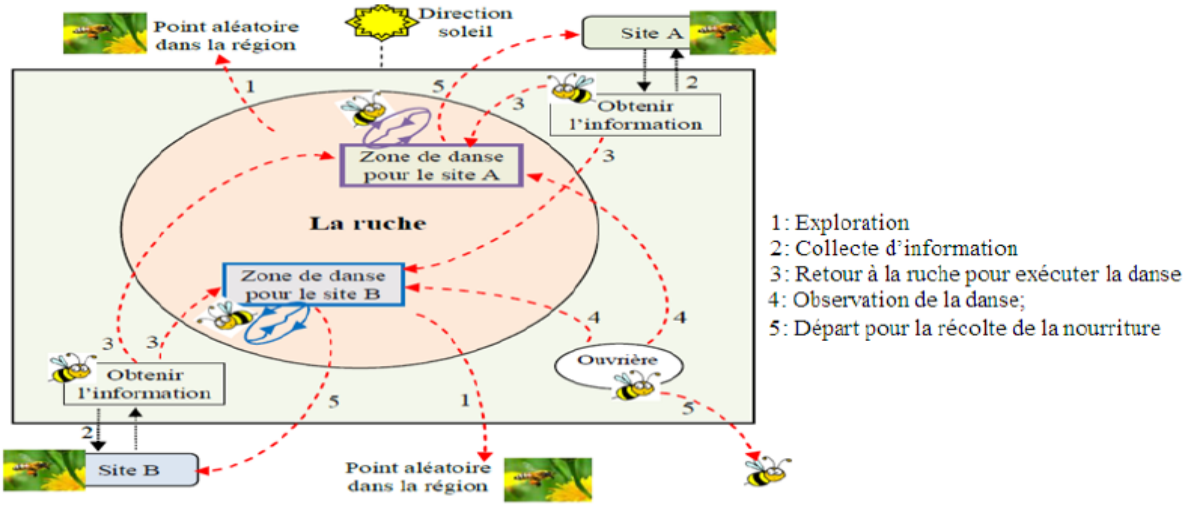


Figure 1.10 : Comportement typique de la recherche de nourriture dans une colonie d'abeilles [28].

On peut définir les étapes de recherche de cet algorithme comme suit [8] :

Algorithme 1.2 : algorithme BA

Placer chaque abeille sur une position aléatoire dans l'espace de recherche ;
Évaluer le fitness de la population ;
Tant que le critère d'arrêt n'est pas satisfait **faire**
    Choisir des solutions pour une recherche locale (exploitation) ;
    Affecter les abeilles pour commettre la recherche locale sur les solutions choisies et la fitness évaluée ;
    Pour chaque solution, sélectionner la meilleure amélioration ;
    Remplacer les solutions restantes avec des solutions aléatoires (scouts) ;
Fin Tant que

1.5.3.4 L'algorithme de colonies des abeilles artificielles

L'algorithme de colonie d'abeilles artificielles ou Artificial Bee Colony (ABC) a été introduit par Dervis Karaboga et développe depuis 2005 par Karaboga et Basturk pour les problèmes d'optimisation continue. C'est un algorithme a population, d'inspiration naturaliste, basé sur le butinage des abeilles [21].

Dans l'algorithme des colonies d'abeilles artificielles (ABC) y a des cycles de recherche se compose de trois étapes après l'étape d'initialisation : placer les abeilles employées **(les ouvrières)** sur les sources de nourriture et calculer leurs quantités de nectar ; placer **les spectatrices** sur les sources de nourriture et calculer les quantités de nectar ; et déterminer les abeilles scoutes **(les exploratrices)** et les placer sur les sources de nourriture déterminées au hasard.

Dans l'ABC, une position de source alimentaire représente une solution possible au problème à optimiser. Lors de l'initialisation, un ensemble de positions de source de nourriture est produit de manière aléatoire et les valeurs des paramètres de contrôle de l'algorithme sont attribuées. La quantité de nectar récupérable à partir de la source de nourriture correspond à la qualité de la solution représentée par cette source de nourriture. Ainsi, les quantités de nectar des sources de nourriture existantes aux positions initiales sont déterminées. En d'autres termes, la qualité (valeurs) des solutions initiales est calculée. Chaque abeille employée est déplacée vers sa zone de source de nourriture pour déterminer une nouvelle source de nourriture dans le voisinage de l'emplacement actuel, puis sa quantité de nectar est évaluée. Si la nouvelle quantité de nectar est plus élevée, elle oublie la quantité précédente et se souvient de la nouvelle. Une fois que les abeilles employées ont terminé leur recherche, elles reviennent dans la ruche et partagent leurs informations sur les quantités de nectar de leurs sources avec les spectateurs qui attendent sur la zone de danse. Tous les spectateurs déterminent successivement une zone de source de nourriture avec une probabilité basée sur leurs quantités de nectar. Chaque spectateur détermine une source de nourriture voisine dans le voisinage de celui auquel elle a été assignée, puis sa quantité de nectar est évaluée [30].

1.5.3.5 L'algorithme de colonies des abeilles artificielles multi-objectif (MOABC)

La colonie d'abeilles artificielles multi-objectif (MOABC) est l'un des algorithmes méta-heuristiques les plus récents. Il s'agit d'une nouvelle approche basée sur la population qui a montré de bonnes performances dans le traitement de différents types de problèmes d'optimisation.

Dans cet algorithme MOABC, A food source position représente une solution possible au problème à optimiser. La quantité de nectar d'une source de nourriture correspond à la qualité de la solution représentée par cette source de nourriture. La colonie d'abeilles artificielles est classée en trois types avec certaines responsabilités :[31]

- Abeilles employées.
- Abeilles spectatrices.
- Abeilles scoutes.

La structure algorithmique de l'optimisation MOABC proposée est donnée comme suit : [31]

- **Initialisation**

Au stade de l'initialisation, les abeilles scoutes initialisent un ensemble de sources de nourriture (NCS) de manière aléatoire et l'archive externe est bien initialisée. NCS est égal au nombre de taille de la colonie de l'abeille employée et de l'abeille spectateur. Chaque membre de la population représente une solution au problème, noté par $food_i (i = 1 \dots NCS)$.

- **Les abeilles employées**

Au stade des abeilles employées, pour chaque abeille employée trouve une nouvelle position de source de nourriture (sol_i) au voisinage de sa position de source de nourriture actuelle ($food_i$).
Chacune des abeilles employées compare la position actuelle de la source de nourriture avec une position de source de nourriture voisine et choisit la meilleure en utilisant une technique de sélection gourmande. Les solutions non dominées sélectionnées sont stockées dans une archive externe.

- **Les abeilles spectateurs**

Au stade des abeilles spectateurs, pour chaque abeille ($food_i$), elle est sélectionnée au hasard et apprend à partir d'une archive externe produite par les abeilles employées, Après avoir produit la nouvelle solution, la sélection gourmande est appliquée pour décider quelle solution entre dans l'archive externe.

- **Les abeilles scoutes**

Au stade des abeilles scoutes, l'abeille scoute est chargée de trouver une nouvelle position de source de nourriture et d'évaluer la qualité de leur nectar. lorsque l'abeille scoutes trouve la source de nourriture, elle se convertit en abeille employée. Dans cette étape, si la source de nourriture n'est pas améliorée par le numéro de trialles (le numéro de trialles pour une source de nourriture de libération est supérieur à un paramètre de contrôle « Limite »), alors cette source de nourriture est épuisée par son abeille employée et celle -ci devient une abeille scoute.

On peut définir cette structure de recherche de cet algorithme comme suit :[40]

Algorithme 2.2 : algorithme MOABC

MOABC (Population, Limit, Iff, Taille de population ,nbr de génération)
Initialisation
Trial = [0,0,….,0]// i :1,…,m m = Taille de population
Génération de la population initiale
Calculé la fitness de chaque individu dans la population
génération = 1
Répéter

(1) Envoyer des abeilles employées
Pour i = 1 à Taille de de population **faire :**
Parent1 = individu[i]
Parent2 = choisi un individu aléatoirement de puis la Population tel que
Parent2 ≠ Parent1
Point de croisement = aléatoire[1,longeur(individu[i]]
Enfant = croisement(Parent1, Parent2,Point de croisement
Calculé fitness (Enfant)
Si Enfant.fitness ≥Parent1.fitness **alors :**
Individu[i] = Enfant
Sinon :
Trial[i] = Trial + 1
(2) Envoyer des abeilles spectateurs
Iff = aléatoire]−1,1[
Pour i = 1 à taille de population **faire** :
Calculé la probabilité de l'individu[i]
Si la probabilité de l'individu[i] ≥ iff **alors :**
Parent1 = individu[i]
Parent2 = choisi un individu aléatoirement de puis la Population tel que
Parent2 ≠ Parent1
Point de croisement = aléatoire[1,longeur(individu[i]]
Enfant = croisement(Parent1, Parent2,Point de croisement
Calculé fitness (Enfant)
Si Enfant.fitness≥Parent1.fitness **alors :**
Individu[i] = Enfant
Sinon
Trial[i] = Trial + 1
(3) Envoyez des abeilles scoutes
Pour k = 1 à taille de population **faire** :
Si Trial[k] >= Limit **alors :**
Ajouter la meilleure solution au front
Individu[k] = nouveau individu généré
Trial[k] = 0
génération = génération+1
Jusqu'à (génération == nbr de génération)
(4) Arrangement du front
Appliqué la méthode fast non dominted sorting sur le front
Retourner f_0:
f_0: est le front qui contient les meilleures solutions

1.6 Conclusion

Ce chapitre a pour objectif de présenter les principales définitions nécessaires à la présentation des problèmes d'optimisation multi-objectif. On a parlé sur les problèmes d'optimisation multi-objectif selon l'intervention du décideur dans le processus de décision. On s'est penché sur les méthodes utilisées dans le domaine de l'optimisation multi objectif essentiellement les méta-heuristiques. Parmi elles, on trouve le MOABC que nous allons exploiter pour optimiser le système manufacturier reconfigurable.

chapitre 2 :

Systèmes Manufacturiers Reconfigurables (RMSs)

2.1 Introduction

L'évolution rapide du marché des produits conduit à l'apparition de Systèmes Manufacturiers Reconfigurables ou Reconfigurables Manufacturing Systems (RMS)

2.2 Les systèmes manufacturiers

Un système est un ensemble composite constitué de personnels, de matériels, de logiciels et de procédures. Tous ces éléments sont en interaction mutuelle dans un environnement donné et sont organisés pour répondre à un besoin. Chaque système est déterminé par la nature de ses éléments constitutifs, les interactions entre ces derniers et le critère d'appartenance au système. .[39]

Dans le système de production, il y a les ressources humaines et les ressources physiques. En d'autres termes, on peut le définir comme un ensemble d'opérateurs et de postes de travail et/ou de machines qui sont intégrés. Le rôle de ces machines est d'effectuer une série contrôlée d'opérations répétitives sur les matières premières pour obtenir une forme finale souhaitée ou encore pour assembler un ensemble de pièces afin d'obtenir un produit final. Les postes de travail et/ou les machines sont connectés par l'intermédiaire d'un mécanisme de transfert qui peut être un convoyeur, un robot mobile (AGV : Automated Guided Vehicle) ou tout simplement un opérateur.[39]

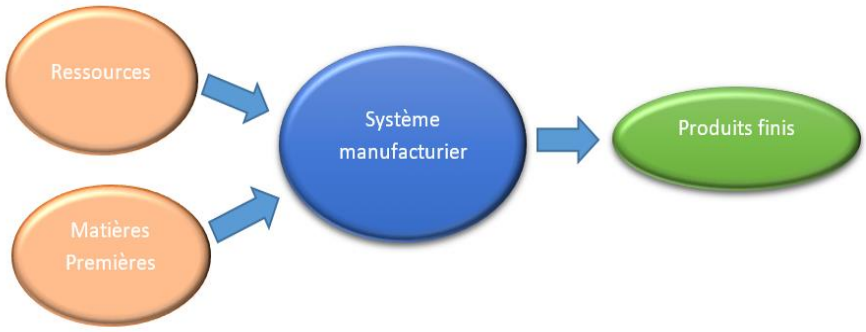


Figure 2.1: Schéma classique d'un système manufacturier.

2.3 Systèmes dédiés et systèmes flexibles

A la fin du 20ème siècle on distinguait deux types principaux de systèmes manufacturiers DMS et FMS, auxquels viennent de s'ajouter les systèmes manufacturiers reconfigurables (Reconfigurable Manufacturing Systems - RMS) au début de ce nouveau millénaire.

2.3.1 Systèmes manufacturiers dédiées (Dedicated Manufacturing Systems - DMSs)

Les Systèmes manufacturiers dédiées (DMSs), ou lignes de transfert, sont basées sur une automatisation fixe et produisent les produits ou pièces de base d'une entreprise à grand volume.

Lorsque le volume est élevé, le coût par pièce est relativement faible. Par conséquent, les DMSs sont rentables tant que la demande du marché correspond à l'offre ; mais avec la pression croissante de la concurrence mondiale, il existe de nombreuses situations dans lesquelles les lignes dédiées ne fonctionnent pas à pleine capacité et créent ainsi des pertes. Bien sûr, produire une variété de produits est impossible avec un DMS, et par conséquent, leur rôle dans la fabrication moderne se détériore. [6]

2.3.2 Système Manufacturiers Flexibles (Flexible Manufacturing Systems-FMSs)

Le système de fabrication flexible est la combinaison d'un module de machine de fabrication et d'un équipement de manutention contrôlé par des systèmes informatiques pour le traitement automatique afin d'obtenir un coût de changement minimum. Programmables et peuvent produire une variété de produits sur le même système. Malgré cet avantage, les systèmes flexibles n'ont pas été largement adoptés et que de nombreux fabricants qui ont acheté des FMS ne sont pas satisfaits de leurs performances.[36]

Les inconvénients des FMS sont : [7]

- Ils nécessitent des machines plus chères que les DMS.
- La raison du fonctionnement à un seul outil des machines CNC.
- Le taux de production des FMS est très faible par rapport à leurs homologues DMS.
- La capacité de production des FMS est généralement inférieure à celle des lignes dédiées.
- Elles ne sont pas conçues pour un changement rapide de leur capacité, c'est-à-dire qu'ils ne réagissent pas aux changements du marché.

DMS	FMS
Avantages: • Non coûteux • Rapide - fonctionnement multi-outils	**Avantages:** • convertible • Capacité évolutive
Limitations: • Non flexible - pour une seule pièce • Capacité fixe - non évolutif	**Limitations:** • coûteux • Lent - fonctionnement avec un seul outil

Tableau 2.1: Une simple comparaison entre les DMSs et FMSs [7]

2.4 Système manufacturier reconfigurable (Reconfigurable Manufacturing Systems-FMSs)

2.4.1 Définition d'un RMS

Un système manufacturier qui peut être créé en incorporant des modules de processus de base (à la fois matériels et logiciels) qui peuvent être réorganisés ou remplacés rapidement et de manière fiable. On peut définir le RMS comme un ensemble de machines à outils reconfigurables (RMTs). Il a la capacité de fabriquer au même moment plusieurs types de produits appartenant à une même famille et en quantités irrégulières. Le système peut être reconfiguré physiquement (hard) et/ou logiquement (soft). [5]

La reconfiguration permettra d'ajouter, de supprimer ou de modifier des capacités de processus, des contrôles, des logiciels ou une structure de machine spécifique pour ajuster la capacité de production en réponse à l'évolution des demandes ou des technologies du marché. Ce type de système offrira une flexibilité personnalisée pour une famille de pièces particulière et sera ouvert, de sorte qu'il puisse être amélioré, mis à niveau et reconfiguré, plutôt que remplacé.[34]

Koren a défini RMS comme « un système manufacturier reconfigurable (RMS) est un système conçu au départ pour un changement rapide de structure, ainsi que dans les composants matériels et logiciels, afin d'ajuster rapidement la capacité de production et les fonctionnalités au sein d'une famille de produits ». [1]

Le RMS a combiné la caractéristique du taux de production élevé du DMS et la caractéristique de flexibilité du FMS. Il est conçu pour répondre aux besoins d'un produit famille en changeant rapidement ses modules matériels et logiciels en réaction à la rapidité changements dans les tendances du marché. RMS est capable de produire une large gamme de produits en réorganisant le système de fabrication lui-même. Il se compose de modules matériels et logiciels rapidement ré-organisables, évitant de devenir obsolètes. [1]

À côté de cela, il est ouvert, il peut donc être mis à niveau fréquemment grâce à l'opération de nouvelles méthodes. Ce système prend en compte les exigences de la machine et sont conçus pour faire face aux circonstances dans lesquelles la productivité et la capacité du système sur le point de changer sa configuration pour produire une variété de produits et il a donc une grande importance pour répondre aux variations actuelles du marché. [1]

2.4.2 Caractéristiques d'un RMS

Un RMS possède certaines caractéristiques clés qui permettent un haut degré de réactivité du système aux besoins du marché. Ces caractéristiques, au nombre de six, devraient être intégrées dans le système reconfigurable dès la phase de conception pour assurer un degré élevé de reconfigurabilité. Ces caractéristiques sont : [2]

- **La personnalisation** (flexibilité limitée à une famille de produits) : c'est l'aptitude à adapter une flexibilité personnalisée (juste nécessaire au bon moment) du système et des machines pour répondre à de nouvelles spécifications d'une famille de produits.

- **La convertibilité** (conception pour des modifications de la fonctionnalité) : La capacité de transformer facilement la fonctionnalité du système et des machines existantes pour répondre aux nouvelles exigences de production.

- **L'évolutivité** (conception pour des changements de capacité) : La possibilité de modifier facilement la capacité de production en ajoutant ou soustrayant des ressources de production (par exemple des machines) et/ou changement des composantes du système.

- **La modularité** (les composants sont modulaires) : La classification des fonctions opérationnelles en unités qui peuvent être manipulées et échangées entre les systèmes de production pour un arrangement optimal.

- **L'intégrabilité** (interfaces simples pour une intégration rapide) : La capacité d'intégrer des modules rapidement et précisément à l'aide d'un ensemble d'interfaces mécaniques, informationnelles, et des interfaces de contrôle qui facilitent l'intégration et la communication.

- **La diagnosticabilité** (conception pour des diagnostics faciles) : c'est l'aptitude à lire automatiquement l'état courant du système et de la commande afin de détecter et de diagnostiquer les sources de défaillances et de les corriger rapidement.

2.4.3 Configuration et reconfiguration du RMS

2.4.3.1 Configuration du RMS

Une configuration est un état du système qui répond à un contexte particulier. Pour définir la configuration du système, on doit définir un ensemble de paramètres du système qui déterminent cette configuration. Le changement d'un paramètre implique le changement de la configuration du système.

Le nombre de machines, le nombre des lignes d'assemblage, le nombre de stations de chargement, le nombre de stations de déchargement, le nombre d'opérateurs, le positionnement de ces objets dans l'atelier. Sont quelques exemples de paramètres de configuration. [2]

2.4.3.2 Reconfiguration du RMS

La reconfigurabilité d'un système manufacturier est la possibilité de l'ajout, de la surpression ou de la modification des fonctionnalités, de la commande, de la structure des machines ou du processus du système en réponse aux changements de la demande du marché en produits, en volumes ou en technologie. [3]

2.4.3.3 Types de reconfiguration

Les différents types de reconfiguration qui touche un RMS, ces reconfigurations peuvent être [35]:

- Dans la structure du système : réorganiser les machines suivant une configuration série, parallèle ou hybride
- Une reconfiguration logicielle du système.
- Une reconfiguration dans le système de contrôle associé au système.
- Une reconfiguration au sein d'une machine du système : par exemple à jouter ou supprimer des modules, changement de quelques composants.
- Une reconfiguration dans le processus de fabrication du système.

2.4.4 Machines reconfigurables (Reconfigurable Machines-RMs)

Les machines reconfigurables (reconfigurable machine -RMs) sont des machines dont les structures peuvent être changées pour fournir une fonctionnalité alternative et/ou la capacité de la mise à jour selon la demande. Elles sont toujours conçues en fonction des caractéristiques communes de la famille de produits [5].

