Remerciements

Je tiens à remercier le dieu, le tout puissant de ma donnée la patience, la santé et le courage pour finir ce travail.

Je tiens à remercier profondément mon encadreur : Monsieur KALFALI TOUFIK qui m'a encouragé à
faire le maximum d'efforts dans ce travail, sans ces encouragements ce mémoire n'aurait sans doute pas abouti.

Et à toutes les personnes qui m'ont aidé de près et de loin.

Dédicace

C'est avec joie que je dédie ce modeste travail :

À mes très chers parents qui j'espère rendre fière, pour leurs patiences et encouragements. Que dieu les protègent.

À mes amis et mes collègues pour les bons moments que j'ai passé avec eux

Hacene halim

Sommaire

Liste des figures

Liste des Tableaux

Introduction générale

L'optimisation combinatoire occupe une place très importante en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire. Les problèmes d'optimisation sont utilisés pour modéliser de nombreux problèmes dans différents secteurs de l'industrie (télécommunications, électronique, mécanique, chimie, transport, …).

Le problème de placement de composants électroniques est l'un des problèmes d'optimisation combinatoire, les études ont montré qu'il est pratiquement impossible d'assurer une solution optimale dans les cas complexes où le nombre de composants d'un circuit dépasse les milliers. Il s'est avéré que l'utilisation des méthodes exactes est pratiquement impossible puisqu'elles nécessitent un temps d'exécution insupportable, l'utilisation des heuristiques approximatives s'avère donc très intéressante.

Objectif

Comparaison expérimentale entre deux méthodes d'optimisation.

Organisation du mémoire

Afin de bien présenter notre travail nous avons choisi de la structure suivante :

- **Chapitre 1 : état de l'art sur l'optimisation combinatoire**

La première partie de notre travail est consacrée à l'expose de l'état de l'art des techniques d'optimisation capables de résoudre un certain de problèmes. Deux grandes classes de méthodes sont présentées : les méthodes exactes qui consistent généralement à énumérer, de manière implicite, l'ensemble des solutions de l'espace de recherche et garantissent de trouver une solution optimale, et les méthodes approchées qui traitent généralement des problèmes de grande taille, elles n'assurent pas de trouver la solution optimale mais sont efficaces. Les méthodes les plus connues et les plus utilisées vont être détaillées dans notre travail.

- **Chapitre 2 : présentation de problème de placement de composants électronique**

Nous proposons dans la deuxième parité de notre travail le problème qu'on va utiliser pour l'étude comparative ensuite nous présenterons sa formalisation.

- **Chapitre 3 : Conception de l'outil de résolution**

La partie troisième montre les méthodes de résolution choisies pour la comparaison et décrit la conception générale et la conception déraillée de notre système.

- **Chapitre 4 : Implémentation**

La dernière partie sera consacrée à la présentation de l'implémentation de notre système et les réglages de paramètres de contrôle de chaque méthode de résolution. Nous parlerons aussi de l'analyse des résultats obtenus après des séries d'expérimentations et la discussion de ces résultats.

Enfin, nous achevons ce mémoire par une conclusion générale qui exposera les perspectives envisagées.

1. Introduction

L'optimisation combinatoire est un outil indispensable combinant diverses techniques de la mathématique discrète et de l'informatique afin de résoudre des problèmes d'optimisation combinatoire de la vie réelle **[1]**.

Un problème d'optimisation combinatoire consiste à trouver la meilleure solution dans un ensemble discret de solutions appelé ensemble des solutions réalisables. En général, cet ensemble est fini mais de cardinalité très grande **[1]**.

Il s'agit, en général, de maximiser (problème de maximisation) ou de minimiser (problème de minimisation) une fonction objectif sous certaines contraintes .Le but est de trouver une solution optimal dans un temps d'exécution raisonnable.

Dans ce chapitre nous présentons certaines méthodes parmi les plus représentatives, développées pour résoudre les problèmes d'optimisation combinatoire.

2. Problème du L'optimisation combinatoire

2.1. Définition

L'optimisation combinatoire est minimiser ou maximiser une fonction souvent appelée fonction coût, d'une ou plusieurs variables soumises à des contraintes. L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire **[2]**. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données **[1]**.

2.2. Résolution d'un problème d'optimisation combinatoire

Résoudre un problème d'optimisation combinatoire nécessite l'étude de trois points particuliers :

- la définition de l'ensemble des solutions réalisables,
- l'expression de l'objectif à optimiser,
- le choix de la méthode d'optimisation à utiliser,

Les deux premiers points relèvent de la modélisation du problème, le troisième de sa résolution. Afin de définir l'ensemble des solutions réalisables, il est nécessaire d'exprimer l'ensemble des contraintes du problème. Ceci ne peut être fait qu'avec une bonne connaissance du problème sous étude et de son domaine d'application.

3. Les méthodes d'optimisation combinatoire

Les méthodes d'optimisation peuvent être réparties en deux grandes classes de méthodes pour la résolution des problèmes :