2.4.4.1 Machines outils reconfigurables (Reconfigurable Machines Tools-RMTs)

Les RMTs ont été développés comme des machines modulaires comprenant différents modules qui sont assemblés par des combinaisons de modules de base et auxiliaires selon les exigences de fabrication. Les modules de base sont de nature structurelle, par ex. base de la machine, colonnes et glissières, tandis que les modules auxiliaires à savoir. Changeurs d'outils, têtes de broche, la structure d'angle, les plaques d'adaptation, les espaceurs et les unités d'indexation, etc. sont des modules de mouvement ou cinématiques. Les modules auxiliaires sont relativement plus légers, plus petits et bon marché que les modules de base, de sorte qu'ils peuvent être changés économiquement et rapidement d'une configuration à une autre avec moins d'effort. Les RMTs peuvent être transformés en plusieurs configurations offrant ainsi une fonctionnalité et une capacité réglables en conservant ses modules de base tels quels et en ajoutant, supprimant ou réajustant les modules auxiliaires.[1]

Un RMS implique plusieurs machines-outils reconfigurables (RMTs), qui ont été développées comme des machines modulaires avec des caractéristiques « plug and play ». RMT joue un rôle majeur car ils sont considérés comme des catalyseurs clés pour la conception et le développement des RMS.

La machine-outil joue un rôle primordial dans les systèmes manufacturiers reconfigurables. En effet, un grand effort a été apporté à la modernisation et la sophistication tant au niveau mécanique qu'au niveau de commande et de contrôle des machines outil depuis le début du 20ème siècle. Nous présenterons les principaux caractéristiques des machines-outils, une machine standard est composée de plusieurs éléments : [38]

- Le bâti : il sert d'organe porteur des différents éléments de la machine et de la pièce à usiner.
- La broche : elle permet de faire tourner l'outil lors de l'usinage. La broche est l'un des éléments les plus délicats lors de la réalisation d'une machine-outil.
- Le porte-outil : il assure la fixation de l'outil sur la broche.
- L'outil : c'est l'instrument d'usinage. À chaque procédé d'usinage correspond un outil approprié.
- Dispositif de fixation : la pièce est placée de façon à permettre aux outils d'effectuer les usinages nécessaires.

Dans la Figure 2.2 un exemple d'un RMT. La pièce à manufacturer est située sur un support qui peut se déplacer simultanément sur les deux axes X et Y. Les broches Z1et Z2 peuvent se déplacer linéairement sur leurs axes respectifs, situés sur le même support. La broche Z3 peut usiner la pièce sur un axe horizontal avec des différents angles. Les broches peuvent être rapidement ajoutées ou enlevées en fonction des besoins. Dans la deuxième configuration, on a déplacé le support horizontal du slot B au slot C et le support horizontal a également changé de position. Cette machine, avec ces différentes configurations possibles, a accès à plus de points d'usinage par rapport à une machine classique, et une productivité plus importante par rapport à une machine CNC ayant un outil unique [22].

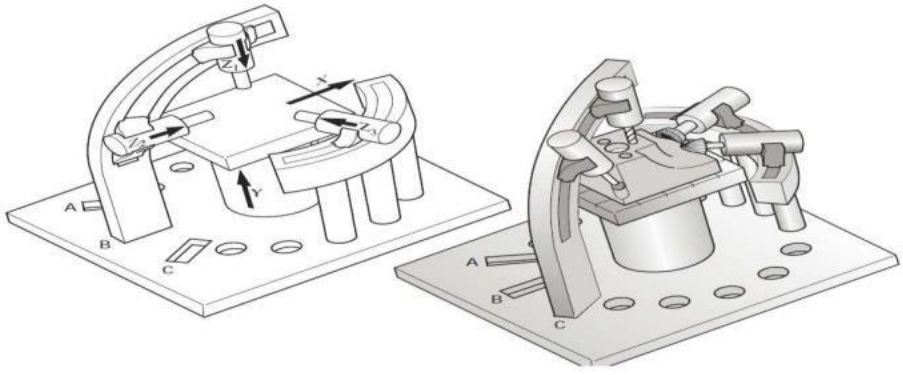


Figure 2.2 : Exemple d'une machines-outils reconfigurable (RMT).[22]

2.4.5 Lignes de production

Détermine les composants nécessaires, les machines et leur ordonnancement pour effectuer toutes les opérations d'un produit particulier en peu de temps et à faible coût. En d'autres termes, la ligne de production assigne à chaque machine un ensemble d'opérations qui peuvent être effectuées sur cette machine pour obtenir le produit souhaité. La figure ci-dessous représente un exemple illustratif d'une ligne de production de 4 opérations. La deuxième colonne représente l'ensemble de machines avec leurs configurations et l'outil disponible (Machine 2(M2)-Configuration 2(C2)-Outil 2(T2)), donc le problème est parmi lesquelles elle a la capacité de réaliser l'opération 2. Donc, pour atteindre le produit final il faut passer à travers une ligne de production commençant par la matière première.

Solutions Réalisable no.	Configuration de la machine réalisable pour la séquence d'opération 2→5→7→15→8→16					
	2	**5**	**7**	**15**	**8**	**16**
1	M2-C2-T2	M1-C2-T3	M1-C3-T4	M2-C2-T4	M1-C1-T3	M1-C1-T3
2	M2-C4-T4	M2-C4-T2	M2-C4-T1	M3-C2-T1	M2-C3-T2	M1-C3-T4
3	M1-C3-T2	M5-C2-T2	M5-C1-T1	M5-C3-T5	M3-C2-T1	M4-C3-T5
4	M4-C3-T3	-	M5-C4-T5	-	M4-C3-T1	M5-C2-T2

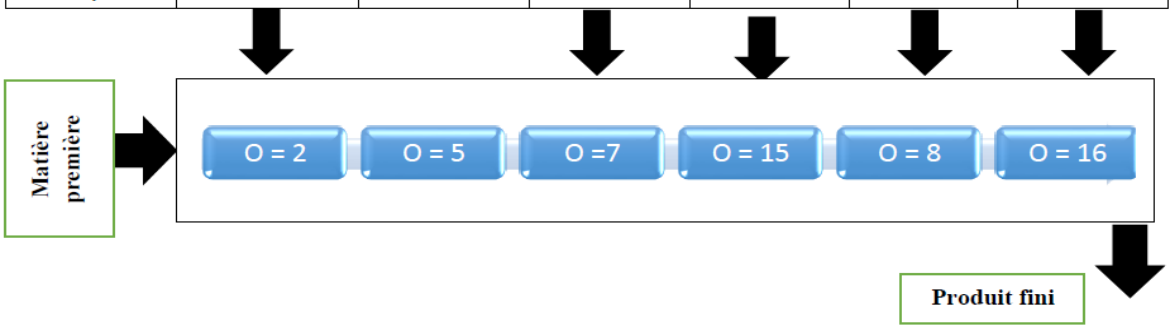


Figure 2.3: Exemple d'une ligne de production [1].

2.4.6 Conception des RMSs

La Figure 1 présente un aperçu général du processus de conception d'un système de production. Tout système de production, y compris un RMS, est destiné à fabriquer un ensemble de produits. Donc, la première étape consiste à définir ces produits. Ensuite, le processus de fabrication (ligne de production dans notre cas) de chaque produit doit être détaillé. Il y a un lien très fort et une influence mutuelle entre le produit et son processus de fabrication. Une fois que le processus de fabrication de chaque produit est établi, on commence la troisième phase qui est la conception du système de production. [2]

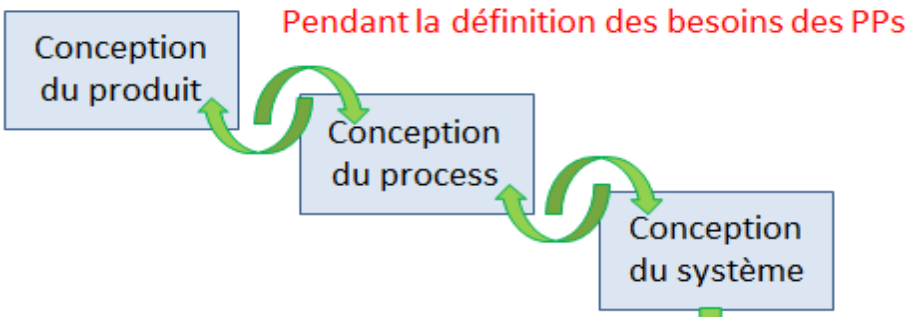


Figure 2.4 : Démarche générale de conception d'un système manufacturier. [2]

2.4.6.1 Principes de conception des RMSS

La conception d'un RMS demande l'application d'une vision à long terme du système de production, pour assurer la faisabilité économique de plusieurs générations de produits et de situations de marché. En d'autres termes, les RMSs doivent être conçus selon les principes de la reconfiguration. Plus ces principes sont applicables à un système de production, plus ce système est reconfigurable

- Un RMS doit fournir des ressources de production réglables pour répondre aux changements imprévisibles du marché et les évènements intrinsèques du système.
- Un RMS doit être conçu autour d'une famille de produits, avec juste suffisamment de flexibilité personnalisée pour produire tous les membres de la famille.
- Les caractéristiques de base d'un RMS devraient être intégrées dans le système dans son ensemble, ainsi que dans ses composants (physiques et logiques).
- La capacité d'un RMS doit pouvoir être rapidement ajustée (incrémenté ou décrémenté) par petits incréments.
- La fonctionnalité d'un RMS doit pouvoir être adaptée rapidement à de nouveaux produits.
- Les capacités d'ajustement intégrées d'un RMS doivent faciliter une réponse rapide à des défaillances imprévues du matériel [2].

2.4.6.2 Conception des produits et formations des familles de produits

La famille de produits est définie comme étant un ensemble de produits similaires partageant un certain nombre de parties, de composants et/ou de modules communs. L'architecture de la famille de produits est un concept très important qui facilite la réalisation de la personnalisation de masse. Avec cette architecture, le fabricant peut développer une stratégie de famille de produits où certains modules fonctionnels sont partagés tandis que d'autres sont fournis avec plusieurs variantes chacun, de sorte que la combinaison de ces derniers fournira une grande variété dans les produits finaux. Une telle approche permettra la fabrication de produits personnalisés demandés par le consommateur.

L'attribut clé dans une famille est le fait que tous les produits peuvent avoir besoin de ressources de production similaires. Ce qui implique qu'un système de production peut produire la totalité de cette famille de produits.[5]

2.4.6.3 Conception d'une ligne de production

La ligne de production constitue le rapport entre le produit (la conception) et le système de production (ressources du système de production). Cette ligne est attachée au produit à travers le lien entre les caractéristiques du produit et les opérations (chaque caractéristique du produit est réalisée en effectuant une ou plusieurs opérations à des machines dans cette ligne).

2.4.7 Les indicateurs de performance dans les RMSs

On trouve différentes mesures de performances utilisées pour évaluer la performance d'un système manufacturier reconfigurable « RMS » telles que le temps de production, le coût de production, la fiabilité du système, la disponibilité, la maintenance, le taux de productivité, les délais, le temps de reconfiguration, etc. Des recherches sont basées sur les mesures de performance et la façon de trouver la meilleure configuration pour les RMS. Dans la section travaux connexe, on a trouvé qu'il existe plusieurs travaux optimise le RMS grâce à l'utilisation des indicateurs de performance qui sont représentés comme des fonctions objectifs.

2.5 Comparaison entre DMS, FMS, RMS

Dans cette partie de chapitre, nous allons procéder à la comparaison des RMSs par rapport aux autres types de systèmes manufacturiers.
L'environnement concurrentiel mondial demande des systèmes fiables, performants et non-couteux, et les DMSs et FMSs sont incapable de satisfaire leurs demandes. La notion de système de production de type RMS englobe les caractéristiques positives des DMS et des FMS. Ajoutons à cela l'amélioration des points indésirables comme il est montré dans le Tableau 2.2 [39].

	DMS	RMS	FMS
Structure du système	Fixe	Modifiable	Modifiable
Structure de la machine	Fixe	Modifiable	Fixe
Réglage du système	Partie	Famille composants	Machine
Flexibilité	Non	Personnalisée	Générale
Évolutivité	Non	Oui	Oui
Exploitation des outils Simultanément	Oui	Oui	Non
Productivité	Elevée	Elevée	Faible
Coût	Faible	Moyen	Onéreux

Tableau 2.2: Comparaison entre les DMS, FMS et RMS [42].

2.6 Les travaux connexes

Cette partie abord brièvement quelques problèmes d'optimisation de la reconfigurabilité des RMS résolus par des méta-heuristiques. Voici quelques études :

Bensmain Abderrahman [22] ont focalisé sur le problème de génération des processus de fabrication optimales dans un RMS, utilisant au maximum les hauts degrés de reconfigurabilité des RMTs pour obtenir des plans efficaces. La fonction de génération des processus de fabrication comporte trois problèmes :

1) La génération des processus de fabrication dans un cas unitaire où ils ont adapté des techniques d'optimisation multi-objectif (NSGA-II et AMOSA).

2) La génération des processus de fabrication dans le cas multi unité où une optimisation basée sur la simulation a été adaptée.

3) L'intégration des fonctions de génération des processus de fabrication avec l'ordonnancement où ils ont développé une nouvelle heuristique permettant d'effectuer cette intégration.

Les expériences numériques démontrant l'efficacité des approches proposées.

Hichem Haddou Bendrbal [5], a travaillé sur la problématique de conception des systèmes manufacturiers reconfigurables (RMSs). Il a pour but de concevoir des systèmes réactifs en se basant sur leurs capacités en fonction de reconfigurabilité. Le travail a été élaboré sur deux niveaux :

1) le niveau des composantes relatif aux modules des machines reconfigurables.
2) le niveau des machines et leurs interactions ainsi que l'impact de ces interactions sur le système.

Afin d'assurer les meilleurs performances du système conçu telles que l'indicateur de modularité, l'indicateur de flexibilité, l'indicateur de robustesse et l'effort d'évolution d'un système reconfigurable, des indicateurs de performance ont été développés pour chaque niveau. Ils ont développé des modèles d'optimisation multi-objectif résolus à travers des méta-heuristiques multi-

objectif (AMOSA) et (NSGA-II). De nombreuses expériences numériques et analyses ont été réalisées afin de démontrer l'applicabilité des approches.

Ashraf et al [1] ont résolu le problème d'optimisation multi-objectif dans l'environnement RMS considérant quatre indicateurs de performance (fonctions objectif) : le coût, la reconfigurabilité, capacité opérationnelle et fiabilité. Ce problème est résolu par l'utilisation de l'algorithme NSGA-II et dans la ligne de production les opérations à fabriquer sont associés à une RMT adéquate. Les ensembles de solutions non-dominées ainsi obtenus pour l'allocation des RMTs à la ligne de production par l'application de NSGA-II sont nombreux. Par conséquent, ces solutions sont ensuite classées par l'application d'une technique de tri de solution, nommé TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution).

Après l'analyse les travaux cités au-dessus, nous avons vus que la majorité des travaux utilise le NSGA-II comme une métha-heuristique pour générer leurs solutions optimales. Vu que les meilleures performances MOABC dans plusieurs problèmes d'optimisation et qu'il n'y a aucun travail l'utilise dans les systèmes manufacturiers reconfigurables. Pour cette raison, on va proposer un nouveau MOABC pour générer des lignes des productions optimales selon deux indicateurs de performance (fonctions objectifs) : le temps de productions, le cout de production.

Travaux	Méta-heuristiques Utilisées	Fonctions objectif évaluées (indicateurs de performance)
[22]	AMOSA / NSGA-II	Minimise le cout /le temps
[1]	NSGA-II /TOPSIS	Minimise le coût, Minimise la reconfigurabilité, Maximise la capacité opérationnelle Maximise la fiabilité
[5]	AMOSA/TOPSIS	Plusieurs indicateurs
Notre Travail	MOABC	Minimise le cout /le temps

2.7 Conclusion

La mondialisation entraîne des fluctuations du marché qui ont entraîné une grave instabilité pour de nombreuses entreprises manufacturières qui luttent pour survivre. Pour rester compétitives, les entreprises manufacturières doivent s'adapter aux changements rapides du nouvel environnement économique mondial. Donc, la réactivité de l'entreprise est essentielle pour rester rentable dans un monde en évolution rapide. L'entreprise devrait être équipée d'un système de production qui peut être rapidement changé et reconfiguré afin de pouvoir suivre les changements du marché.

chapitre 3 :

Conception du Système

3.1 Introduction

La conception du système est une activité itérative multi-niveau ; c'est l'escale la plus importante dans le processus de développement des logiciels. Ce processus consiste à représenter les fonctionnalités du système d'une manière qui permettre d'obtenir rapidement un ou plusieurs programmes réalisant ses fonctionnalités. Celui-ci ne concerne que l'étape d'analyse et de définition des besoins. Ainsi, on vise le design du produit logiciel souhaité par l'utilisateur dans le but d'atteindre une résolution meilleure du problème posé.

Dans notre système, nous allons utiliser la conception fonctionnelle descendante comme suit :

- Le problème à résoudre.
- La construction du schéma général du système et de ces principales unités.
- La conception détaillée du système en question.

3.2 L'objectif du système

L'objectif crucial de notre projet est de rechercher l'ensemble optimal des machines pour la fabrication d'un produit, qui repose sur deux principales tâches :

Séquencer les opérations et les affecter aux machines (rechercher des meilleures séquences des machines avec leurs configurations et leurs outils pour fabriquer le produit désiré). Donc, on recherche de minimiser le temps (temps de changement des machines, le temps de changement d'outils, le temps de changement de configurations, temps de traitement), le coût (coût d'utilisation des machines/outils, coût de changement des machines/outils, coût de reconfiguration), Ce problème appartient donc à la coté des problèmes d'optimisation multi-objectif.

Les modèles mathématiques utilisés pour calculer le coût et le temps sont inspirés des travaux [1][5][22].

Chaque produit a un ensemble de composants (F). Sur la base de ces caractéristiques et parmi un ensemble de machines reconfigurables disponibles (M), un ensemble de machines candidates est sélectionné selon ses fonctionnalités. Après, nous décidons quelles sont les machines les plus optimales dans la ligne de production.

L'algorithme de colonie d'abeilles artificielle multi-objectif (MOABC) est utilisé pour résoudre ce type de problème. Ce qui consiste à essayer d'optimiser la qualité des solutions obtenues et de converger vers les solutions réalisées par cet algorithme.

3.3 Conception du système

3.3.1 Conception Globale

L'objectif de la conception globale est de présenter notre modèle en plusieurs composants plus simples. La figure suivante illustre l'architecture globale proposée pour notre système.

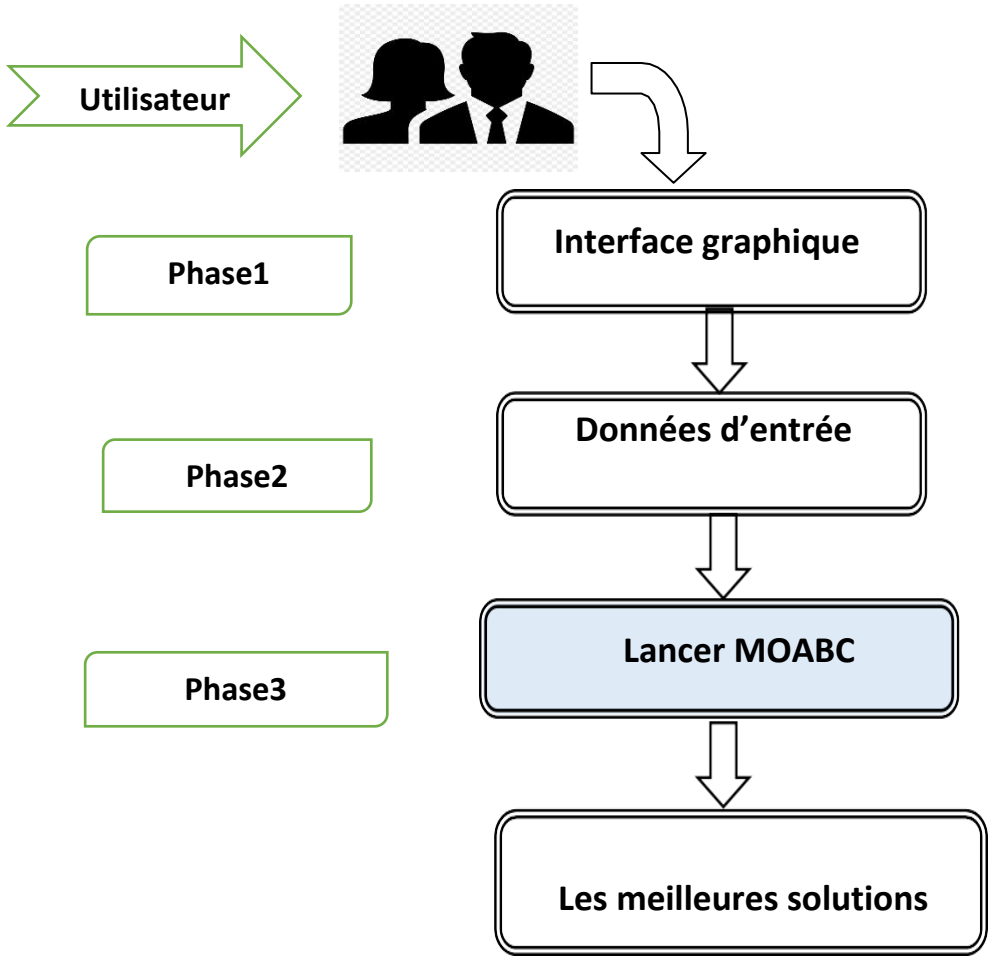


Figure 3.1 : Architecture globale du système.

3.3.1.1 Phase 1

C'est le plus haut niveau de l'application. Il s'agit de l'interface graphique visible par l'utilisateur lors du chargement de l'application ainsi que ses paramètres et ses données.
En d'autres termes, cette phase assure l'interactivité entre l'utilisateur et la machine en lui offrant les moyennes pour manipuler l'exemple à exécuter.

3.3.1.2 Phase 2

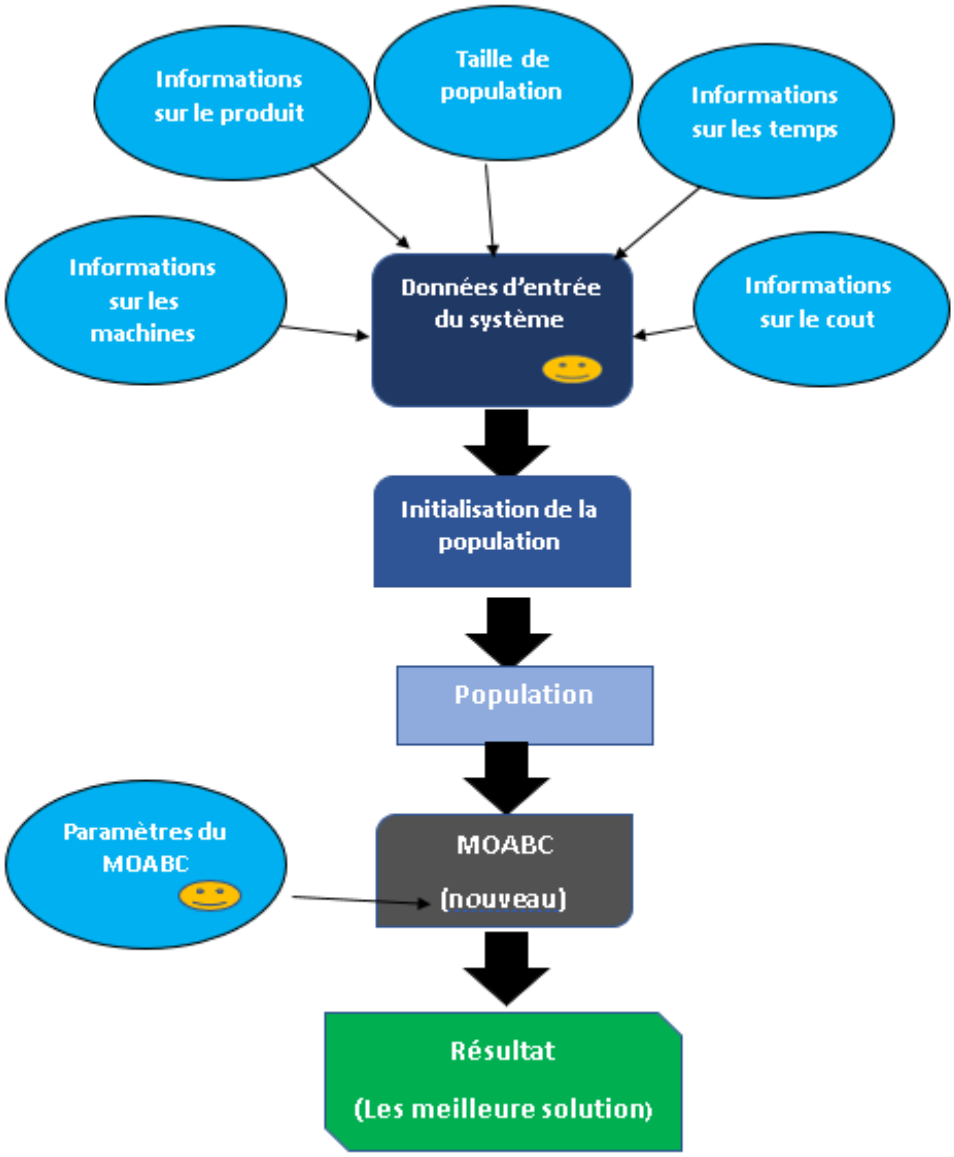
Cette phase faire la relation entre les deux autres phases (les phases 1 et 2) et assure le stockage des données pour qu'elles puissent être utilisées dans la couche noyau. Elle est la liaison entre l'interface et le code source.

3.3.1.3 Phase 3

C'est le module principal du système (le moteur). Il traite les données chargées pour recevoir des résultats. Ce module est l'implémentation du mécanisme de l'algorithme de colonie d'abeilles artificielles multi-objectif (MOABC).

3.3.2 Conception détaillée

L'architecture détaillée du système est proposée pour être utilisée au développement et à la réalisation du système.



: C'est-à-dire, Cette information est entrée par l'utilisateur.

Figure 3.2 : Architecture détaillée du système.

3.3.2.1 Données d'entrée du système

Nous présentons les données d'entrées nécessaires. Ces données regroupées en quatre types (informations sur le produit, informations sur les machines, informations sur le coût et informations sur le temps).

3.3.2.1.1 Information sur le produit

Le produit étant défini comme un ensemble de composants, chaque composant nécessitant un ensemble d'opérations.

- **La séquence d'Operations** : La séquence d'opérations les contraintes de précédence qui exige un ordre précis de succession des opérations.
- **TAD requis :** Pour un produit, un TAD définit la direction avec laquelle une opération doit être effectuée. Un TAD est défini sur le repère tridimensionnel X, Y et Z.
- **Les outils candidats :** Ce genre d'information spécifie les outils pouvant effectuer une opération particulière. Chaque outil à ses propres composants en termes de précision et coût d'utilisation.

3.3.2.1.2 Information sur les machines

Les machines représentent l'environnement duquel le produit va être sorti. Concernent essentiellement leurs configurations possibles et les capacités dont elles disposent dans chacune de leurs configurations, ainsi que les outils nécessaires pour la réalisation de leurs opérations.

3.3.2.1.3 Information sur le temps

L'information sur le temps concerne la durée nécessaire pour les différentes activités pour accomplir toutes les opérations et donnant lieu au produit fini, il compose de quatre principes points :

- Le temps de traitement.
- Le temps de changement de configuration.
- Le temps de changement d'outil.
- Le temps de changement de machine.

3.3.2.1.4 Information sur le coût

L'information sur les coûts se subdivise en trois principes champs :

- Le coût d'utilisation d'une machine/un outil.
- Le coût de changement d'une machine ou d'un outil.
- Le coût de changement de configuration.

3.3.2.2 Structures des données utilisées

3.3.2.2.1 Structure de données du produit

Le produit est représenté par un ensemble des caractéristiques, tel que chaque caractéristique nécessitant un ensemble des opérations. Ainsi que les TADs et les outils nécessaires pour manufacturer ce produit.

- Composants
  - n: Cette valeur représente le nombre total des composants du produit.
  - **NOPk :** Cette valeur représente le nombre d'opérations du composant **k.**
- Opérations
  - **TNOP :** Cette valeur représente le nombre total des opérations nécessaires pour manufacturier un produit.

    Où :

TNOP = $\sum_{k=1}^{n}$ NOPk

- **PS [**1…NPS**] :** Liste représente le graphe de précédence des opérations, où : NPS est le nombre de séquences d'opérations possibles.

TADs

- **OPTAD** [**1...TNOP**] [**1...6**] : Matrice des opérations TADs. Où chaque opération TAD occupe une ligne dans la matrice OPTAD.

Avec :

$$\text{OPTAD } [OP_i^k]\ [d] = \begin{cases} \textbf{1}, \text{ si l'opération } \textbf{i} \text{ de la caractéristique } \textbf{k}, \\ \text{besoin TAD } \textbf{d} \\ \textbf{0}, \textit{Sinon} \end{cases}$$

- **d** index représentant le TAD, d=1 ... 6, il prend l'une des positions des axes de direction

Si d =	1	2	3	4	5	6
TAD dans la direction	X+	Y+	Z+	X-	Y-	Z-

Les outils requis :

- NT : cette valeur représente le nombre d'outils disponibles.
- **OPT [1...TNOP] [1... NT] :** Matrice des outils requis d'opérations. Chaque ligne dans OPT représente l'ensemble des outils pouvant effectuer une opération.

$$\text{OPT } [OP_i^k]\ [t] = \begin{cases} 1, \text{ si l'outil } \textbf{t} \text{ peut effectuer l'opération } \textbf{i} \\ \text{du composant } \boldsymbol{k} \\ 0, \textit{Sinon} \end{cases}$$

3.3.2.2.2 Structure de données des machines

Machines

- **NM** : Nombre de machines disponibles.

Configurations

- **NC [1..NM]** : un tableau contient le nombre de configurations disponible pour chaque machine.

Exemple : NC[2] le nombre de configuration de la machine 2.

- **CTAD [1...NM][1-MAX(NC)][6] :** Matrice des TADs disponibles par configuration de machine.

Outils

- **MT[1-NM][1-NT]** : la matrice des outils pour chaque machine.
- NT : le nombre des outils disponibles.
- **MT[m][t]=** { si la machine m peut utiliser l′outil t ; 0,Sinon

3.3.2.2.3 Structure de données des coûts

Le coût total se compose de cinq coûts suivants :

Coût d'utilisation des machines

- CM [1...NM], représente le coût d'utilisation d'une machine par opération par unité de temps.

Coût de changement de machines

- MCCost[1..NM][1..NM], c'est la matrice des coûts de changement de machines.
- $M[a][b]= C$, donc : **C** est le coût de changement d'une machine **a** par la machine **b** pour effectuer les deux opérations consécutives.

Coût de reconfiguration d'une machine

- CCCost[1..NM][1..MAX(NC)][1..MAX(NC)] : Cette matrice représente le coût de changement d'une configuration dans la même machine.
- CCCost[M][A][B] =Y, le **Y** représente le coût de passage de la configuration **A** à la configuration **B** pour effectuer deux opérations successives sur la même machine **M**.

Coût d'utilisation des outils

- [**1**...***NT***]: Cette matrice représente le coût d'utilisations des outils

Coût de changement des outils

TCCost[1..NT][1..NT]. C'est le coût de passage d'un outil à un autre.

Pour plus de détaille, TCCost[A][B] = Y. Y c'est le coût de passage de l'outil A à l'outil B pour effectuer deux opérations successives sur la même machine.

3.3.2.2.4 Structure de données des temps

Temps de changement de configurations

CCTime [1...NM] [1...MAX (NC)] [1...MAX (NC)], C'est le temps nécessaire pour passer de configuration à une autre sur la même machine.

Temps de changement d'outils

C'est le temps de changement d'outils sur la même machine pour effectuer une Opération.

Temps de traitement

C'est le temps nécessaire pour effectuer une opération particulière sur une machine avec l'une de ses configurations à l'aide d'un outil approprié.

3.4 Fonctions Objectifs

Dans ce projet nous optimisons conjointement deux grands axes : le coût total et le temps total suivant le modèle de [1], [22]. Un seul modèle multi-objectif sera par la suite résolu en se basant sur les techniques de méta-heuristiques.

Nous avons deux objectifs :

1. Le coût total (minimiser).
2. Le temps total (minimiser).

3.4.1 Coût total

Pour calculer le coût total d'un processus de fabrication, il reconnaître de calcule cinq coûts.

- **Coût d'utilisation des machines (MUC)**

C'est le coût d'utilisation des machines pour effectuer toutes les opérations d'une production.

Le MUC est exprimé comme suit :

$$\mathrm{MUC} = \sum_{U=1}^{TNOP} \mathrm{CM[Mj\ (u)]} \times \mathrm{PrTime[\ Mj\ (u)][\ } C_i^j\mathrm{(u)][Tj\ (u)]}$$

Avec :

- $Mj\ (u)$: La machine Mj effectuant l'opération d'index u
- $[Mj\ (u)]$: Coût unitaire de l'utilisation de la machine Mj
- $C_i^j\ (u)$: La configuration de la machine utilisée pour effectuer l'opération d'index u.
- $Tj\ (u)$: L'outil utilisé pour effectuer l'opération d'index u.
- $P[\ Mj\ (u) C_i^j\ (u) Tj\ (u)]$: Temps requis pour achever l'opération.

- **Coût de changement de machine (MCC)**

Il s'agit du coût du passage d'une machine **j** à la machine **j'**, pour effectuer deux opérations successives différentes (**u** et **u'**), où **u** doit être effectué avant **u'**.

Le MCC est exprimé comme suit :

$$\mathrm{MCC} = \sum_{U=1}^{TNOP} \mathrm{MCCOST[Mj\ (u)]\ [Mj\ '(u')]}$$

- **Coût de changement de configuration (CCC)**

C'est le coût nécessaire pour changer les configurations des machines.

Son expression est donnée comme suit :

$$\mathrm{CCC} = \sum_{U=1}^{TNOP} \mathrm{CCCOST[}C_i^j\mathrm{\ (u)]\ [}C_i^j\mathrm{\ (u')]}$$

L'opération **u** doit être effectuée avant **u'** sur la même machine. CCC correspond à la somme du coût de changement de configuration ($CCCi$) de chaque machine dans la ligne de production.

- **Coût d'utilisation des outils (TUC)**

L'expression du coût d'utilisation des outils est donnée comme suit :

$$\text{TUC} = \sum_{U=1}^{TNOP} \text{CT[T(u)]} \times \text{PrTime[Mj (u)][}C_i^j\text{(u)][Tj (u)]}$$

- **Coût d'utilisation des outils (TUC)**

C'est le coût d'utilisation des outils. Son expression est donnée comme suit :

$$\text{TUC} = \sum_{U=1}^{TNOP} \text{CT[T(u)]} \times \text{PrTime[Mj (u)][}C_i^j\text{(u)][Tj (u)]}$$

Avec :

- (u) : L'outil utilisé pour effectuer l'opération d'index u.
- $Mj\ (u)$: La machine Mj effectuant l'opération d'index u
- $C_i^j(u)$: La configuration d'index **i** de la machine d'index **j** effectuant l'opération d'index u.
- $P[\ Mj\ (u)][\ C_i^j\ (u)][Tj\ (u)]$: Temps requis pour achever l'opération u.

- **Coût de changement d'outils (TCC)**

C'est le coût de changer d'outil dans une machine particulière pour effectuer les différentes opérations sur la même machine. Puisque les différentes opérations peuvent exiger des outils de différents types.
Son expression est donnée comme suit :

$$\text{TCC} = \sum_{U=1}^{TNOP} \text{TCCOST[}T_q^j\text{(u)][}T_{q'}^j\text{(u')]}$$

- **Minimiser le coût total (Total Cost)**

Tous les coûts nécessaires sont calculés. Le deuxième objectif relatif au coût est exprimé sous la forme suivante :

Coût total = **MUC** + **MCC** + **CCC** + **TUC** + **TCC**

3.4.2 Temps total

Le temps total est englobé les différents temps nécessaires pour la réalisation de l'ensemble des opérations donnant lieu au produit fini.
Les composants des temps totaux sont les suivantes :

- **Temps de changements des machines (MCT)**

  C'est le temps nécessaire pour le passage d'une machine **j** a la machine **j'**, afin d'effectuer deux opérations successives différentes (u et u').
  Où **u** doit être effectué avant **u'**.

  Son expression est donnée comme suit :

$$\mathrm{MCT} = \sum_{U=1}^{TNOP} \mathrm{MCTime}[\mathrm{Mj}\ (u)]\ [\mathrm{Mj}\ '(u')]$$

- **Le temps de changements d'outils (TCT)**

  C'est le temps de passage d'un outil à un autre outil, il génère si deux opérations successives sur la même machine besoins deux outils.
  Son expression est donnée comme suit :

$$\mathrm{TCT} = \sum_{u=1}^{\mathrm{TNOP}} \mathrm{TCTime}[\ T_q^j(u)][T_{q'}^j(u')]$$

- **Le temps de changement des configurations (CCT)**

  C'est le temps de passage d'une configuration à une autre sur la même machine. Il dépend de la machine et les 2 configurations.

Son expression est donnée comme suit

$$\mathrm{CCT} = \sum_{U=1}^{TNOP} \mathrm{CCTimr}[\ C_i^j(u)]\ [C_i^j(u')]$$

- **Temps de traitement (PT)**

  C'est le temps nécessaire pour effectuer une opération particulière sur une machine. Il dépend de la machine, la configuration adoptée ainsi que l'outil d'usinage utilisé.

  Son expression est donnée comme suit :

$$PT = \mathrm{PrTime}[\ \mathrm{Mj}\ (u)][\ C_i^j\ (u)][\mathrm{Tj}\ (u)]$$

- **Minimiser le temps total (Total Time)**

Le premier objectif est égal à la somme des objectifs ci-dessus.

$$\textbf{Temps total} = \textbf{CCT} + \textbf{T CT} + \textbf{MCT} + \textbf{P T}$$

3.5 Représentation de l'individu (solution ou ligne de production)

Une ligne de production affecte à chaque opération une machine capable de l'exécuter. Pour permettre la manipulation et l'exécution des lignes de production on doit fournir la séquence d'opérations avec la précision de quel composant. Pour manipuler les processus de fabrication qui sont en représentation textuelle, il est nécessaire de passer à un processus de fabrication sous forme matricielle comme proposée dans des travaux précédents [5] [22], où cette matrice est de taille M x N, où M est égale à 3 (Machines, Outils, Configurations), et N dépend du nombre total des opérations.
Cette matrice est interprétée, ci-dessous, de gauche à droite colonne par colonne.

Machine	M2	M1	M1	M2	M2	M3	M1
Configuration	C3	C2	C1	C1	C1	C1	C2
Outil	T2	T1	T3	T2	T3	T1	T4

Tableau 3.1 : Processus de fabrication sous forme d'une matrice.

Dans cet exemple le produit est caractérisé par trois caractéristiques, Où F1 besoin trois opérations pour le finir, F2 besoin deux opérations pour le finir et F3 besoin deux opérations pour le finir. La première colonne doit être lue comme suit : l'opération O1 du composant F1 est effectuée sur la machine M2, avec la configuration C3, en utilisant l'outil T2. Ainsi les colonnes suivantes sont interprétées de la même manière.

3.6 Codages et décodages

3.6.1 Codages de l'individu

Le codage des solutions est une étape clé dans l'implémentation des métaheuristiques, il consiste à passer de la représentation réelle des solutions vers une représentation codée. Le but du codage des solutions est de rendre les solutions plus maniables par l'algorithme de recherche.

Une solution (individu ou chromosome) générée par l'algorithme représente le processus de fabrication qui est considérée comme une matrice **M** × **N** de nombres réels compris entre **0** et **1**, comme le propose [22] où **M** est égale à **3** (Machines, Configuration, Outils), et **N** dépend du nombre total des opérations.
Exemple : une matrice de 3 × 6. Cette solution a 6 opérations pour l'accomplir.

0.35	0.56	0.47	0.12	0.21	0.6
0.05	0.30	0.78	0.92	0.84	0.44
0.70	0.39	0.60	0.95	0.06	0.10

Tableau 3.2 : ligne de production codé en nombre réel.