- Les méthodes exactes,
- Les méthodes approchées,

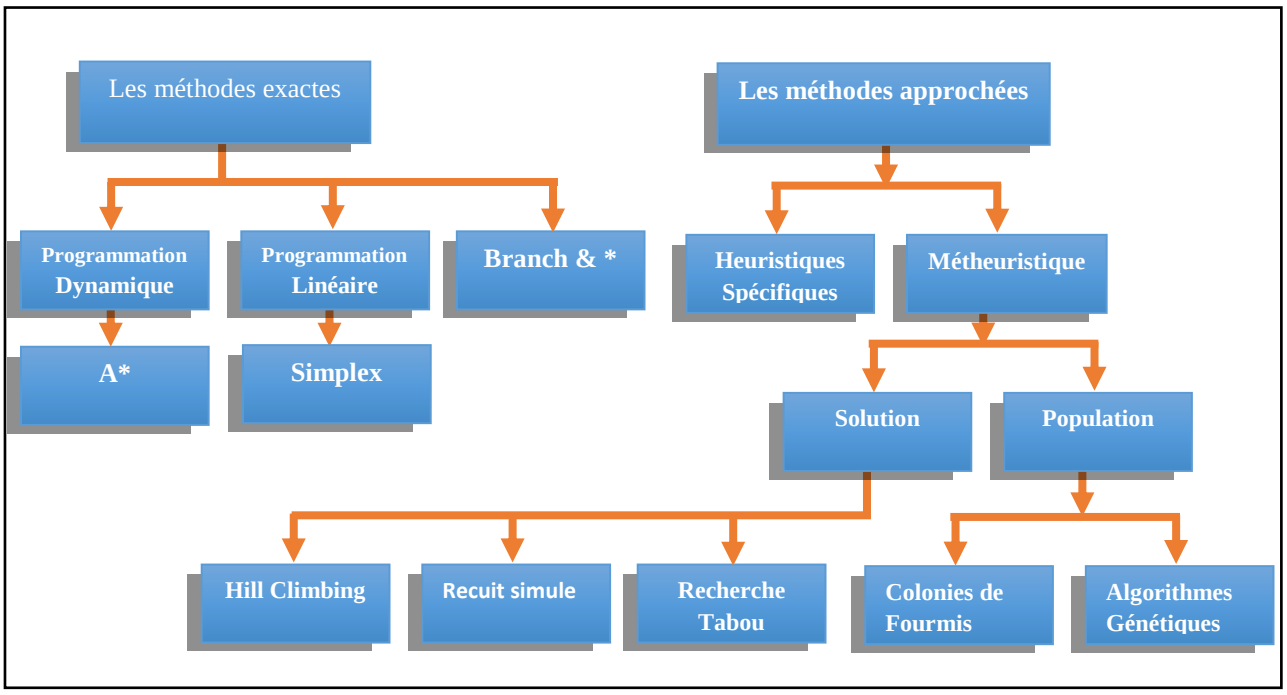

Figure 1:Classification des méthodes d'optimisation combinatoire

3.1. Les méthodes exactes

Les algorithmes exacts sont utilisés pour trouver au moins une solution optimale d'un problème **[3]**. Les algorithmes exacts les plus réussis dans la littérature appartiennent aux paradigmes de quatre grandes classes :

La programmation dynamique,

La programmation linéaire,

Les méthodes de recherche arborescente (Branch & bound),

3.1.1. La méthode de branch and bound

La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème.

Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. Bien entendu, La performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

- **Algorithme général**

Début

Placer le nœud début de longueur *0* dans une liste.

Répéter

Si la première branche contient le nœud recherché **alors**

Fin avec succès.

Sinon

- Supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.
- Calculer les coûts cumulés des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

Jusqu'à (liste vide ou nœud recherché trouvé)

Fin

3.1.2. Programmation Dynamique

La programmation dynamique a été appelée comme cela depuis 1940 par Richard Bellman et permet d'appréhender un problème de façon différente de celle que l'on pourrait imaginer au premier abord. Le concept de base est simple : une solution optimale est la somme de sous-problèmes résolus de façon optimale. Il faut donc diviser un problème donné en sous-problèmes et les résoudre un par un.

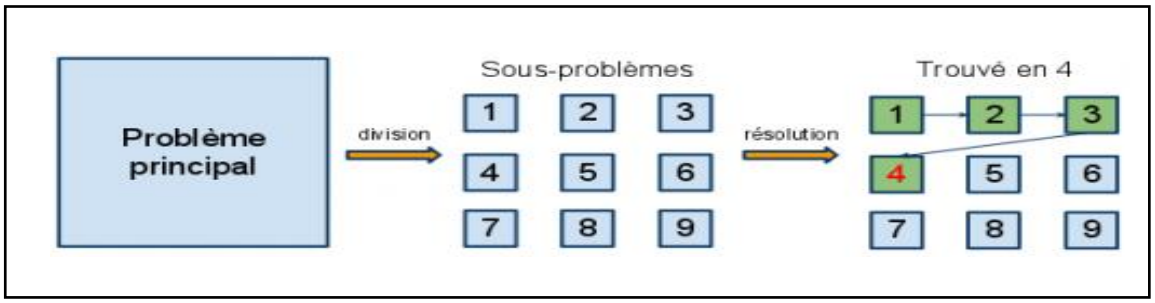

Figure 2: divisé en sous-problèmes

- **Algorithme général de programmation dynamique**

La conception d'un algorithme de programmation dynamique peut être planifiée dans une séquence de quatre étapes.

1. Caractériser la structure d'une solution optimale.
2. Définir récursivement la valeur d'une solution optimale.
3. Calculer la valeur d'une solution optimale en remontant progressivement jusqu'à l'énoncé du problème initial.
4. Construire une solution optimale pour les informations calculées.

3.2. Les méthodes approchées ou heuristique

Une méthode heuristique ou approchée est une méthode d'optimisation qui a pour but de trouver une solution réalisable de la fonction objectif en un temps raisonnable, mais sans garantie d'optimalité. L'avantage principale de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, Dun autre côté les algorithmes d'optimisation tels que les algorithmes de recuit simulé, les algorithmes tabous et les

algorithmes génétiques ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires **[4].**

Les méthodes approchées englobent deux classes :

- Les méthodes constructives,
- Les métaheuristiques,

3.2.1. Les méthodes constructives

Ce sont des méthodes itératives qui construisent pas à pas une solution. Partant d'une solution partielle initialement vide, ils cherchent à étendre à chaque étape la solution partielle de l'étape précédente, et ce processus se répète jusqu'à ce que l'on obtienne une solution complète.

- **Utilisation :** Les méthodes constructives sont généralement utilisables quand la qualité de solution n'est pas un facteur primordial ou la taille de l'instance est raisonnable, en l'occurrence pour générer une solution initiale dans une métaheuristique, ces méthodes rapides et faciles d'implémentation.

3.2.1.1. L'Algorithme glouton

Construction de une solution réalisable en ramenant à une suite de décisions qu'on prend à chaque fois au mieux en fonction d'un critère local sans remettre en question les décisions déjà prises. Généralement, la solution obtenue set approchée.

- **Avantage** : algorithmes simples à implémenter,
- **Inconvénient** : solutions approchées obtenues plus ou moins bonnes, critère local,

- **Exemple : Placement optimal de pièces 2D (Bin Packing).**

On dispose de plaques rectangulaires toutes identiques dans lesquelles on veut placer des pièces rectangulaires sans chevauchement. Les pièces à placer ont des dimensions diérèses.