Une fois les résultats obtenus, il faut passer à l'étape du décodage des solutions codées. Afin d'aboutir à cela, on recourt à des procédures de décodage c'est-à-dire pour : les machines, les configurations et les outils.

3.6.2 Décodages de l'individu

3.6.2.1 Décodage des machines

Le décodage des machines est la troisième étape du décodage d'une solution. Ce décodage doit être effectué après celui des composants et des opérations. La procédure est résumée comme suit :

1. Identifier parmi les machines reconfigurables, celles qui possèdent les $\boldsymbol{TAD}$ et les outils requis pour effectuer l'opération de la même colonne.

2. Créer une liste de la taille n des machines reconfigurables identifiées. n est le nombre des machines reconfigurables pouvant réaliser l'opération de la même colonne.

Par hypothèse, les machines ayant le petit index sont placées dans la liste avant les autres. Par exemple $M1$ avant $M2$ avant $M3$...)

3. Multiplier n par le nombre réel se trouvant dans la solution codée, dans la cellule correspondante à la cellule à décoder. Cette étape renvoie un nombre réel p ($0<p\leq n$)

4. Arrondir le nombre réel obtenu p au nombre entier immédiatement supérieur, où $1 \leq p \leq n$.

5. La machine de la position p dans la liste des machines candidates est placée dans la cellule en cours de décodage.

Ce processus est ensuite répété jusqu'à ce que tous les éléments de la troisième ligne soient décodés.

3.6.2.2 Décodage des configurations

Le décodage des configurations est la quatrième étape du décodage d'une solution. Ce décodage est effectué après celui des machines. La procédure est résumée comme suit :

1. Identifier, parmi les machines déjà décodées par la procédure du décodage des machines, les configurations pouvant effectuer l'opération concernée en termes de $TADs$ et d'outil.

2. Ces configurations sont classées dans une liste de taille n. Où n est le nombre de configurations identifiées.

Par exemple, si parmi 7 configurations de la machine, les configurations $C1,2\ et\ C5$ sont capables d'effectuer l'opération concernée, alors la liste comprend $C1 – C2 – C5$ en tenant compte que n =3. Par hypothèse, les configurations ayant le petit index sont placées dans la liste avant les autres. Par exemple ($C2$ avant $C3$).

3. Multiplier n par le nombre réel se trouvant dans la solution codée, dans la cellule correspondante à la cellule à décoder. Cette étape renvoie un nombre réel p ($0<p\leq n$).

4. Arrondir le nombre réel obtenu p au nombre entier immédiatement supérieur, où $1 \leq p \leq n$.

5. La configuration de la position p dans la liste des configurations candidates est placée dans la cellule en cours de décodage. Ce processus est ensuite répété jusqu'à ce que tous les éléments de la quatrième ligne soient décodés.

3.6.2.3 Décodage des outils

Le décodage des outils est la dernière étape du décodage d'une solution. Après cette étape, on peut obtenir un individu (une solution) qui pourrait être évalué.

Le décodage des outils est réalisé à travers les étapes suivantes :

1. Identifier parmi les outils :
   - Ceux qui sont disponible pour la machine concernée.
   - Ceux qui ont la capacité d'effectuer l'opération à réaliser.

2. Ces outils sont classés dans une liste de taille n. Où n est le nombre des outils identifiés.

   Par hypothèse, les outils ayant le petit index sont placés, dans la liste, avant les autres. Par exemple (T1 avant T2).

3. Multiplier n par le nombre réel se trouvant dans la solution codée, dans la cellule correspondante à la cellule à décoder. Cette étape renvoie un nombre réel

p ($0<p\leq n$)

4. Arrondir le nombre réel obtenu p au nombre entier immédiatement supérieur, où $1 \leq p \leq n$.

5. L'outil de la position p dans la liste des outils candidates est placé dans la cellule en cours de décodage.

Ce processus est ensuite répété jusqu'à ce que tous les éléments de la cinquième ligne soient décodés

3.7 Algorithme colonie des abeilles artificielles multi objectif adapté

3.7.1 Fonctionnement

L'algorithme colonie des abeilles artificielles multi-objectif « Multi-Objective Artificial Bees Colony (MOABC) » a été conçu pour résoudre une multitude de problèmes multi-objectifs. L'un de ces problèmes est le problème de la sélection des machines reconfigurables pour trouver le meilleur arrangement des machines (RMTs) et générer des processus de productions optimales dans l'environnement des systèmes manufacturiers reconfigurables. Afin d'atteindre cet objectif, on a adapté le **MOABC**.

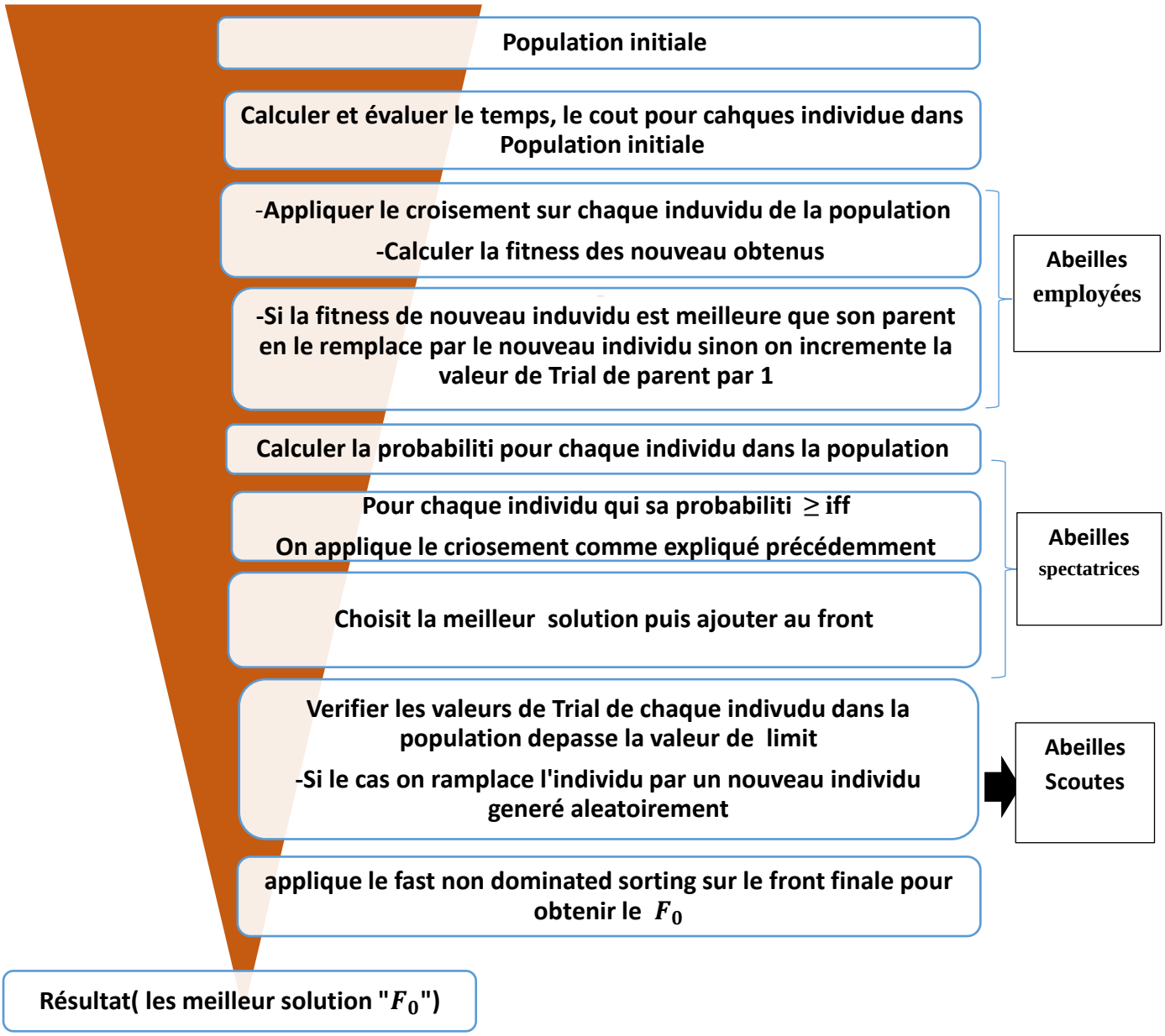


Figure 3.3 : Algorithme MOABC proposé.

Dans les sections suivantes on va détailler beaucoup plus les étapes de l'algorithme MOABC.

3.7.2 Génération de la population initiale

La population initiale est générée de façon aléatoire selon le nombre d'individus définis supposant N (N processus de fabrication codés). Ces individus codés doivent être décodés pour le même individu pour permettre l'évaluation. C'est à dire pour calculer le temps total (f1) et le coût total (f2).

3.7.3 Évaluation de la population

Pour calculer les fonctions fitness (f1 et f2), dans notre cas la minimisation du temps total de production (temps total) et la minimisation le coût total de production (Coût total), nous exige de suivre les différentes phases de décodages citées ultérieurement et qui respecte les contraintes d'affectation des opérations aux machines. Ces valeurs vont déterminer la qualité des solutions obtenues et donneront la meilleure solution existe

3.7.4 Fonction de tri rapide non dominée (Fast Non-dominated Sorting)

Cette technique renvoie une population classée (F) avec plusieurs fronts $(\mathrm{F1}, \mathrm{F2}, \ldots\ldots)$ en fonction de deux objectifs, minimiser le coût et minimiser le temps.

Donc, pour établir cette population classée c'est à dire quelles solutions feront parties de chaque front de Pareto, il faut réaliser un classement des résultats obtenus par chaque séquence de chromosomes générée. Dans la première étape, nous calculons deux entités : 1) Compteur de domination Np, qui représente le nombre de solutions qui dominent la solution p et 2) Sp qui est l'ensemble de solutions que la solution p domine. Toutes les solutions qui font parties du premier front non-dominé doivent avoir leur compteur de domination égal à zéro. Partant de là, pour chaque solution p avec $Np = 0$, nous visitons chaque membre (q) dans son ensemble Sp et nous réduisons le compteur de domination. Pendant cette réduction, si un membre q du compteur de domination devient zéro, nous l'ajoutons dans une liste Q. Ces membres appartiennent au deuxième front non dominé. Cette procédure décrite est faite avec chaque membre de Q et le troisième front est identifié suivant la même approche. Cette démarche continue jusqu'à l'identification de tous les fronts. Pour chaque solution p dans le deuxième ou le plus haut niveau de non-domination, le compteur de domination Np peut être égal au moins à $N - 1$ ($N = taille\ de\ la\ population$). Alors chaque solution p sera visitée au moins $N - 1$ fois avant que son compteur de domination soit égal à zéro. A ce moment-là, la solution est affectée à un niveau de non-domination et elle ne sera pas de nouveau visitée.[41]

3.7.5 La sélection

- Le rôle principal de l'opérateur de sélection dans une méthode d'optimisation est d'imposer une direction pour le processus de recherche. La direction de la recherche devrait être en accord avec les préférences du décideur.

- La sélection a donc pour objectif d'identifier les individus qui doivent se reproduire et par conséquent elle permet de décider sur la survie ou la disparition d'un individu.

- Dans notre application, cette sélection est pour la phase de l'abeilles spectatrice, cette sélection se fait à travers les étapes suivantes :

  - On choisit iff aléatoirement et iff $=]-1,1[$.
  - Calculer la probabilité pour chaque individu.
  - L'individu qui est une probabilité$\geq$ iff choisit par l'abeille employée pour applique le croisement.

3.7.6 Le croisement

L'opérateur du croisement produit un enfants en prenant deux parents et un point de croisement comme paramètre. Chaque parent est coupé en deux fragments au même point (point de croisement), puis l'enfant prend le fragment avant le point de croisement d'un parent, et le fragment après le point de croisement de l'autre parent. De cette manière la qualité d'enfants (les fonctions d'objectifs) sera déterminée en grande partie de la qualité des parents.

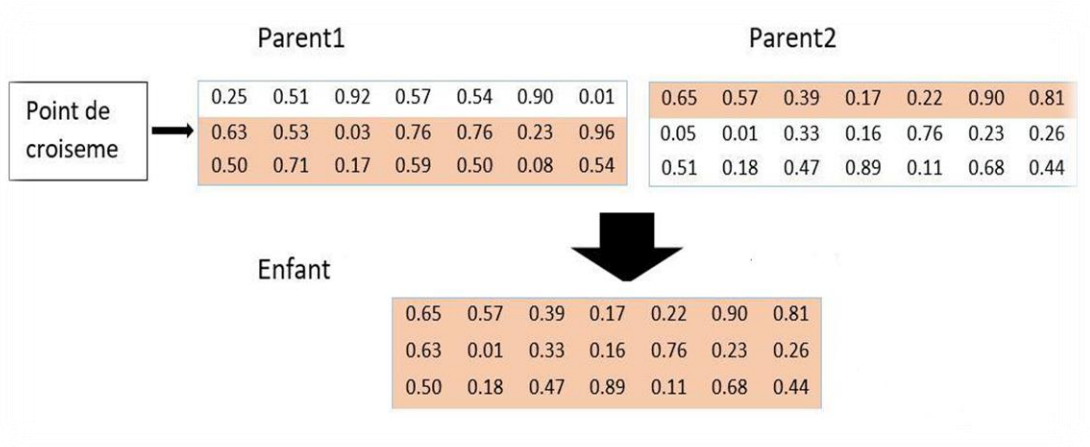


Figure 3.4: Croisement à un point.

3.8 Conclusion

Dans ce chapitre nous avons vu la conception de notre projet, il est en deux étapes (conceptions globale et la conception détaillée). Nos principaux objectifs nous a facilité l'identification et la présentation du problème de génération lié à la ligne de production dans le cadre des systèmes reconfigurables.

chapitre 4 :

Impléméntation du Système

4.1 Introduction

Après avoir la conception générale de notre projet dans le chapitre précède, nous devons passer à l'étape suivante du projet qui est l'implémentation.

Dans ce chapitre final, nous allons présenter les différents outils de développement ainsi que les langages appris et exploités dans la réalisation de notre projet. Nous exécutons un exemple illustratif sur notre application pour l'algorithme traité ultérieurement (MOABC), et nous présentons et discutons quelques résultats expérimentaux.

4.2 Langage de programmation et outils de développement

Dans la réalisation de notre projet et sur les deux côtés (programmation et théorique) nous avons utilisé des outils et langages qui nous aident.

4.2.1 Langage de programmation Python

Python est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété, Python est un langage idéal pour l'écriture de scripts et le développement rapide d'applications dans de nombreux domaines et sur la plupart des plateformes [43].

Figure 4.1: Logo de python.

4.2.2 L'environnement de programmation Pycharme

Pycharm est un environnement de développement intégré (IDE) utilisé pour la programmation en Python. Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration du logiciel de gestion des versions, et supporte le développement web avec Django. Pycharm est développé par la société tchèque JetBrains. Il fonctionne sur plusieurs plates-formes Windows, Mac OS X et Linux.

Figure 4.2: Logo de PyCharm.

4.2.3 Le Package (Tkinter) Tool kit Interface

Le paquet tkinter (interface Tk) est l'interface Python standard de la boîte à outils d'IUG Tk. Tk et tkinter sont disponibles sur la plupart des plates-formes Unix, ainsi que sur les systèmes Windows. Dans notre application nous avons utilisons ce package pour développer nos interfaces graphiques, parce qu'elle est simple à utiliser et donne des bons résultats.

Figure 4.3: Logo de TKinter.

4.3 Présentation des interfaces de notre système

Notre système contient un ensemble des interfaces qui faciliter à l'utilisateur de saisie les données.

4.3.1 Interface principale

La figure suivante représente l'interface de notre application finale.

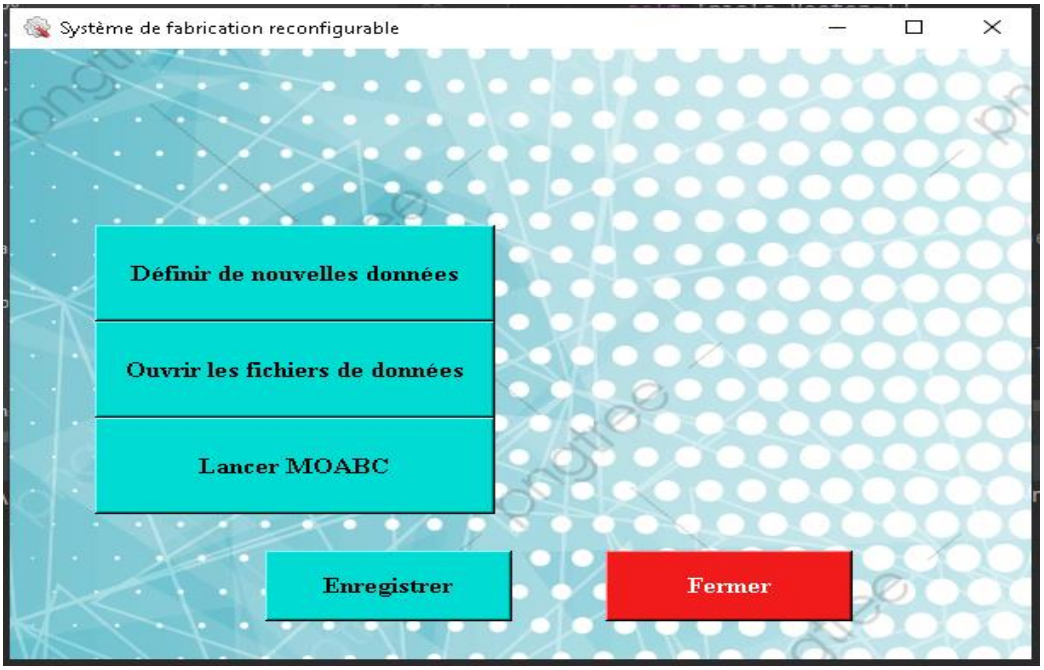


Figure 4.4: Interface principale.

Notre Interface principale offre de nombreuses fonctionnalités à l'utilisateur :

- Créer un nouveau problème (informations sur les machines et le produit).
- Sauvegarder les différentes données saisies au format CSV qui sont des fichiers lisibles tout en offrant la possibilité de les consulter.
- Sans oublier la possibilité de lancement de l'algorithme MOABC.

4.3.1.1 Bouton « Définir de nouvelles données »

Avec la commande « **Définir de nouvelles donnée »** l'utilisateur peut créer de nouvelles machines reconfigurables et pour cette raison un produit en sélectionnant leurs configurations conformément à la formulation expliquée dans le chapitre 3.

Les données d'entrée sont saisies dans la fenêtre illustrée par la figure suivante :

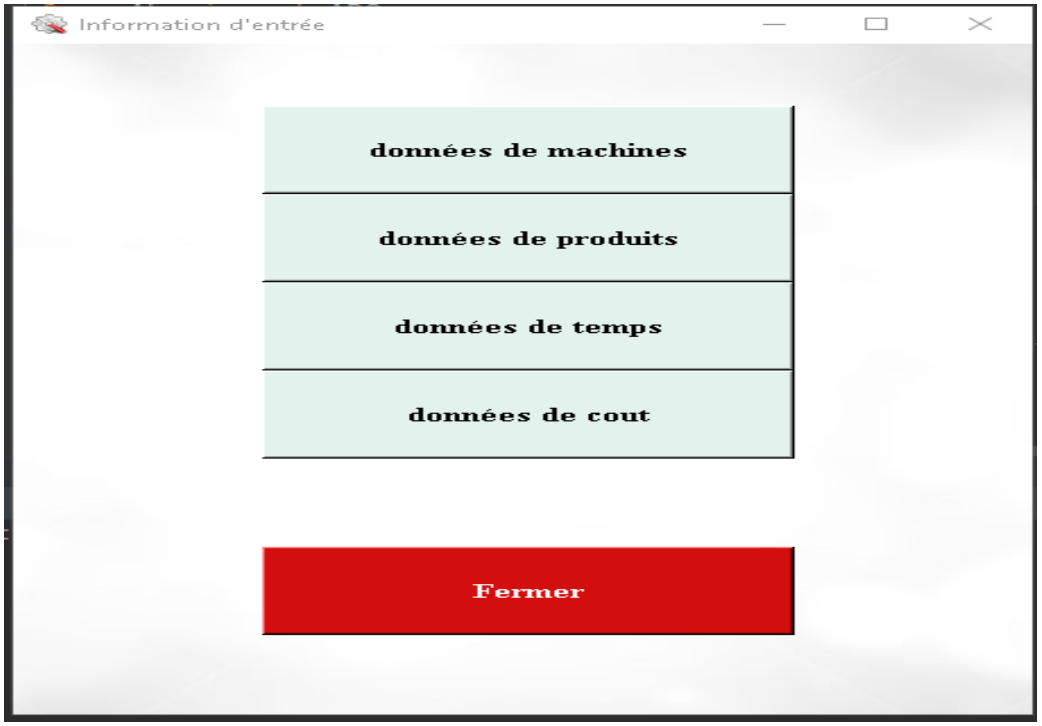


Figure 4.5: Interface des données d'entrée.

A partir de cette fenêtre l'utilisateur doit saisir les « **données de machines** », puis les «**données de produits**», après, l'utilisateur peut saisir des «**données de temps**» et des «**données de coût**».

➢ Interfaces des données de machines

La première étape des données d'entrée consiste à saisir les données de machines. En cliquant sur le bouton «**données de machines**», une nouvelle fenêtre s'affiche pour entrer les données des machines.

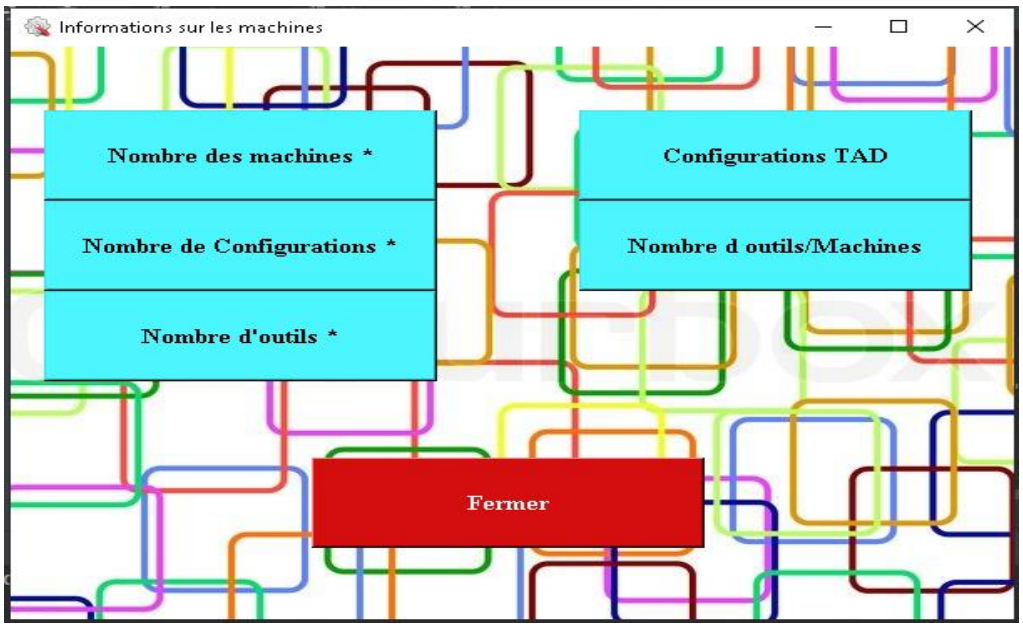


Figure 4.6: Interface des données de machines.

A partir de cette fenêtre l'utilisateur doit identifier le nombre de machines, configurations, outils, outils/machines, outils disponibles et les valeurs des six axes de mouvement pour chaque configuration d'une machine.

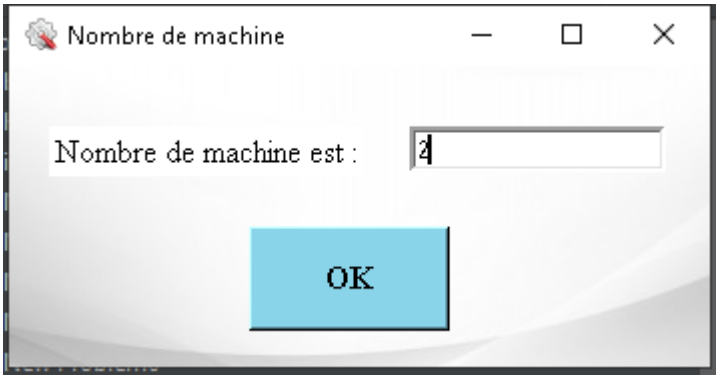


Figure 4.7: Interface nombre de machines

Pour la configuration TAD « CTAD » l'utilisateur doit identifier les axes « X, Y, Z » de chaque configuration d'une machine. Voir la figure suivante.

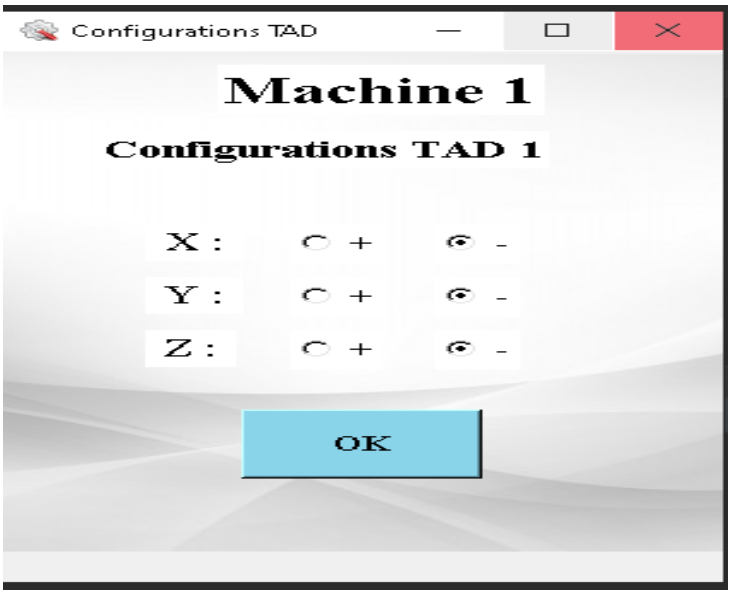


Figure 4.8: Interface de la configuration TAD.

L'interface Nombre d'outil / Machine permet à l'utilisateur de sélectionner l'ensemble d'outils disponibles de chaque machine.

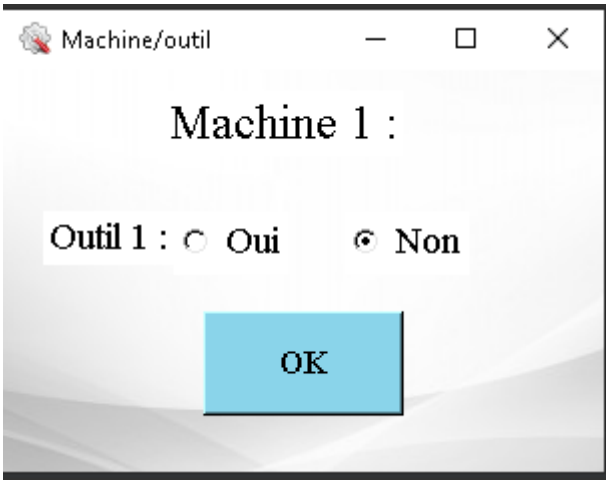


Figure 4.9: Interface machine/outil.

➢ Interfaces des données du produit

La deuxième étape de la saisie des données d'entrée consiste à entrer les données du produit. En cliquant sur le bouton « **données du produit** » une nouvelle fenêtre s'affiche permettant la sélection des choix appropriés au produit défini.

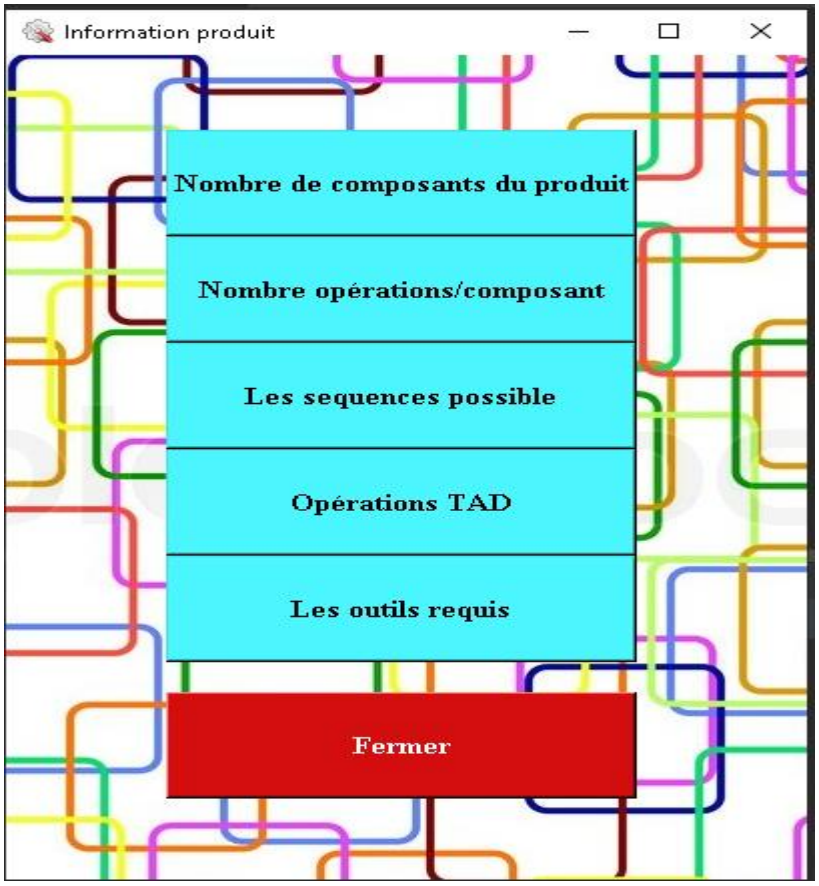


Figure 4.10: Interface des données de produit

A partir de cette fenêtre l'utilisateur doit identifier le nombre des composants d'un produit, le nombre d'opérations de chaque composant, les séquences d'opérations possibles et les opérations TAD.

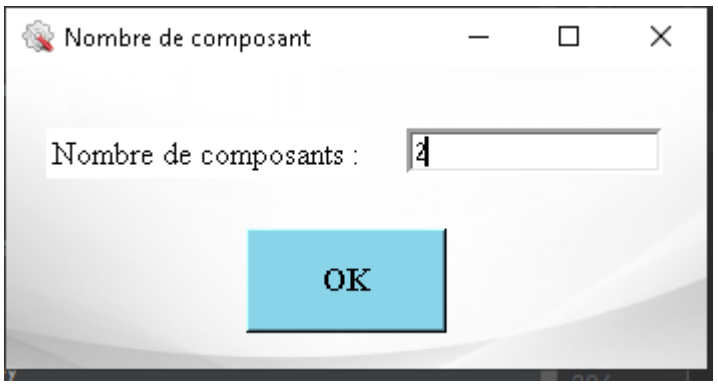


Figure 4.11: Interface pour définir le nombre de composants d'un produit

A partir du bouton «**Les séquences possibles**», l'utilisateur doit saisir les séquences possibles pour les opérations d'un composant se précise.

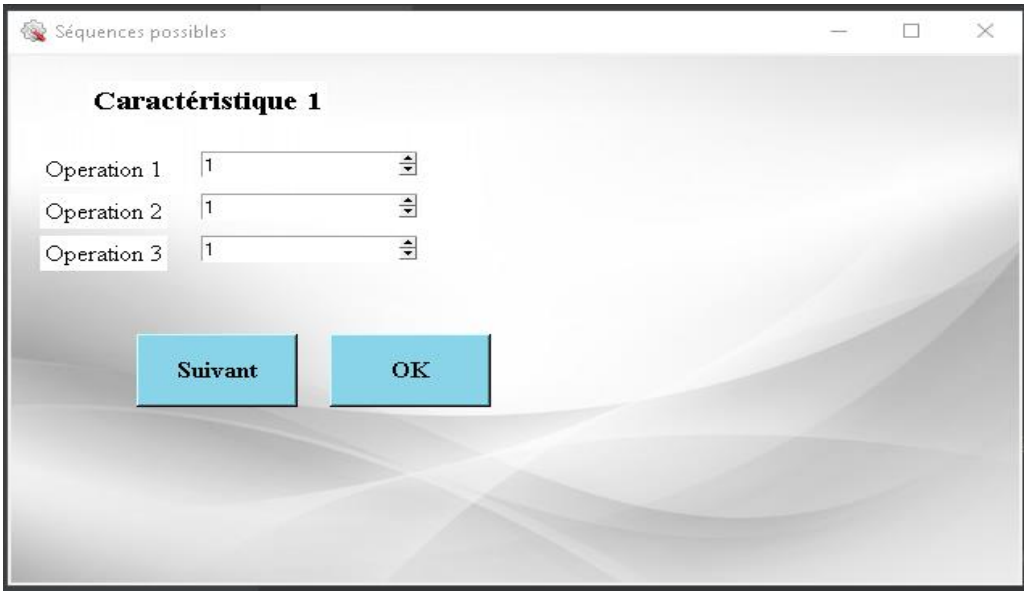


Figure 4.12: Interface des séquences possibles.

Ainsi le bouton «**Opération TAD**», offre à l'utilisateur la manipulation des opérations TAD.

Figure 4.13: Interface d'une operation TAD.

➢ Interfaces des données de temps

La troisième étape permet à l'utilisateur de saisir toutes les données de temps grâce au bouton **«données de temps»**.

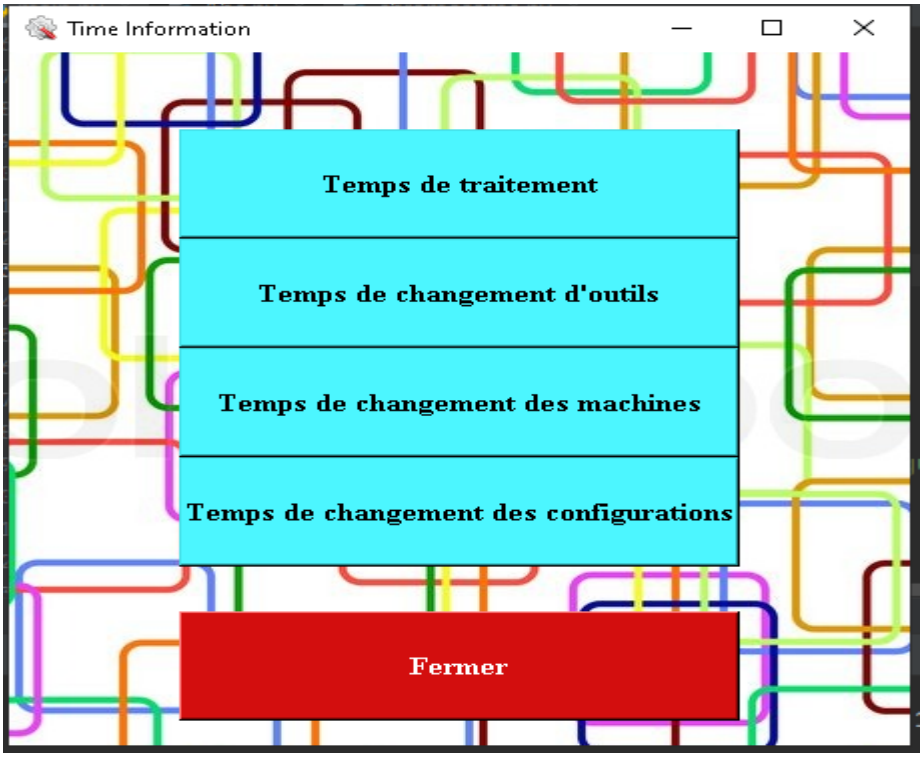


Figure 4.14: Interface des données de temps.

- **Temps de changement d'outils**

Le temps total de changement d'outils correspond à l'ensemble des changements d'outils nécessaires à la réalisation d'un produit.

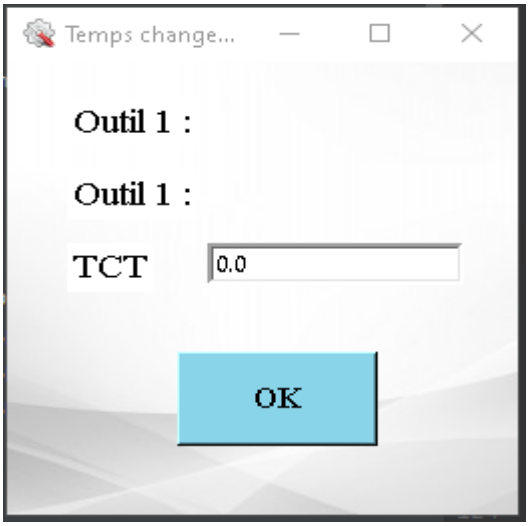


Figure 4.15: Interface de temps de changement d'outils.

- **Temps de changement des machines**

Le temps total de transfert représente comme la durée nécessaire pour transporter le produit d'une machine vers une autre jusqu'à l'obtention du produit fini.

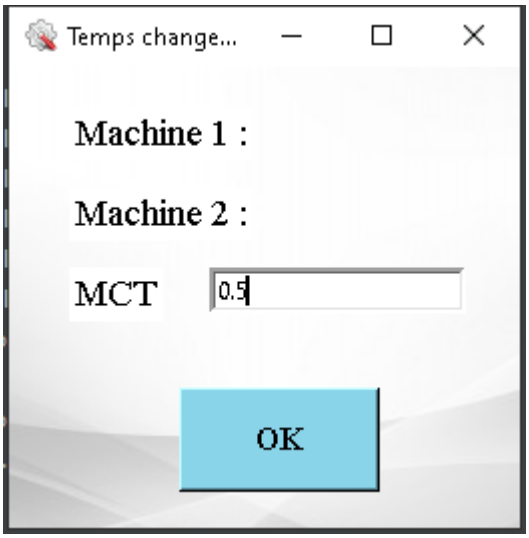


Figure 4.16: Interface de temps de changement des machines.

➢ **Temps de changement de configuration**

Le temps total de changement de configuration désigne la somme de toutes les durées consommées lors des changements individuels des configurations des machines pour réaliser le produit défini.

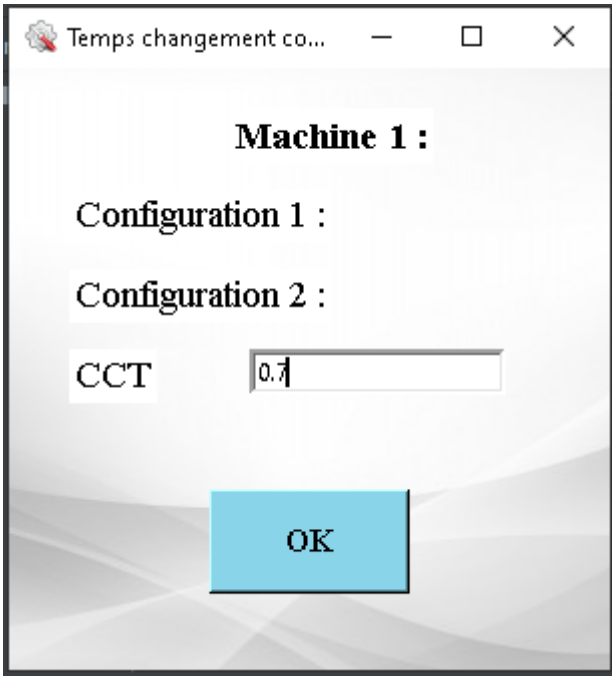


Figure 4.17: Interface de temps de changement de configuration.

➢ Interfaces des données de coût

La quatrième étape permet à l'utilisateur de saisir toutes les données de coût grâce au bouton de « **données de coût** ».