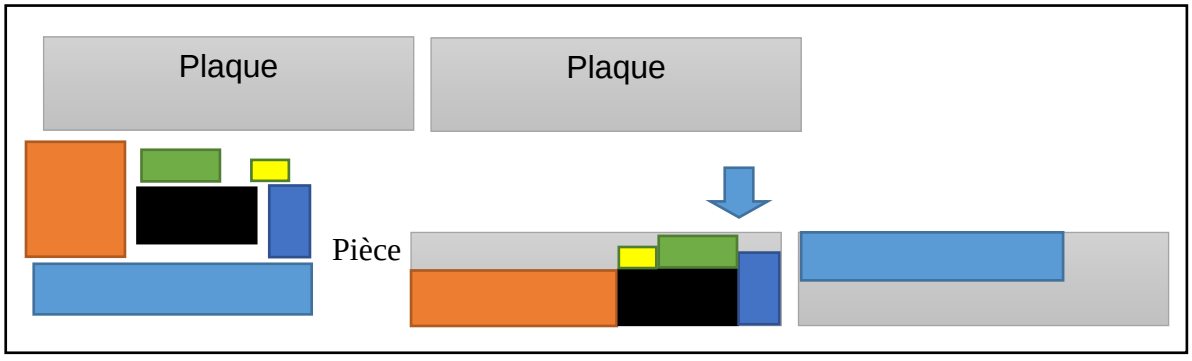

Figure 3 : Placement optimal des pièces

On veut trouver le placement pour minimiser le nombre de plaques utilisées.

Algorithme glouton : trier les pièces en fonction de leur taille et placer d'abord les pièces les plus grandes.

3.2.2. Les 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 se sont des méthodes inspirées de la nature, ce sont des heuristiques modernes dédiées à la résolution des problèmes et plus particulièrement aux problèmes d'optimisation, qui visent d'atteindre un optimum global généralement enfoui au milieu de nombreux optima locaux.

Les métaheuristiques qui se subdivisent en deux sous-classes :

- Les méthodes de voisinage.
- Les méthodes évolutives.

3.2.2.1. Les méthodes de voisinage

Ces méthodes partent d’une solution initiale (obtenue de façon exacte, ou par tirage aléatoire) et s’en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l’espace des solutions. Dans cette catégorie, se rangent :

- le recuit simulé.
- la méthode Tabou le terme de **recherche locale** est de plus en plus utilisée pour qualifier ces méthodes.

3.2.2.1.1. Le recuit simulé

La méthode du recuit simulé est une généralisation de la méthode Monte-Carlo ; son but est de trouver une solution optimale pour un problème donné. Elle a été mise au point par trois chercheurs de la société IBM : S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983, et indépendamment par V. Cerny en 1985 à partir de l'algorithme de Metropolis ; qui permet de décrire l'évolution d'un système thermodynamique **[5]**.

L’idée principale du recuit simulé tel qu’il a été proposé par Metropolis en 1953 est de simuler le comportement de la matière dans le processus du recuit très largement utilisé dans la métallurgie. Le but est d’atteindre un état d’équilibre thermodynamique, cet état d’équilibre

(où l'énergie est minimale) représente - dans la méthode du recuit simulé – la solution optimale d'un problème ; L'énergie du système sera calculé par une fonction coût (ou fonction objectif). La méthode va donc essayer de trouver la solution optimale en optimisant une fonction objectif, pour cela, un paramètre fictif de température a été ajouté par Kirkpatrick, Gelatt et Vecchi. En gros le principe consiste à générer successivement des configurations à partir d'une solution initiale S0 et d'une température initiale T0 qui diminuera tout au long du processus jusqu'à atteindre une température finale ou un état d'équilibre (optimum global).