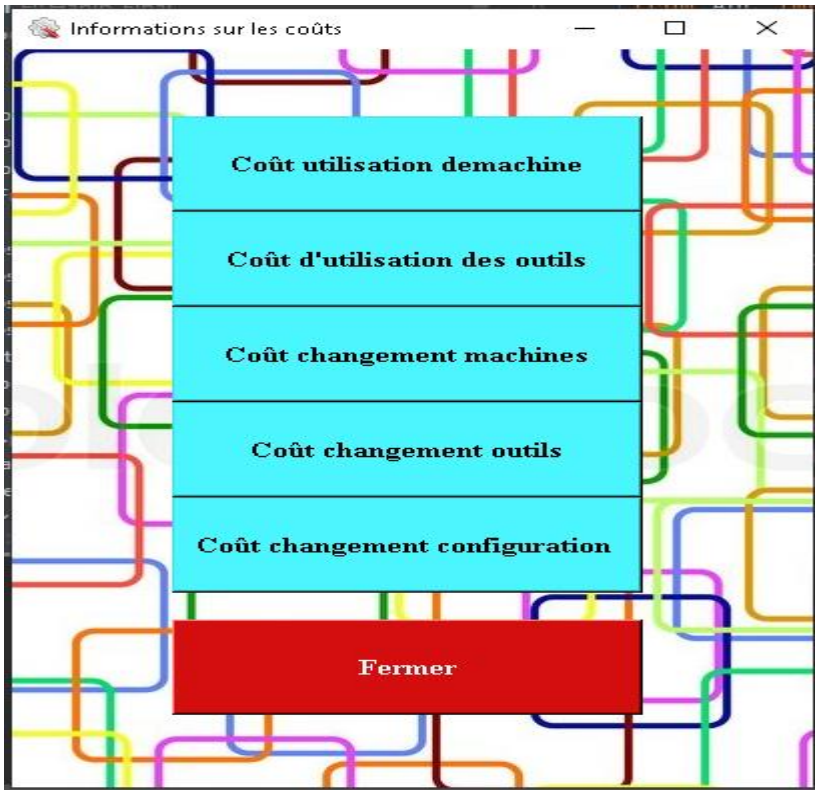


Figure 4.18: Interface des données de coût.

- **Coût d'utilisation de la machine**

La fenêtre permet à l'utilisateur de saisir le coût d'utilisation de chaque machine.

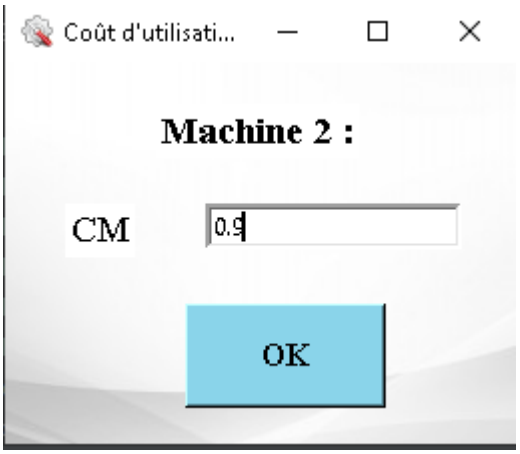


Figure 4.19: Interface de coût d'utilisation d'une machine.

- **Coût d'utilisation des outils**

La fenêtre (figure 4.20) permet à l'utilisateur de saisir le coût d'utilisation de chaque outil.

Figure 4.20: Interface de coût de d'utilisation d'outils.

- **Coût de changement des machines**

Le coût total de changement de machines représente l'ensemble des coûts nécessaires pour transporter le produit d'une machine vers une autre.

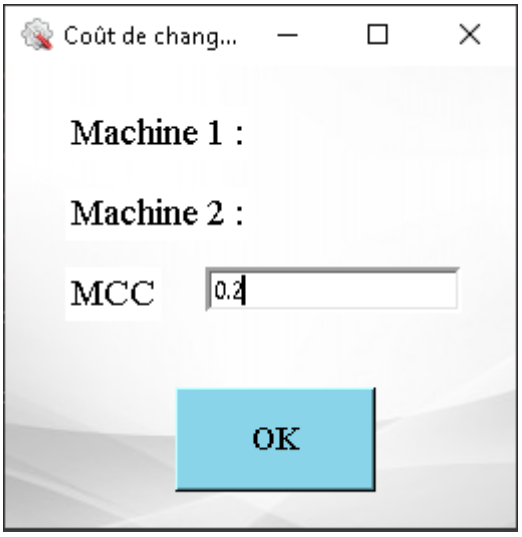


Figure 4.21: Interface de coût de changement des machines.

- **Coût de changements des outils**

Le coût total de changement d'outils correspond à la somme des coûts de changements d'outils nécessaires à la réalisation du produit.

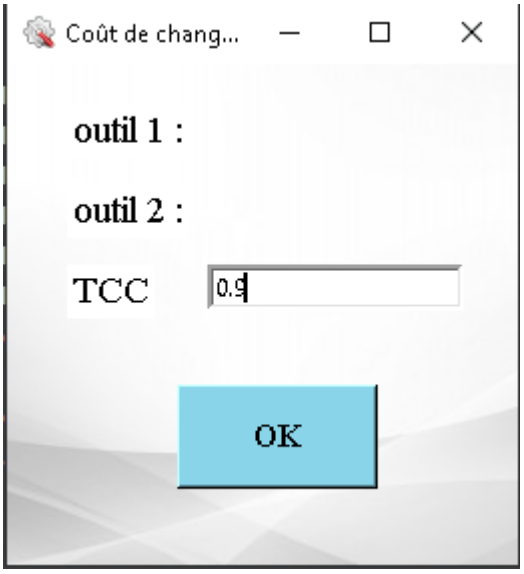


Figure 4.22: Interface de coût de changement d'outils

- **Coût de reconfiguration d'une machine**

Le coût total de changements de configurations désigne l'ensemble des coûts générés par les actions de reconfiguration des machines.

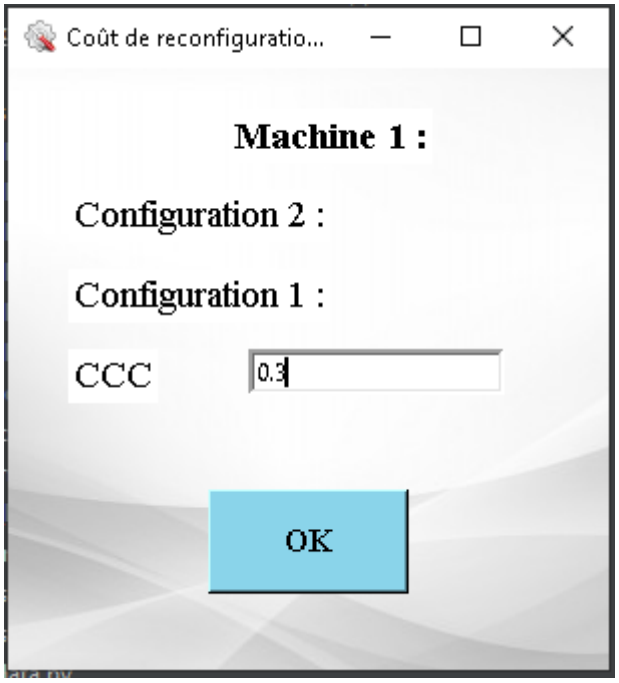


Figure 4.23: Interface de coût de reconfiguration d'une machine.

4.3.1.2 Bouton «Ouvrir les fichiers des données»

La commande «Ouvrir les fichiers des données» permet à l'utilisateur de consulter les fichiers où les données sont stockées.

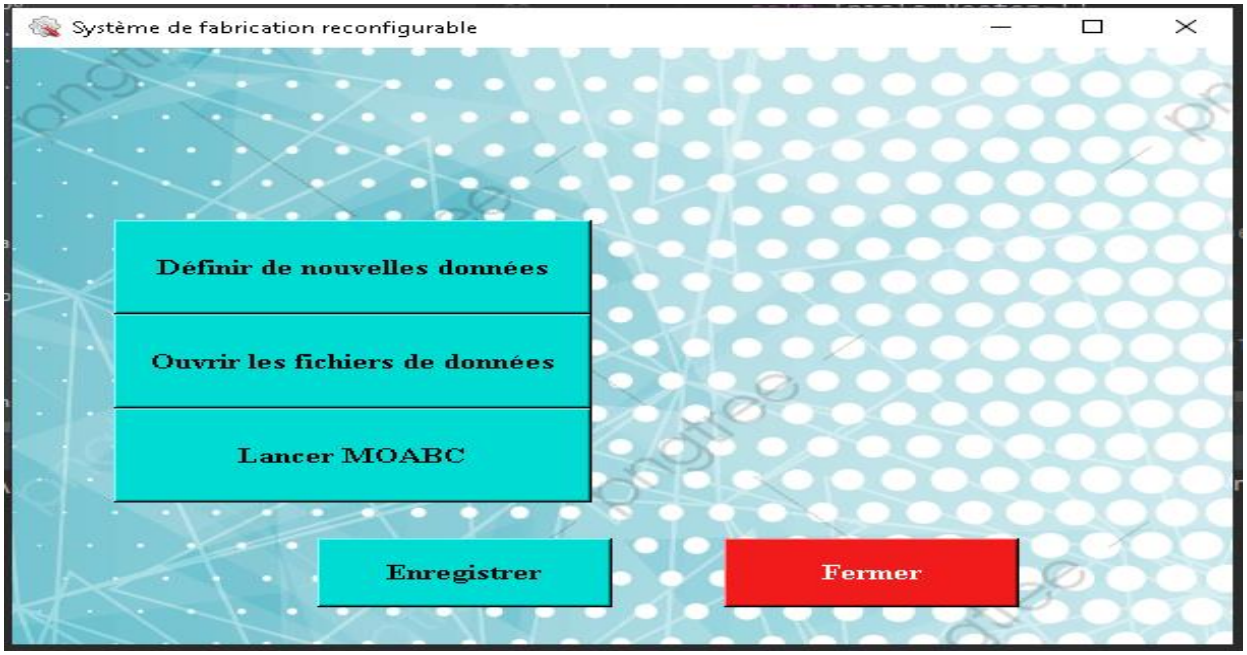


Figure 4.24: Interface du gestionnaire des données.

4.3.1.3 Bouton «Lancer MOABC»

La commande «**Lancer MOABC**» permet à l'utilisateur de saisir les parametre de MOABC.

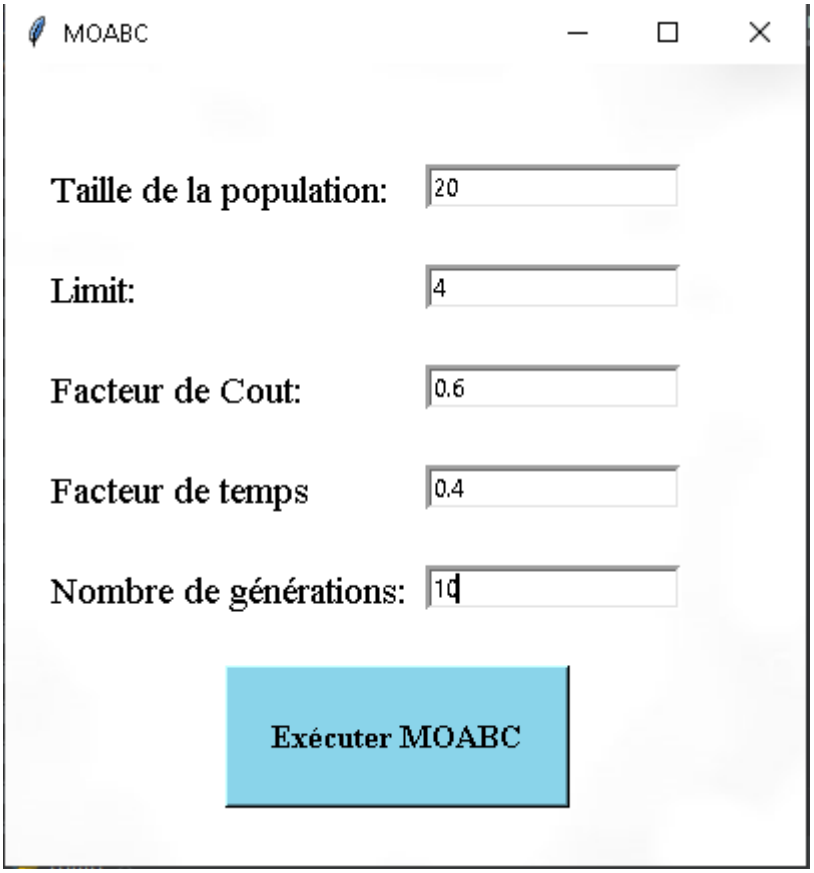


Figure 4.25: Interface de Lancer MOABC.

- **Bouton « Exécuter MOABC»**

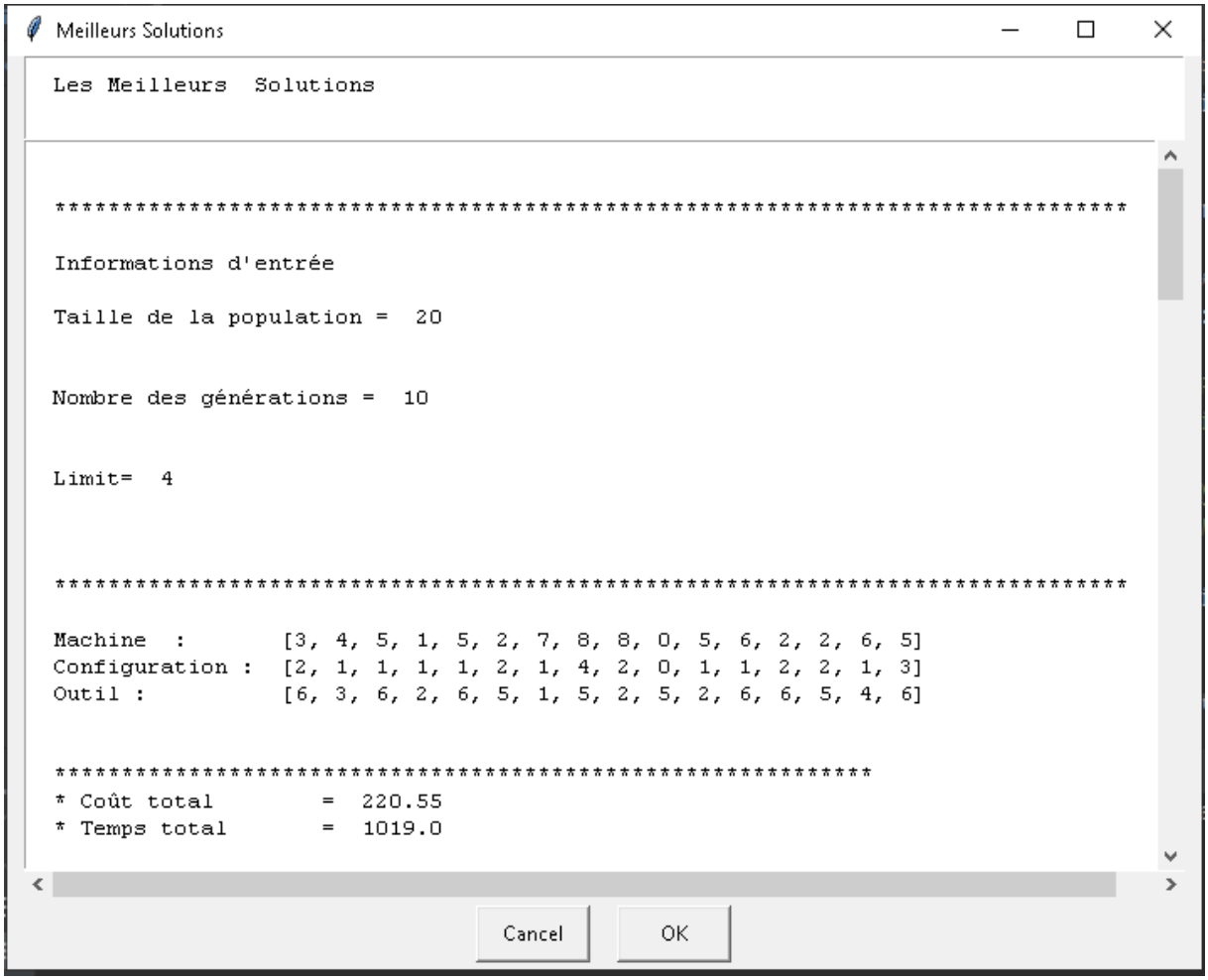


Figure 4.26: Interface résultat de MOABC.

4.4 Evaluation des résultats

Dans notre cas d'étude, nous prenons un exemple illustratif pour tester le fonctionnement de notre application et évaluer les performances de l'algorithme implémentés.

4.4.1 Etude de cas

Les données d'entrée de machines se composent de huit machines (Mch1, Mch2, Mch3, Mch4, Mch5, Mch6, Mch7, Mch8) qui partagent un ensemble de six outils (T1, T2, T3, T4, T5, T6). Le tableau 4.1, ci-dessous illustre les données d'entrée des machines qui incluent les configurations TAD et l'ensemble des outils disponibles pour chaque machine

Machines	Configurations	TAD						Outils requis
		X+	X-	Y+	Y-	Z+	Z-	
Mch 1	**Cf1**	1	0	0	1	0	1	**T1, T2, T3, T4, T5**
	Cf2	1	0	1	0	1	0	
	Cf3	0	1	0	1	1	0	
Mch 2	**Cf1**	0	1	1	0	1	0	**T3, T5, T6**
	Cf2	0	1	0	1	1	0	
	Cf3	1	0	1	0	1	0	
	Cf4	1	0	0	1	0	1	
Mch 3	**Cf1**	1	0	0	1	1	0	**T1, T4, T6**
	Cf2	0	1	1	0	0	1	
Mch 4	**Cf1**	1	0	0	1	0	1	**T1, T3**
	Cf2	0	1	1	0	0	1	
Mch 5	**Cf1**	1	0	0	1	1	0	**T1, T3, T4, T5, T6**
	Cf2	0	1	0	1	1	0	
	Cf3	0	1	0	1	1	0	
Mch 6	**Cf1**	0	1	1	0	1	0	**T2, T5, T1, T4, T6**
	Cf2	1	0	1	0	1	0	
	Cf3	0	1	1	0	0	1	
Mch 7	**Cf1**	1	0	0	1	0	1	**T1, T6**
	Cf2	0	1	1	0	1	0	
Mch 8	**Cf1**	0	1	1	0	1	0	**T2, T4, T5**
	Cf2	1	0	1	0	1	0	
	Cf3	1	0	0	1	1	0	
	Cf4	1	0	1	0	0	1	

Tableau 4.1: Configurations TAD et outils disponibles.

Dans cet exemple, le produit est composé de quatre composants, le premier composant a nécessité trois opérations à effectuer. Le deuxième et quatrième composants ont nécessité quatre opérations à effectuer. Enfin, le troisième a eu besoin de cinq opérations à effectuer. Donc, comme total on a 16 opérations à réaliser.

Et voici la séquence d'opérations et illustre dans ce tableau :

Composant	1	4	2	3	2	3	4	1	3	1	2	4	3	4	2	3
Operation	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	5

Tableau 4.2: séquence d'opérations à réaliser.

Le tableau suivant indique les données d'entrée du produit, à savoir les composants et leurs opérations TAD, ainsi que l'outil requis pour chaque opération.

Composants	Opérations	TAD						Outils requis
		X+	X-	Y+	Y-	Z+	Z-	
Composant 1	**Opr1**	1	0	0	1	0	1	**T1**
	Opr2	0	1	1	0	1	0	**T5**
	Opr3	1	0	1	0	1	0	**T2**
Composant 2	**Opr1**	1	0	0	1	1	0	**T4**
	Opr2	0	1	0	1	0	1	**T6**
	Opr3	0	1	0	1	1	0	**T1**
	Opr4	1	0	1	0	0	1	**T5**
Composant 3	**Opr1**	0	1	1	0	0	1	**T1**
	Opr2	0	1	0	1	1	0	**T5**
	Opr3	1	0	0	0	0	1	**T1**
	Opr4	0	1	0	1	1	0	**T5**
	Opr5	0	0	1	1	0	0	**T2**
Composant 4	**Opr1**	1	0	0	1	0	1	**T3**
	Opr2	1	0	0	1	1	0	**T4**
	Opr3	0	1	0	1	1	0	**T1**
	Opr4	0	1	1	0	1	0	**T6**

Tableau 4.3: Opérations TAD et outils requis

Comme on a vu dans le chapitre 3, les informations du temps se composent de 4 entités (temps de traitement, temps de changement de machines, temps de changement d'outils, temps de changement de configuration).

Nous présentons les données de chacune d'entre elles dans les tableaux suivants :

- ## Le temps de traitement

Pour savoir le temps de traitement d'une opération sur une machine, on est obligé de prédéfinir le type d'opération, type de configuration et l'outil utilisé. Voir le tableau suivant.

Composants	Opérations	Mch1	Mch2	Mch3	Mch4	Mch5	Mch6	Mch7	Mch8
Composant 1	**Opr1**	195		471			286		
	Opr2	500	112	365					124
	Opr3		330		265				
Composant 2	**Opr1**	113						519	
	Opr2	246	222			157			
	Opr3		418	570			100		
	Opr4	615		133	96			324	
Composant 3	**Opr1**	145	112			122			
	Opr2	114						111	231
	Opr3	509		90			24		
	Opr4		535			84	204		
	Opr5		449						70
Composant 4	**Opr1**	200			416			662	
	Opr2	347		123					
	Opr3	294	312						401
	Opr4	500			395			108	196

Tableau 4.4: Temps de traitement

- **Le temps de changement d'outils**

Le temps de changement d'outil de notre exemple est résumé dans le tableau suivant :

Outils / Outils	Outil 1	Outil 2	Outil 3	Outil4	Outil5	Outil6
Outil 1	0	59	12	80	70	61
Outil 2	10	0	19	20	15	24
Outil 3	12	18	0	8	15	12
Outil 4	47	15	12	0	20	49
Outil 5	37	10	21	18	0	11
Outil 6	23	18	42	11	29	0

Tableau 4.5: Temps de changement d'outils.

- **Le temps de reconfiguration**

Le temps de changement de configuration des différentes machines de notre exemple est exposé dans le tableau suivant :

	Configurations	Cf1	Cf2	Cf3	Cf4
Mch 1	**Cf1**	00	32	54	
	Cf2	25	00	18	
	Cf3	29	45	00	
Mch 2	**Cf1**	00	49	75	47
	Cf2	49	00	34	16
	Cf3	68	43	00	70
	Cf4	64	13	11	00
Mch 3	**Cf1**	00	34		
	Cf2	09	00		
Mch 4	**Cf1**	00	48		
	Cf2	22	00		
Mch 5	**Cf1**	00	51	12	
	Cf2	29	00	62	
	Cf3	36	84	00	
Mch 6	**Cf1**	00	87	92	
	Cf2	25	00	33	
	Cf3	39	14	00	
Mch 7	**Cf1**	00	55		
	Cf2	17	00		
Mch 8	**Cf1**	00	13	17	61
	Cf2	29	00	31	54
	Cf3	38	42	00	19
	Cf4	11	60	23	00

Tableau 4.6: Temps de reconfiguration.

- **Le temps de changement des machines**

Le temps de changement des machines de notre exemple est détaillé dans le tableau 4.7 :

Machines	Mch1	Mch2	Mch3	Mch4	Mch5	Mch6	Mch7	Mch8
Mch 1	00	80	62	58	91	52	85	79
Mch 2	68	00	68	85	75	68	52	58
Mch 3	62	56	00	66	64	95	91	73
Mch 4	52	98	73	00	79	45	82	41
Mch 5	85	57	60	97	00	53	66	77
Mch 6	79	52	81	59	93	00	73	63
Mch 7	44	83	58	76	64	53	00	95
Mch 8	98	65	42	57	53	49	63	00

Tableau 4.7: Temps de changement des machines.

Comme expliqué dans le chapitre 3, les informations concernant les coûts se composent de cinq sommes (Coût d'utilisation des machines, coût de changement de machines, coût de reconfiguration d'une machine, coût d'utilisation et de changement des outils).

Nous présentons, dans ce qui suit, la quantité de chaque coût.

- **Coût d'utilisation des machines**

Le coût d'utilisation des machines de notre exemple est détaillé dans le tableau suivant :

Machines	Coût d'utilisation
Mch 1	50
Mch 2	20
Mch 3	25
Mch 4	22
Mch 5	30
Mch 6	26
Mch 7	39
Mch 8	36

Tableau 4.8: Coût d'utilisation des machines

- **Coût d'utilisation d'outils**

Le coût d'utilisation des outils de notre exemple est détaillé dans le tableau suivant :

Outils	Coût d'utilisation
Outil 1	08
Outil 2	14
Outil 3	25
Outil 4	10
Outil 5	15
Outil 6	19

Tableau 4.9 : Coût d'utilisation d'outils.

- **Coût de changement des machines**

Le coût de changement des machines de notre exemple est résumé dans le tableau suivant

:

Machines	Mch 1	Mch 2	Mch 3	Mch 4	Mch 5	Mch 6	Mch 7	Mch 8
Mch1	00	06	12	15	17	19	14	19
Mch2	07	00	10	04	06	13	14	11
Mch3	11	19	00	15	07	08	09	15
Mch4	12	14	10	00	11	16	19	15
Mch5	12	14	09	21	00	19	18	15
Mch6	08	05	14	07	11	00	10	16
Mch7	16	11	08	13	09	10	00	11
Mch8	06	17	12	15	19	20	14	00

Tableau 4.10 : Coût de changement des machines.

- **Le coût de changement d'outils**

Le coût de changement d'outils de notre exemple est expliqué dans le tableau suivant:

Outils	Outil 1	Outil 2	Outil 3	Outil4	Outil 5	Outil6
Outil 1	00	16	07	05	8.5	08
Outil 2	8.5	00	10	05	09	7.5
Outil 3	2	5.5	00	4.5	16	12
Outil 4	4.5	9.5	07	00	11	10
Outil 5	14	08	05	13	00	06
Outil 6	10	06	17	09	07	15

Tableau 4.11: Coût de changement des outils.

- **Coût de reconfiguration des machines**

Le coût de reconfigurations des machines de notre exemple est résumé dans le tableau suivant:

Machine	Configurations	Cf1	Cf2	Cf3	Cf4
Mch 1	**Cf1**	00	13	03	
	Cf2	15	00	11	
	Cf3	09	19	00	
Mch 2	**Cf1**	00	09	17	17
	Cf2	19	00	24	06
	Cf3	18	13	00	26
	Cf4	10	13	11	00
Mch 3	**Cf1**	00	14		
	Cf2	25	00		
Mch 4	**Cf1**	00	18		

	Cf2	22	00		
Mch 5	**Cf1**	00	15	12	
	Cf2	22	00	16	
	Cf3	16	14	00	
Mch 6	**Cf1**	00	17	12	
	Cf2	05	00	23	
	Cf3	19	14	00	
Mch 7	**Cf1**	00	15		
	Cf2	17	00		
Mch 8	**Cf1**	00	13	17	24
	Cf2	09	00	13	30
	Cf3	28	12	00	19
	Cf4	11	08	23	00

Tableau 4.12 : Coût de reconfiguration des machines.

4.4.2 Exécution de l'algorithme MOABC

À ce stade, toutes les données des machines, informations sur le produit, le temps et le coût et sont identifiés. Pour exécuter l'algorithme (MOABC), il est indispensable de déterminer les paramètres de MOABC. Les paramètres pris en compte pour notre cas sont les suivants :

- Taille de la population =20
- Limit =4
- Facteur de Cout = 0.6
- Facteur de temps = 0.4
- Nombre de générations = 10

Après l'exécution de l'algorithme, on obtient la figure qui représente le front Pareto réalisé par l'algorithme MOABC

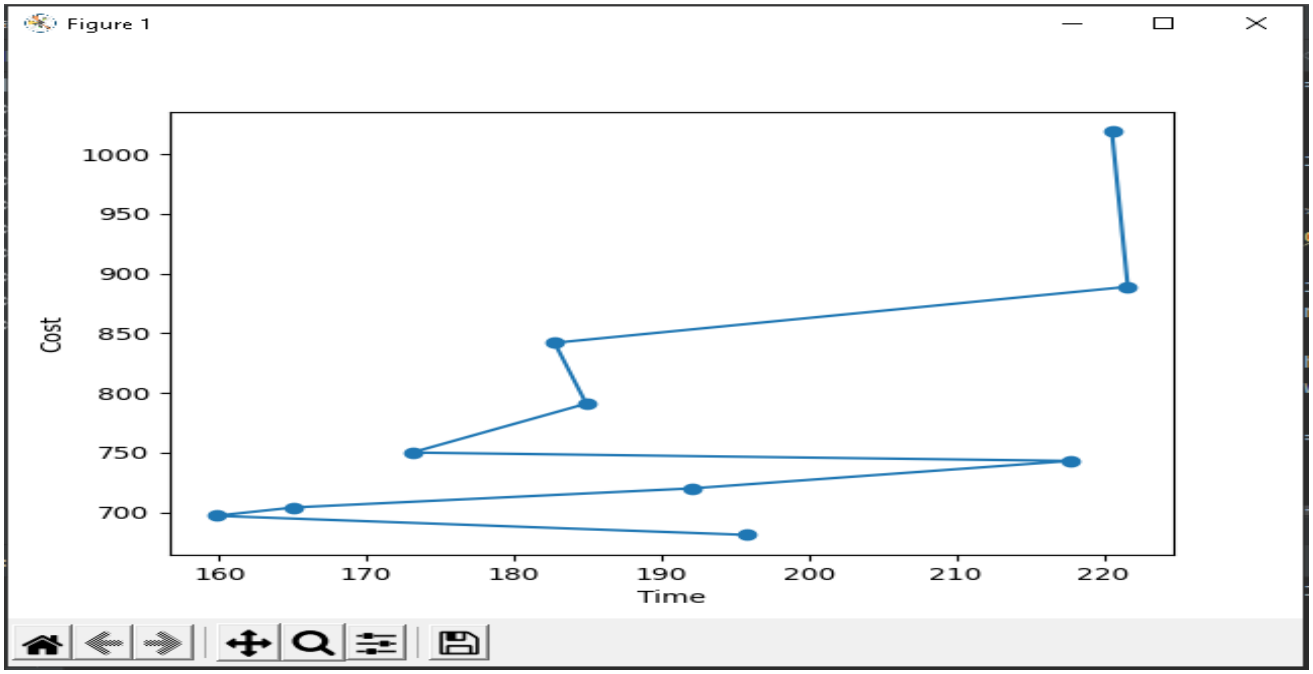


Figure 4.27: courbe de l'algorithme MOABC

Le coût total et le temps total sont résumés dans le tableau suivant :

Coût total	Temps total
220.55	1019
221	889
182	842
184.91	791
173.09	750
217.13	743
192.09	720
165.09	704
159.77	697
195.77	681

Tableau 4.13: Les valeurs du coût total et temps total pour chaque solution.

Et enfin, pour exécuter la séquence d'opérations qui a été représenté dans le tableau 4.2, les meilleures solutions représentants les machines, les configurations et les outils sont résumés comme suivant :

Solution1
Machine : [3, 2, 5, 5, 5, 5, 4, 8, 5, 7, 1, 8, 7, 6, 2, 2]
configuration : [2, 4, 1, 1, 1, 3, 1, 4, 3, 1, 4, 2, 2, 1, 2, 2]
Outil : [6, 6, 6, 4, 6, 6, 1, 5, 1, 1, 4, 2, 1, 6, 5, 6]
Solution 2
Machine : [1, 5, 3, 8, 1, 1, 3, 8, 3, 5, 2, 2, 2, 2, 8, 2]
configuration : [1, 1, 2, 4, 1, 2, 1, 3, 1, 3, 4, 2, 1, 2, 1, 2]
Outil : [2, 6, 4, 5, 3, 1, 6, 5, 6, 5, 5, 6, 6, 5, 4, 5]
Solution 3
Machine : [3, 2, 8, 8, 5, 5, 2, 8, 5, 2, 1, 8, 7, 6, 2, 2]
configuration : [2, 4, 3, 3, 1, 2, 4, 4, 3, 4, 3, 2, 2, 1, 2, 2]
Outil : [4, 6, 5, 4, 6, 6, 5, 5, 6, 5, 4, 2, 1, 5, 5, 6]
Solution 4
Machine : [6, 2, 3, 5, 5, 5, 7, 8, 2, 2, 6, 8, 2, 2, 5, 5]
configuration : [3, 4, 1, 1, 1, 2, 1, 4, 2, 4, 4, 2, 1, 1, 3, 3]
Outil : [6, 6, 6, 6, 6, 5, 1, 5, 6, 5, 5, 2, 6, 6, 5, 5]
Solution 5
Machine : [6, 2, 3, 5, 5, 5, 7, 8, 2, 2, 3, 8, 2, 2, 5, 5]
configuration : [3, 4, 1, 1, 1, 3, 1, 4, 2, 4, 3, 2, 1, 1, 3, 2]
Outil : [6, 6, 6, 4, 6, 5, 1, 5, 6, 5, 5, 2, 6, 6, 5, 5]

Solution 6
Machine : [6, 2, 3, 5, 5, 5, 7, 8, 2, 2, 8, 8, 2, 2, 5, 5]
configuration : [3, 4, 1, 1, 1, 2, 1, 4, 2, 4, 3, 2, 1, 1, 3, 3]
Outil : [4, 3, 6, 6, 6, 5, 1, 5, 6, 5, 5, 2, 6, 6, 5, 5]
Solution 7
Machine : [6, 2, 3, 5, 5, 5, 7, 8, 2, 2, 1, 8, 2, 2, 5, 5]
configuration : [3, 4, 1, 1, 1, 2, 1, 4, 2, 4, 3, 2, 1, 1, 3, 3]
Outil : [6, 6, 6, 4, 4, 6, 1, 2, 6, 5, 4, 2, 6, 6, 4, 6]
Solution 8
Machine : [6, 2, 3, 5, 5, 5, 7, 8, 2, 2, 6, 8, 2, 2, 5, 5]
configuration : [3, 4, 1, 1, 1, 2, 1, 4, 2, 4, 3, 2, 1, 1, 3, 3]
Outil : [4, 3, 6, 6, 6, 5, 1, 5, 6, 5, 5, 2, 6, 6, 4, 6]
Solution 9
Machine : [3, 2, 3, 5, 3, 5, 2, 8, 2, 2, 5, 8, 2, 6, 2, 2]
configuration : [2, 4, 1, 1, 1, 3, 4, 4, 2, 4, 2, 2, 1, 1, 2, 2]
Outil : [6, 6, 6, 4, 6, 6, 5, 2, 6, 5, 5, 2, 6, 5, 5, 5]
Solution 10
Machine : [3, 2, 3, 8, 5, 5, 2, 8, 5, 2, 4, 8, 7, 6, 2, 2]
configuration : [2, 4, 1, 3, 1, 3, 4, 4, 3, 4, 3, 2, 2, 1, 2, 2]
Outil : [4, 3, 6, 2, 6, 6, 5, 5, 6, 5, 5, 2, 6, 6, 5, 6]
Solution 11
Machine : [3, 2, 3, 5, 3, 5, 2, 8, 2, 2, 1, 8, 7, 6, 2, 2]
configuration : [2, 4, 1, 1, 1, 3, 4, 4, 2, 4, 4, 2, 2, 1, 2, 2]
Outil : [4, 6, 6, 6, 6, 5, 5, 2, 6, 5, 4, 2, 6, 5, 5, 5]

4.5 Conclusion

Dans ce chapitre, nous avons présenté les outils que nous avons utilisés pour faire notre application, puis les résultats de notre implémentation sous forme d'un ensemble d'interfaces utilisateur graphiques (GUI). En dernière section, on a traité et testé les résultats grâce à l'exemple appliqué sous les paramètres prédéfinis.

Conclusion générale

Pendant des dernières années, il y a eu un intérêt croissant pour les systèmes de production reconfigurables. Le RMS est classifié comme étant un paradigme manufacturier changeable qui est plus adapté pour faire face aux exigences de l'environnement manufacturier actuel. Ceci est possible grâce à la conception des RMSs qui intègre dès le départ l'évolutivité, la reconfigurabilité ainsi que les changements aux niveaux des fonctionnalités et/ou de ses capacités.

Au cours de préparation de ce mémoire, nous avons essayé de trouver une solution aux problèmes de génération des processus de fabrication de production dans le Système Manufacturier Reconfigurable et ainsi nous avons traité le problème d'optimisation multi objectif pour sélectionner les meilleures machines reconfigurables pouvant réaliser un produit final dans un temps et couts minimisés basant sur l'adaptation d'un algorithme nommé «MOABC ».

Durant la réalisation de notre projet nous avons appris des connaissances sur :

- Les problèmes d'optimisation multi objectifs.
- L'algorithme d'optimisation multi objectifs « ABC ».
- Les Systèmes Manufacturiers Reconfigurables « RMSs ».

Basant sur les travaux de [1], [2], [22] Nous avons implémenté l'algorithme d'optimisations multi-objectif « MOABC » pour obtenir les meilleures solutions.

La conception de telle approche ainsi que l'expérimentation préliminaire du modèle proposé, ont donné lieu à des propositions et à des réflexions définies comme étant des perspectives de recherche, qui sont les suivants :

- D'ajouter d'autres foncions objectifs
- Faire une comparaison entre ABC et une autre méta-heuristique comme NSGA II, MOGA, ACO (Ant Colony Optimization).
- D'ajouter d'autres indicateurs de performances.

Bibliographies:

[1] - Masood Ashraf , Some Aspects of Planning for Reconfigurable Manufacturing Systems. PhD Thesis, Aligarh Muslim University, 2018.

[2]- Khaled Lameche ,Proposition d'une méthodologie pour la conception des systèmes de production reconfigurables et d'un outil associé d'aide à la décision par simulation de flux.

[3]- Safia Lamrani , Raid Hasan, Processus de conception des Systèmes Manufacturiers Reconfigurables (SMR): Approche à l'aide des Réseaux de Pétri (RdP, Patrick Martin1 1 Laboratoire de Génie Industriel et Production Mécanique (LGIPM).

[4]- Raïd HASAN, Patrick MARTIN, Démarche de conception intégrée produit - Système Manufacturier Reconfigurable (SMR) Luc LOSSENT CRAN – CNRS UMR 7039

[5]- Hichem Haddou Benderbal ,Développement d'une nouvelle famille d'indicateurs de performance pour la conception d'un système manufacturier reconfigurable (RMS) : approches évolutionnaires multicritères PhD Thesis, 2018.

[6]- Y koren, Book: Reconfigurable Manufacturing Systems and Transformable Factories, Springer, 2006, pp. 27-45. General RMS Characteristics Comparison with Dedicated and Flexible System .

[7]- G. Zhang, R. Liu, L. Gong, et Q. Huang ,« An analytical comparison on cost and performance among DMS, AMS, FMS and RMS», in Reconfigurable manufacturing systems and transformable factories, Springer

[8]- HAMED Mostef ,Thèse présentée en vue de l'obtention du diplôme de :
Doctorat en Sciences en :Génie Electrique , Contribution à l'optimisation multi objective Dynamique du coût et l'émission de gaz considérant les contraintes pratiques et de sécurités

[9]- Mohsen Ejday Optimisation multi-objectifs à base de métamodèle pour les procédés de mise en forme , l'École nationale supérieure des mines de Paris Spécialité " Mécanique Numérique ''

[10] MAHDI SAMIR , Optimisation Multiobjectif Par Un Nouveau Schéma De Coopération Méta/Exacte , Université Mentouri de Constantine Faculté d'Ingénieur, Département de l'Informatique.

[11]- Patrick D. Surry and Nicholas J ,The COMOGA method: constrained optimisation by multi-objective genetic algorithms. Radcliffe Quadstonc Ltd, lG Chester Street, Edinburgh, EH3 7RA, UK, and: Department of Mathematics, University of Edinburgh.

[12]- Yann Collette, Patrick Siarry. Optimisation multi-objectif. (2002). Éditions eyrolles 61, Bld Saint-Germain 75240 Paris Cedex 05. Livre.

[13]-Younes Bahmani, Optimisation multicritère de l'ordonnancement des activités de la production et de la maintenance intégrées dans un atelier Job Shop.

[14] Eckart Zitzler Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications A dissertation submitted to the Swiss Federal Institute of Technology Zurich for the degree of Doctor of Technical Sciences.

[15]-Florian Mazière. Modèles de parallélisme pour les métaheuristiques multi-objectifs. Informatique [cs]. Universite de Reims Champagne-Ardenne, 2019.