- **Algorithme général de recuit simulé**

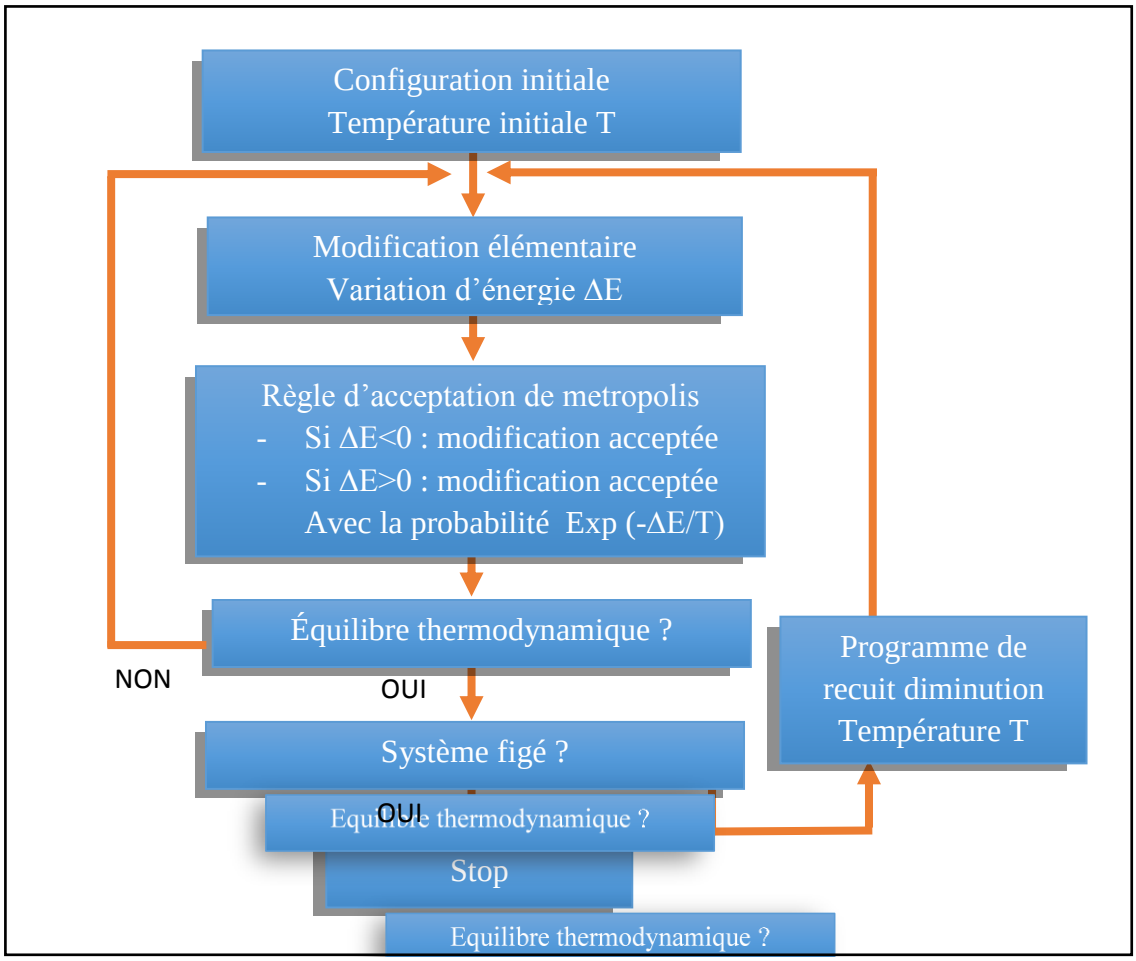

Figure 4: Organigramme de l'algorithme du recuit simulé

3.2.2.1.2. La recherche tabou

La méthode de recherche tabou, ou simplement méthode tabou, a été formalisée en 1986 par F. Glover **[6]**. Sa principale particularité tient dans la mise en œuvre de mécanismes inspirés de la mémoire humaine. L'idée consiste à garder la trace du cheminement passé dans une mémoire et de s'y référer pour guider la recherche

- **Notion de base**

À chaque itération, l'algorithme tabou choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction de coût. Pour cette raison, on dit de la recherche avec tabou qu'elle est une méthode agressive.

- **La liste tabou (mémoire)**

L'idée de base de la liste taboue consiste à mémoriser les configurations ou régions visitées et à introduire des mécanismes permettant d'interdire à la recherche de retourner trop rapidement vers ces configurations.

- **Algorithme générale**

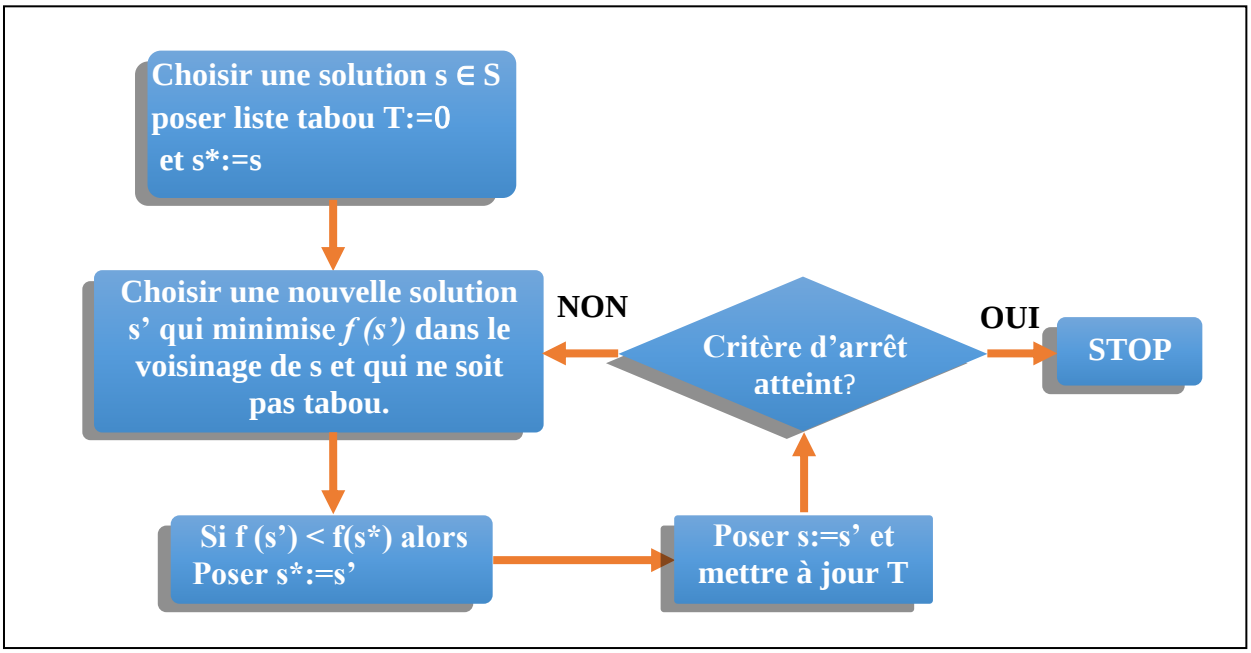

Figure 5: Organigramme de l'algorithme tabou

3.2.2.2. Les méthodes évolutives.

Ces méthodes se distinguent de celles déjà étudiées par le fait qu'elles opèrent sur une population de solutions, pour cela, elles sont souvent appelées des méthodes à base de population. Certaines d'entre elles ont des principes inspirés de la génétique et du comportement des insectes. La complexité de ces deux phénomènes biologiques a servi de modèle pour des algorithmes toujours plus sophistiqués ces vingt dernières années **[7]**. Nous avons retenu seulement par les algorithmes génétiques.

3.2.2.2.1. Les algorithmes génétiques

Les algorithmes génétiques sont inspirés de la théorie de l'évolution et des processus biologiques qui permettent à des organismes de s'adapter à leur environnement. Ils ont été inventés dans le milieu des années 60 (Holland, 1962; Rechenberg, 1965; Fogel et al, 1966) **[8]**.

La sélection naturelle, que Darwin appelle l'élément "propulseur" de l'évolution, favorise les individus d'une population qui sont le mieux adaptés à un environnement. La sélection est suivie de croisements et de mutations au niveau des individus, constitué d'un ensemble de gènes. Ainsi deux individus "parents", qui se croisent, transmettent une partie de leur patrimoine génétique à leurs descendants. L'individu enfant fait que celui-ci est plus au. Au fur et à mesure des générations son sélectionne les individus les mieux adaptés, et l'augmentation du nombre des individus bien adaptés fait évoluer la population entière **[9]**.

La mise en œuvre des algorithmes génétiques nécessite plusieurs étapes à détailler. La première est le codage d'un individu représenté par un chromosome. La seconde est le calcul de la qualité. La troisième est de définir les opérateurs de reproduction.

➢ **Algorithme générale**

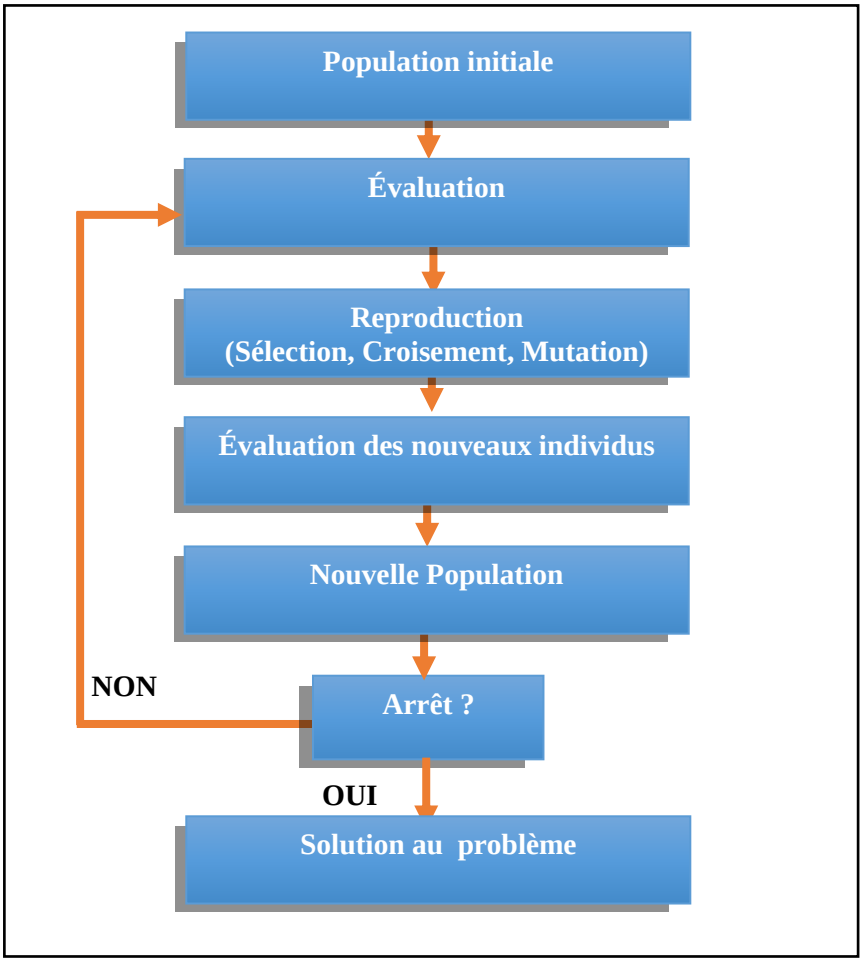

Figure 6: Fonctionnement des algorithmes génétiques.

3.2.2.2.1.1. Codage des individus

Un principe de codage de l'élément de population. Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne le succès des algorithmes génétiques. Les codages binaires ont été très utilisés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs pour l'optimisation de problèmes à variables réelles. Des exemples du codage présentés dans la (figure 7).

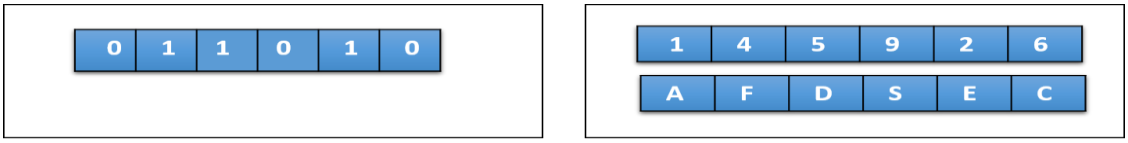

Figure 7: Codage des solutions.

Les algorithmes génétiques utilisent trois opérateurs pour générer de nouvelles solutions :

- L'opérateur de sélection qui permet de choisir des solutions parentes sur lesquelles la reproduction va être faite pour générer des nouvelles solutions.
- L'opérateur de croisement qui permet de croiser les deux solutions parentes et créer de nouvelles solutions.
- L'opérateur de mutation qui permet de diversifier les nouvelles solutions afin qu'elles ne ressemblent pas trop aux solutions parentes.

3.2.2.2.1.2. L'opérateur de sélection

La sélection consiste à choisir des individus qui permettront de générer de nouveaux individus. Plusieurs méthodes existent pour sélectionner des individus destinés à la reproduction. On citera les deux méthodes classiques les plus utilisées.

La sélection par la roulette : La sélection des individus par le système de la roulette s'inspire des roues de loterie **[10]**. À chacun des individus de la population est associé un secteur d'une roue. L'angle du secteur étant proportionnel à la qualité de l'individu qu'il représente. Vous tournez la roue et vous obtenez un individu. Les tirages des individus sont ainsi pondérés par leur qualité. Et presque logiquement, les meilleurs individus ont plus de chance d'être croisés et de participer à l'amélioration de notre population. (La figure 8) illustre une population de 5 individus dont les performances sont représentées en roulette.

individu	Fonction objectif	pourcentage
1	30	39%
2	5	7%
3	16	21%
4	10	13%
5	15	15%

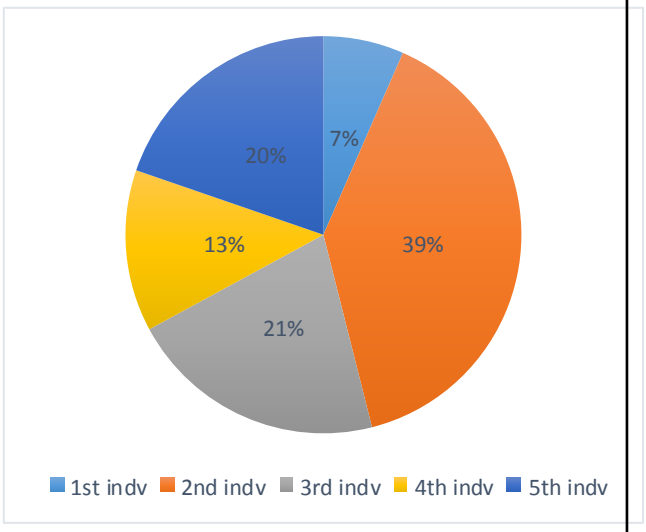

Figure 8: Sélection par roulette.

La sélection par tournoi : Le principe de la sélection par tournoi augmente les chances pour les individus de piètre qualité de participer à l'amélioration de la population. Le principe est très rapide à implémenter. Un tournoi consiste en une rencontre entre plusieurs individus pris au hasard dans la population. Le vainqueur du tournoi est l'individu de meilleure qualité. Cette méthode est en général satisfaisante (La figure 9).