[16]- Massimiliano Caramia • Paolo Dell'Olmo ,Multi-objectiveManagement Freight Logistics Increasing Capacity, Service Level and Safety with Optimization Algorithms.

[17]- J. Dréo A. Pétrowsk P. Siarry E. Taillard, Metaheuristics for Hard Optimization Methods and Case Studies Simulated Annealing, Tabu Search, Evolutionary and Genetic Algorithms, Ant Colonies,…, livre.

[18]- Peter Dueholm Justesen, Multi-objective Optimization using Evolutionary Algorithms, Department of Computer Science University of Aarhus Denmark, January 13, 2009.

[19]- MULTIOBJECTIVE OPTIMIZATION AND GENETIC ALGORITHMS , open source engeerining ,A scilab Professional Partner, www.openeering.com

[20]- KHALED LOUKHAOUKHA ,Tatouage numérique des images dans le domaine des ondelettes basé sur la décomposition en valeurs singulières et l'optimisation multi-objective, Thèse présentée à la Faculté des études supérieures de l'Université Laval dans le cadre du programme de doctorat en Génie Électrique , Faculté des Sciences et de Génie UNIVERSITÉ LAVAL QUÉBEC.

[21]- Peio Loubière, Amélioration des métaheuristiques d'optimisation `a l'aide de l'analyse de sensibilit´ ,Thèse présentée pour l'obtention du grade de DOCTEUR DE L'UNIVERSITE PARIS-EST.

[22]- BENSMAÏNE Abderrahmane ,Algorithmes évolutionnaires et méthodes approchées multicritères pour la génération des processus de fabrication dans un environnement reconfigurable, THÈSE Pour obtenir le titre de Docteur de l'université de Lorraine En « Automatique, Traitement du Signal et des Images, Génie Informatique ».

[23]- Riadh Madiouni. Contribution à la synthèse et l'optimisation multi-objectif par essaims particulaires de lois de commande robuste RST de systèmes dynamiques. Informatique et langage [cs.CL]. Université Paris-Est, 2016.

[24]- Pr L.Djerou , Algorithmes évolutionnaires non élitistes pour l'optimisation multiobjectif, Module: Optimisation Multicritères et dans l'Incertain, Université Mohamed KhiderBiskra Faculté des sciences exactes et des sciences de la nature et de la vie Département d'informatique.

[25]- ARAVIND SESHADRI ,MULTI-OBJECTIVE OPTIMIZATION USING EVOLUTIONARY ALGORITHMS (MOEA).

[26]- Mauro Birattari ,Tuning Metaheuristics: A Machine Learning Perspective, Mauro Birattari, Ph.D. Chercheur qualifié du F.R.S.-FNRS IRIDIA, CoDE, FSA - CP 194/6 Université Libre de Bruxelles.

[27]- Souhil MOUASSA, Optimisation de l'écoulement de puissance par une méthode métaheuristique (technique des abeilles) en présence d'une source renouvelable (éolienne) et des dispositifs FACTS, Faculté de Technologie Présenté au département d'Electrotechnique Pour obtenir le diplôme De Magister En Electrotechnique, UNIVERSITE Ferhat ABBAS SETIF - (ALGERIE).

[28]- RAIHANI Hind, Adaptation de l'algorithme des colonies d'abeilles
pour l'optimisation et le dimensionnement des circuits intégrés analogiques, Université Sidi Mohamed Ben Abdellah Faculté des sciences ET Techniques –Fès Département Génie Electrique.

[29]- HARUN YAHYA, Book "The Miracle of the honeybee", G. M. D. Cd., Ed. Okmeydani-Istanbul-Turkey,2007.

[30]- Nurhan KARABOGA, Mehmet Bahadır CETINKAYA , A novel and efficient algorithm for adaptive filtering: Artificial bee colony algorithm, Department of Electronics Engineering, Faculty of Engineering, Erciyes University, 38039, Melikgazi, Kayseri, TURKEY.

[31]- Hamid BOUALI, Bachir BENHALA, Hamid BOUYGHF, Performance study of Multi-Objective Artificial Bee Colony (MOABC) Algorithm by Numerical Problems Benchmark, Conference Paper · April 2020, Université Moulay Ismail LEAB,Department of physics FS,Meknes,Moulay Ismaïl,University, Morocco.

[32]- Pr L.Djerou , Optimisation multiobjectif, Module: Optimisation Multicritères et dans l'Incertain, Université Mohamed KhiderBiskra Faculté des sciences exactes et des sciences de la nature et de la vie Département d'informatique.

[33]- Ravi Kumar, Arun Kumar, Sumit Kumar , Review of Trends and development in flexible and reconfigurable manufacturing systems, International Research Journal of Engineering and Technology (IRJET).

[34]- M. G. MEHRA B I , A . G . ULSOY and Y. KOREN, Reconfigurable manufacturing systems: Key to future manufacturing.

[35]- HOUCINE DAMMAK, RECONFIGURATION DYNAMIQUE DES SYSTÈMES MANUFACTURIERS NON FIABLES,Universite LAVAL Québec, Canada.

[36]- Mostafa G. Mehrabi, A.Galip Ulsoy and Yoram Koren, Reconfigurable manufacturing systems and their enabling technologies, Department of Mechanical Engineering and Applied Mechanics, The University of Michigan, Ann Arbor, MI 48109-2125, USA.

[37]- Hasan Aladad. Conception du système de fabrication de pièces mécaniques en grand série : formalisation de la configuration géométrique (enveloppe) et cinématique de Machine-Outil Reconfigurable (MOR). Sciences de l'ingénieur [physics]. Arts et Métiers ParisTech, 2009.

[38]- Mohamed ESSAFI, Conception et optimisation d'allocation de ressources dans les lignes d'usinage reconfigurables, Docteur de l'École Nationale Supérieure des Mines de Saint-Étienne Génie Industriel .

[39]- Imad Chalfoun. Conception et deploiement des Systemes de Production Reconfigurables et Agiles (SPRA). Micro et nanotechnologies/Microelectronique. Universite Blaise Pascal - Clermont-Ferrand II, 2014.

[40]- M. Gitizadeh ,H. Khalilnezhad ,R. Hedayatzadeh, TCSC allocation in power systems considering switching loss using MOABC algorithm Article in Electrical Engineering , June 2013.

[41]-Guillermo CAMPOS CIRO, Développement de méthodes d'ordonnancement efficaces et appliquées dans un système de production mécanique. (2003)

[42] Y. Koren, X. Gu, et W. Guo, « Reconfigurable manufacturing systems: Principles, design, and future trends », *Front. Mech. Eng.*, vol. 13, no 2, p. 121–136, 2018.

[43]- Le tutoriel Python, https://www.python.org/

Annexe

Note importante :

Dans cette annex, nous avons mentionné les fonctions les plus importantes de notre application, le reste des fonctions est dans le code source joint au mémoire.

1. Enrergistrement les données de cout

Le code suivant illustre comment enregistrer la partie des données représentée par le coût.

```
#  Save cost data in csv files
def save_Cost(self):

    os.getcwd()  # get the current working directory

    # self.CM = np.zeros((self.NM, self.n, max(self.NOPF)))
    with open("New Problem\Cost Data\\CM.csv", "w", newline='') as fcm:
        writer = csv.writer(fcm , delimiter=';')
        writer.writerow(self.CM)


    # cost of using a tool
    # self.CT = np.zeros((self.NM, self.NT, self.n, max(self.NOPF)))
    with open("New Problem\Cost Data\\CT.csv", "w", newline='') as fct:
        writer = csv.writer(fct , delimiter=';')
        writer.writerow(self.CT)


    # tool changeover cost
    with open("New Problem\Cost Data\\TCCost.csv", "w", newline='') as tccost:
        writer = csv.writer(tccost , delimiter=';')
        writer.writerow(self.TCCost)


    # machine change cost
    with open("New Problem\Cost Data\\MCCost.csv", "w", newline='') as mccost:
        writer = csv.writer(mccost , delimiter=';')
        writer.writerow(self.MCCost)


    # configuration change cost
    with open("New Problem\Cost Data\\CCCost.csv", "w", newline='') as cccost:
        writer = csv.writer(cccost , delimiter=';')
        writer.writerow(self.CCCost)
```

2. Enregistrement les données entrées

Le code suivant montre la fonction qui est chargé toutes les données saisies par utilisateur.

```
# this method saves inputs in csv files
def save_inputs(self):

    os.getcwd()  # get the current working directory

    if not os.path.exists("New Problem"):  # if folder "New Problem" doesn't exist, it should create it
        os.mkdir("New Problem")  # create physic directory
    else:  # delete the folder "New Problem" and create a new one
        shutil.rmtree("New Problem")  # remove directory "New Problem" and its content
        os.mkdir("New Problem")

    # create directory for saving Machines data
    #  test if the directory is already exist with this path
    if not os.path.exists("New Problem\Machines Data"):
        os.mkdir("New Problem\Machines Data")  # create physic directory
    self.save_machines()

    # create directory for saving Product data
    #  test if the directory is already exist with this path
    if not os.path.exists("New Problem\Product Data"):
        os.mkdir("New Problem\Product Data")  # create physic directory
    self.save_product()

    # create directory for saving time data
    #  test if the directory is already exist with this path
    if not os.path.exists("New Problem\Time Data"):
        os.mkdir("New Problem\Time Data")  # create physic directory
    self.save_Time()

    # create directory for saving cost data
    #  test if the directory is already exist with this path
    if not os.path.exists("New Problem\Cost Data"):
        os.mkdir("New Problem\Cost Data")  # create physic directory
    self.save_Cost()
```

3. Initialisation (codage)

Le codage des solutions est une étape clé dans l'implémentation des métaheuristiques, il consiste à passer de la représentation réelle des solutions vers une représentation codée.

```
# ***********************************Individual Initialisation*****************************************
def individual_generation(self):
    for i in range(3):
        line = [round(random.uniform(0.01, 0.99), 2) for j in range(self.TNOP)]
        self.encoded_individual.append(line)
```

4. Décodage

Le décodage des machines est la troisième étape du décodage d'une solution. Ce décodage doit être effectué après celui des composants et des opérations.

Le code suivant illustre la méthode qui fait décodage de caractéristique.

```
# ************************************Individual Decoding Process*********************
def features_decoding(self):
    if len(self.features) != 0 and self.TNOP != 0:
        # self.decoded_individual = np.zeros((5, self.TNOP))
        self.decoded_individual_initialisation()
        i = 0
        for item in self.encoded_individual[0]:
            #print('@@@@@',item)
            p = item * len(self.features)
            integer, decimal = divmod(p, 1)
            if decimal < 0.5 or p == 0.5:
                pr = int(round(p)) + 1
            else:
                pr = int(round(p))

            if pr <= len(self.features):
                pr1 = self.features[pr - 1]
                self.decoded_individual[0][i] = pr1
                self.features.remove(self.features[pr - 1])
                i += 1
    else:
        print("len(self.features) = ", self.features)
```

- **Décodage d'un individu**
  Ce code présente toutes les méthodes de décodage pour décoder un individu dans notre application.

```
# ***************************************************************************************************************
def individual_decoding(self):
    self.machines_decoding()
    self.configurations_decoding()
    self.tools_decoding()
```

- **Population initiale**

  La population initiale est générée de façon aléatoire selon le nombre d'individus définis.

```
def population_initialisation(self, pop_size):
    self.pop_size = pop_size
    """self.Calculate_Food_Sources()"""
    while len(self.population) < self.nrbOFoodSource:
        ind = Fitness()
        ind.individual_generation()
        ind.individual_decoding()
        #print("@@@@incoded individual",ind.decoded_individual)
        if ind.decoded_individual not in self.population:
            ind.ftime_individual()
            ind.fcost_individual()
            self.population.append(ind)
```

5. Coût d'utilisation des machines (MUC)

C'est le coût d'utilisation des machines pour effectuer toutes les opérations d'une production.

```
# ************************************Cost Function*************************************************************
def Machine_Usage_Cost(self):
    MUC = 0
    # self.CM = np.zeros((self.NM, self.n, max(self.NOPF)))
    for j in range(len(self.decoded_individual[0])):
        self.MUC += self.CM[int(self.decoded_individual[2][j])-1] * self.PrTime[int(self.decoded_individual[0][j])-1][
                                                                    int(self.decoded_individual[1][j])-1][
                                                                    int(self.decoded_individual[2][j])-1]
    #print('muc', self.MUC)
    return self.MUC
```

6. Coût total

Tous les coûts nécessaires sont calculés, présente par le code suivant.

```
# ********************************************************************************************************************
def fcost_individual(self):
    self.total_cost = self.Machine_Usage_Cost() + self.Configuration_Change_Cost() + self.Tool_Usage_Cost() + \
                      self.Tool_Change_Cost() + self.Machine_Change_Cost()
```

7.Temps de changement des configurations (CCT)

C'est le temps de passage d'une configuration à une autre sur la même machine.

```
# ********************************************************************************************************************
def Configuration_Change_Time(self):
    # CCT = 0
    calculated_time = []
    for j in range(len(self.decoded_individual[2])):
        CCTi = 0
        if self.decoded_individual[2].count(self.decoded_individual[2][j]) > 1 and \
                        self.decoded_individual[2][j] not in calculated_time:
            time = []
            for i in range(len(self.decoded_individual[2])):
                if self.decoded_individual[2][j] == self.decoded_individual[2][i]:
                    time.append(i)
            for k in range(len(time) - 1):
                # self.CCTime = np.zeros((self.NM, max(self.NC), self.n, max(self.NOPF),
                # max(self.NC), self.n, max(self.NOPF)))
                CCTi += self.CCTime[int(self.decoded_individual[2][j])-1][
                                    int(self.decoded_individual[3][time[k]])-1][
                                    int(self.decoded_individual[3][time[k + 1]])-1]

            calculated_time.append(self.decoded_individual[2][j])
        self.CCT += CCTi
    return self.CCT
```

7. Temps total

Le temps total est englobé les différents temps nécessaires pour la réalisation de l'ensemble des opérations donnant lieu au produit fini.

```
# ********************************************************************************************************************
def ftime_individual(self):
    self.total_time = self.Processing_Time() + self.Configuration_Change_Time() + self.Tool_Changeover_Time() + \
                      self.Machine_Change_Time()
```

8. Croisement

L'opérateur du croisement produit un enfant en prenant deux parents et un point de croisement, présenté par le code ci-dessous.

```
def crossover(self, parent01, parent02, cross_point):

    child01 = Fitness()


    # before cross point
    for k in range(3):
        a  = []
        i = 0
        while i < cross_point:
            #for k in range(3):
            a.append(parent01.encoded_individual[k][i])
            i += 1
        # after cross point
        i = cross_point
        while i < len(parent01.encoded_individual[0]):
        #for k in range(3):
            a.append(parent02.encoded_individual[k][i])
            i += 1
        child01.encoded_individual.append(a)
    return child01
```

9. Fonction de tri rapide non dominée (Fast Non-dominated Sorting)

Cette technique renvoie une population classée (F) avec plusieurs fronts (F1, F2, ……) en fonction de deux objectifs, minimiser le coût et minimiser le temps, le code suivant illustre cette méthode.

```
def fast_non_dominated_sorting(self, pop):

    F = {1: []}
    for p in pop:
        p.S = []
        p.nS = 0
        for q in pop:
            if (p.total_cost < q.total_cost) and (p.total_time < q.total_time) :
                p.S.append(q)#88888888888888888888

            elif (q.total_cost < p.total_cost) and (q.total_time < p.total_time):
                p.nS += 1

        if p.nS == 0:
            p.rank = 1
            F[1].append(p)
    i = 1
    while len(F[i]) != 0:
        Q = []
        for p in F[i]:
            for q in p.S:
                q.nS -= 1
                if q.nS == 0:
                    q.rank = i + 1
                    Q.append(q)

        i += 1
        F[i] = Q


    return F
```

10. L'algorithme de colonies des abeilles artificielles multi-objectif (MOABC)

- **Les abeilles employées**

  Pour chaque abeille employée trouve une nouvelle position de source de nourriture, présenté par le code suivant :

```
def employedBees(self):
    for j in range(self.nrbOFoodSource):
        #print("individual",self.population[j])
        crossPoint = random.randint(1,len(self.population[j].decoded_individual[0])-2)
        parent2Index = random.randint(0,self.nrbOFoodSource-1)
        while parent2Index == j :
            parent2Index = random.randint(0,self.nrbOFoodSource-1)
        newIndividual = self.crossover(self.population[j], self.population[parent2Index], crossPoint)
        newIndividual.individual_decoding()
        if newIndividual.decoded_individual not in self.population:
            newIndividual.ftime_individual()
            newIndividual.fcost_individual()
        # lazem ndir m3yar bah ngoul itha solution khir men lo5ra**********----------///////
        inndv = self.whosTheBestIndividual(self.population[j], newIndividual)
        if inndv == self.population[j]:
            self.Trials_Vector[j] += 1
        elif inndv == newIndividual:
            self.population[j] = newIndividual
```

- **Les abeilles spectatrices**

Le code suivant illustre la phase des abeilles spectatrices

```
def outlookerBees(self):
    self.indvidualProbaCalculation()
    self.iff = random.uniform(-0.9999,0.9999999)

    for i in range(self.nrbOFoodSource):
        if self.SourceFoodProba[i]>= self.iff :
            crossPoint = random.randint(1,len(self.population[i].encoded_individual[0])-2)
            parent2Index = random.randint(0,self.nrbOFoodSource-1)
            while parent2Index == i :
                parent2Index = random.randint(0,self.nrbOFoodSource-1)
            newIndividual = self.crossover(self.population[i], self.population[parent2Index], crossPoint)
            newIndividual.individual_decoding()
            if newIndividual.decoded_individual not in self.population:
                newIndividual.ftime_individual()
                newIndividual.fcost_individual()
            inndv = self.whosTheBestIndividual(self.population[i], newIndividual)
            if inndv == self.population[i]:
                self.Trials_Vector[i] +=1
            elif inndv == newIndividual:
                self.population[i] = newIndividual
    self.find_individual()
```

- **Les abeilles scoutes**

L'abeille scoute est chargée de trouver une nouvelle position de source de nourriture et d'évaluer la qualité de leur nectar, cette phase présentée par le code ci-dessous.

```
def scootbees(self):
    change = True
    while change == True :
        change = False
        for i in range(self.nrbOFoodSource):
            if(self.Trials_Vector[i] == self.limit):
                self.Trials_Vector[i] = 0
                change = True
                ind = Fitness()
                ind.individual_generation()
                ind.individual_decoding()
                ind.ftime_individual()
                ind.fcost_individual()
        if change :
            self.outlookerBees()
            self.scootbees()
```

11.Main MOABC

Ce code illustre l'implémentation de l'algorithme MOBC.

```
def ABCmain_loop(self, pop, nb_iterations):

        # open an output file
    os.getcwd()
    orig_stdout = sys.stdout
    output_file = open('Pareto Front/Pareto Front.txt', 'w')
    sys.stdout = output_file


    self.Calculate_Food_Sources()
    self.population_initialisation(self.nrbOFoodSource)
    self.Calculate_limit()
    self.intiate_Trials_Vector()
    self.intiate_Proba_Vector()

    t = 0
    print("\n*****************************************************************************************\n")
    print("Informations d'entrée\n")
    print("Taille de la population = ", self.nrbOFoodSource)
    print("\n")
    print("Nombre des générations = ", self.generations_nb)
    print("\n")
    print("Limit= ", self.limit)
    print("\n")
    print("\n*****************************************************************************************\n")
    t=0
    while t < nb_iterations:
        self.employedBees()
        self.outloockerBees()
        self.scootbees()
        t += 1
```

```
    asb = self.remove_duplicates(self.front)



    for i in range(len(asb)):
        self.printing_idividual(asb[i].decoded_individual)#zid True bah tafichi details
        print("\n")
        print("************************************************************")
        print("* Coût total        = ", round(asb[i].total_cost,3))
        print("* Temps total       = ", round(asb[i].total_time,3))
        print("************************************************************")
        print("\n")

    X = []
    Y = []
    for i in range(len(asb)):
        X.append(asb[i].total_cost)
        Y.append(asb[i].total_time)



    sys.stdout = orig_stdout
    output_file.close()

    open_pareto_front = open('Pareto Front/Pareto Front.txt').read()
    codebox("Les Meilleurs  Solutions", text=open_pareto_front, title="Meilleurs Solutions")
```