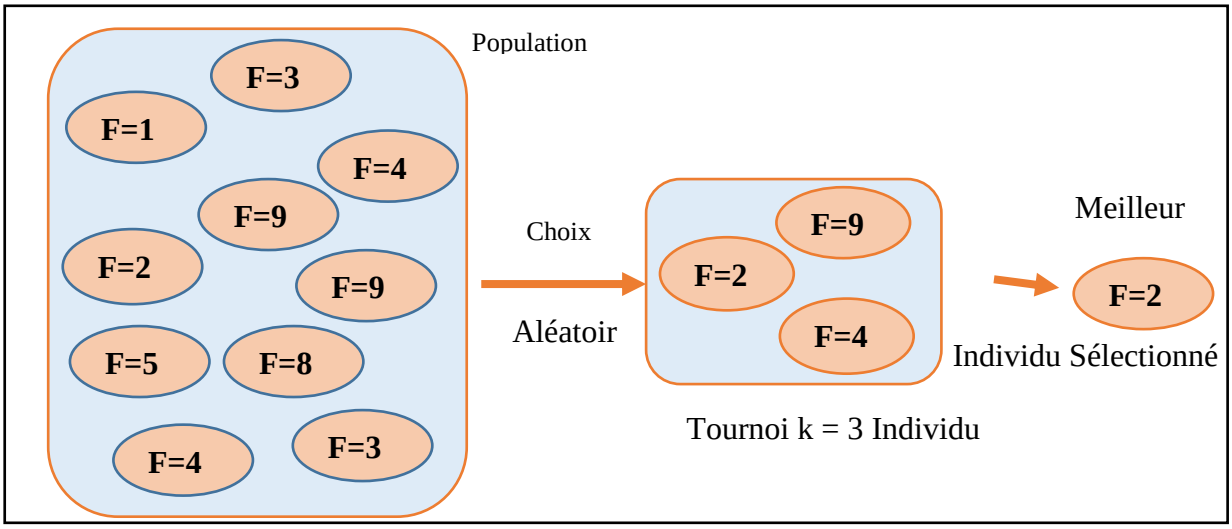

Figure 9: Sélection par tournoi.

3.2.2.2.1.3. L'opérateur de croisement:

Les croisements permettent de simuler des reproductions d'individus dans le but d'en créer des nouveaux. Il est tout à fait possible de faire des croisements aléatoires. Toutefois, une solution largement utilisée est d'effectuer des croisements multi-points.

Le croisement à un point : Il a été initialement défini pour le codage binaire. Le principe consiste à tirer aléatoire une position pour chaque parent et à échanger les sous-chaines des parents à partir des positions tirées. Ce qui donne naissance à deux nouveaux individus ind 1 et ind 2 (Figure 10).

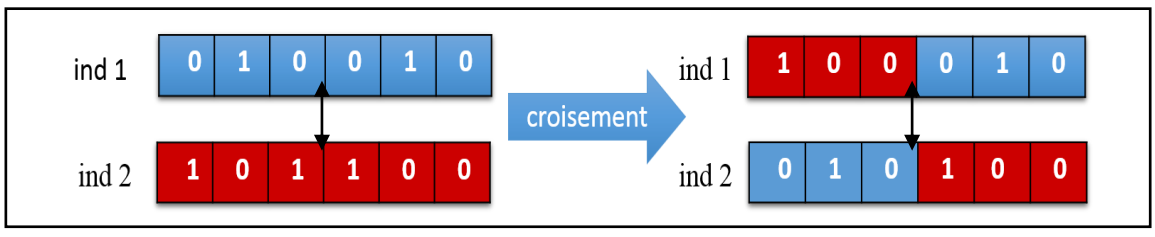

Figure 10: Croisement à un point.

Les croisements multi-points : Elle reprend le mécanisme de la méthode de croisement à un point en généralisant l'échange à 3 ou 4 sous chaines (Figure 11).

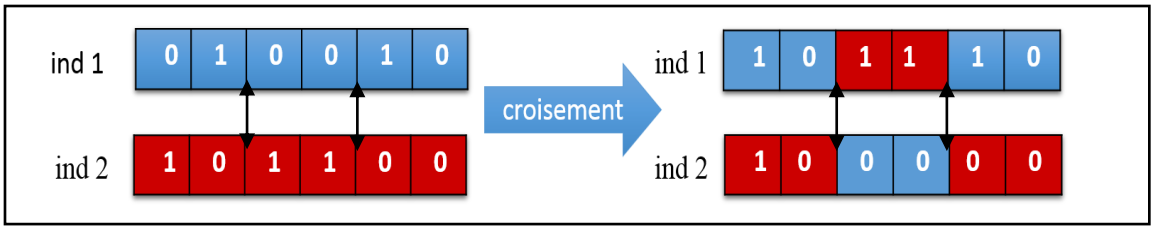

Figure 11 : Croisement à deux points.

3.2.2.2.1.4. L'opérateur de Mutation

L'opération de mutation protège les algorithmes génétiques des pertes prématurées d'informations pertinentes. Elle permet d'introduire une certaine information dans la population, qui a pu être perdue lors de l'opération de croisement. Ainsi elle participe au maintien de la diversité, utile à une bonne exploration du domaine de recherche (Figure 12).

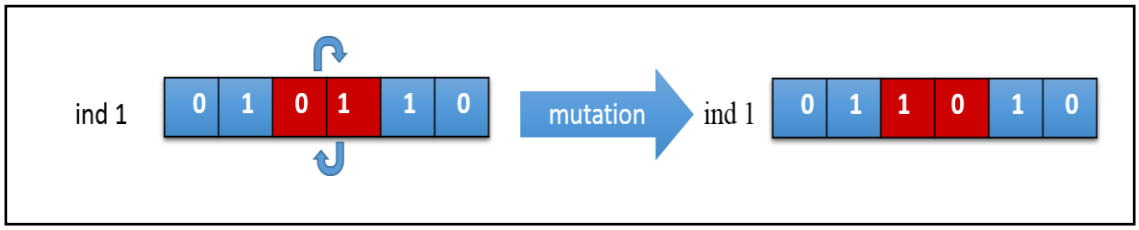

Figure 12: Une mutation.

3.2.2.2.1.5. L'opérateur de remplacement

Cet opérateur est consisté à réintroduire les descendants obtenus par application successive des opérateurs de sélection, de croisement et de mutation (la population P') dans la population de leurs parents (la population P). On trouve des méthodes de remplacement différentes par exemple méthode de remplacement stationnaire **[11]**.

Le remplacement stationnaire : dans ce cas, les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives, et le nombre d'individus de la population ne varie pas tout au long du cycle d'évolution simulé, ce qui implique donc d'initialiser la population initiale avec un nombre suffisant d'individus. Cette méthode peut être mise en œuvre de 2 façons différentes :

- La première se contente de remplacer la totalité de la population P par la population P', cette méthode est connue sous le nom de remplacement générationnel**.**

- La deuxième méthode consiste on remplace les mauvaise individus trouvées dans la population P par les meilleurs de la population P'.

4. Conclusion

Nous avons présenté au cours de ce chapitre plusieurs méthodes de résolution heuristiques, Il est nécessaire de faire appel à des heuristiques permettant de trouver de bonnes solutions appro

chées. Donc pour résoudre un problème on doit choisir les méthodes adéquates qui peuvent être adapté au type du problème. Après la présentation du problème qu'on va traiter dans le chapitre suivant on va choisir certaines méthodes adéquates vus dans ce chapitre pour la résolution de notre problème.

1. Introduction

Le problème de placement de composants électronique dans un circuit est l'un des principaux problèmes rencontres dans la technologie d'intégration à très grande échelle VLSI (Very-Large-Scale-Intégration),il est connue pour être un problème réellement difficile et qui a été de manière assez détaillée depuis la création des tout premiers circuit intégré à forte densité d'intégration ,notamment les microprocesseurs .

La signification de ce problème est principalement due à la difficulté de placer les composant électroniques d'un circuit de façon à minimiser la longueur totale des fils reliant les composants entre eux Dans ce chapitre , on va essayer de faire une étude de ce problème dans sa globalité , afin de bien adapter les méthodes de résolution à ce problème dans les chapitre à venir .

2. La technologie VLSI

L'intégration à très grande échelle *(*VLSI - Very-Large-Scale Integration) est une technologie de circuit intégré dont la densité d'intégration permet de supporter plus de 100 000 composants électroniques sur une même puce, aujourd'hui plusieurs dizaines de millions de ports logiques représentent un chiffre normal pour un microprocesseur comme étant un dispositif VLSI.

Le ralentissement du degré d'intégration est principalement dû à la difficulté de dissipation thermique ainsi que les effets de bruits parasites dans les circuits intégrés.

L'augmentation incessante du degré d'intégration pousse les concepteurs de circuits VLSI à modifier leur méthode de conception. La multitude des problèmes à traiter oblige les concepteurs à considérer chaque traitement algorithmique (placement, routage…) indépendamment les uns des autres.

La conception d'un VLSI a été organisée en plusieurs activités de conception, chaque activité est elle-même une problématique avec son propre traitement algorithmique indépendant. Parmi ces problèmes on trouve le problème de packing, routing, via-minimisation, et de placement.

3. Différents problèmes de conception VLSI

On va voir quelques problèmes de conception qu'on juge les plus importants :

3.1. Problème de packing

Dans le problème de packing, il s'agit de trouver le placement le plus économique possible pour un ensemble de composants électroniques, de tailles et de formes différentes, dans un rectangle. Il faut aussi respecter les contraintes d'espacement entre les composants pour laisser l'espace au fils conducteurs **[12]**.

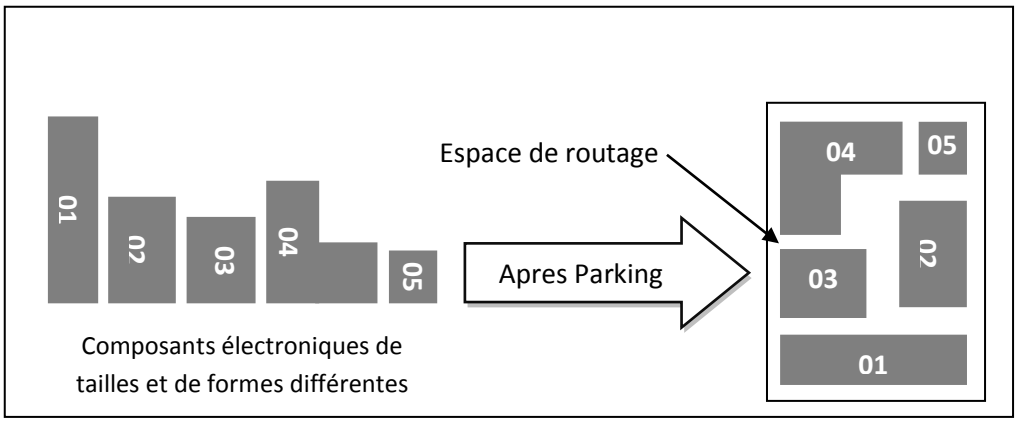

Figure 13: Exemple de packing

3.2. Problème de routing

Dans ce problème aussi connue sous le nom de *Wiring Problem*, il s'agit de définir formellement les chemins précis des fils conducteurs nécessaires pour l'interconnexion des éléments du circuit. Les contraintes imposées sont généralement le diamètre des fils et les positions des vias (trou percé à travers l'épaisseur de la carte) **[13]**.

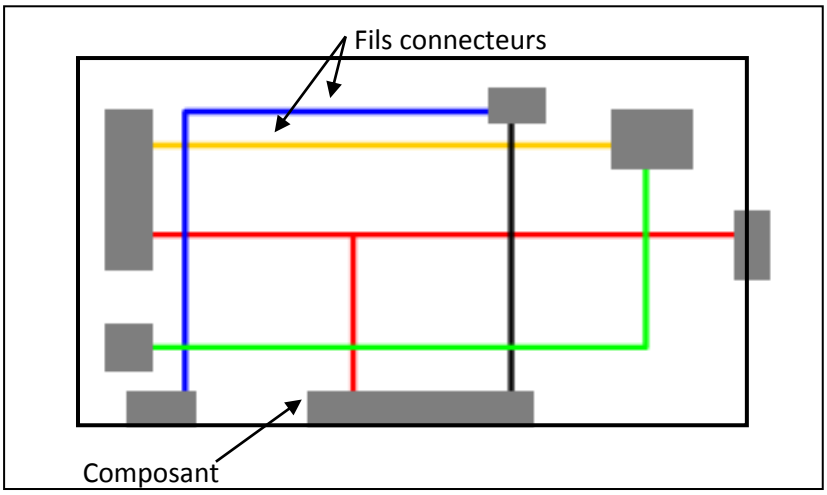

Figure 14: Exemple de routing

3.3. Problème de Via-Minimization

Ce problème intervient à la fin du processus de conception, après le placement des composants et le routage des fils, il y aura des croisements entre eux. Le problème de Via Minimization s'agit donc d'éviter ces croisements, la procédure consiste à découper les fils, les placer sur la face supérieure ou inférieure de la carte et les reliés par des vias de façon à éviter les croisements de fils **[13]**.

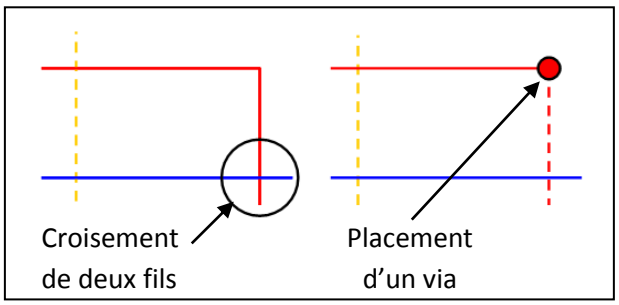

Figure 15: Exemple de placement de via

Mais il faut réaliser ceci tout en plaçant un nombre minimum de vias pour minimiser les problèmes de fiabilité du circuit.

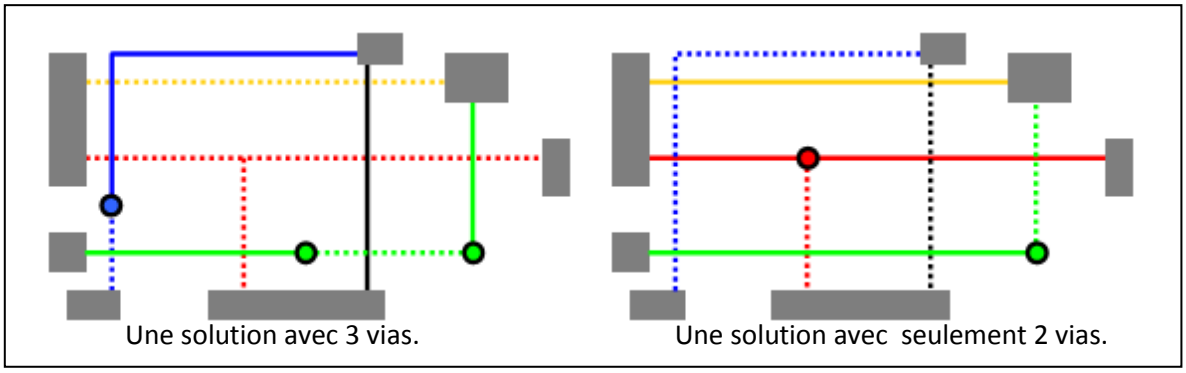

Figure 16 : Exemple sur deux solutions de Via Minimization.

4. Problème de placement de composant électronique

Il est aussi nommé problème de Connection-cost Dans ce problème on doit positionner les composants électroniques dans une grille de positions prédéterminées de façon à minimiser la quantité de fils nécessaire pour effectuer l'interconnexion des composants du circuit **(Figure 17)**, Ce type de problème est considéré comme un problème d'optimisation combinatoire, et ce qui concerne leur complexité, il a été prouvé par (Sahni 1980, Donath 1980, Leighton 1983) que le problème de placement est NP-difficile, donc il ne peut être résolu de manière exacte avec un temps d'exécution polynomial **[14].**

C'est un problème qui a beaucoup de solutions mais qui reste ouvert, parmi ces objectifs :

- la minimisation des longueurs de connexions.
- la minimisation de la chaleur dissipée par les composants électroniques.
- La minimisation de temps de calcul.
- Réduire le coût d'interconnexions entre les composants.
- Garantir la « *routabilité* ».
- Atteindre la plus grande densité possible.
- Minimiser la consommation.

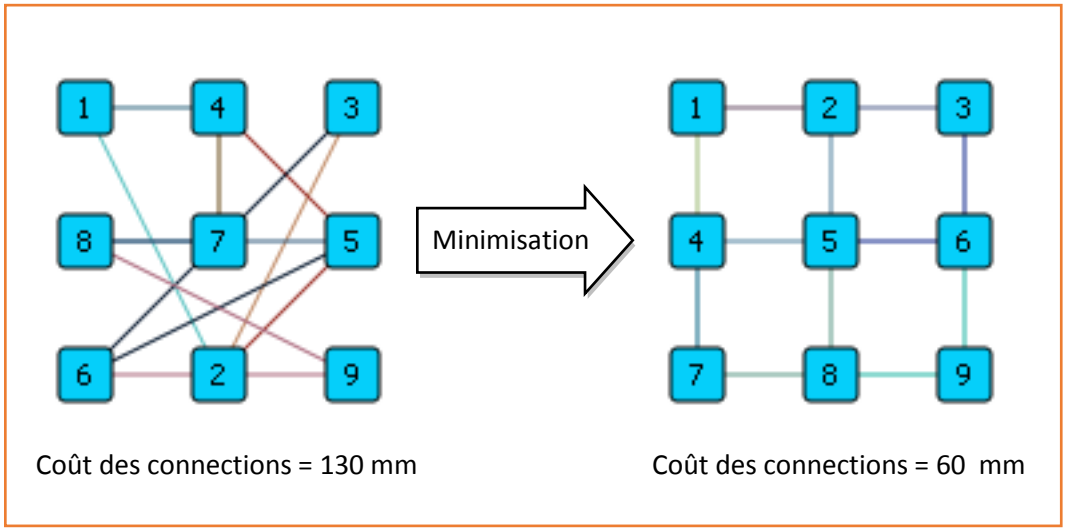

Figure 17: Exemple de placement avec minimisation du coût

5. Formalisation du problème de placement

Pour résoudre le problème de placement de composants électroniques, on doit le représenter de manière formelle, pour cela on utilise un graphe $G = \{E_G, V_G\}$ comme modèle. G est un graphe non orienté dans lequel E_G est l'ensemble des nœuds qui représentent les composants électroniques, et V_G l'ensemble des arcs qui représentent les fils connecteurs. Les nœuds du graphe sont numérotés de 1 à N, où N est le nombre de composants électroniques du circuit, chaque nœud de E_G est une paire (x, y), $x \in R$, $y \in R$, qui désigne la position du composant électronique dans le circuit. Et chaque arc de V_G est une paire (i, j), $i \in \{1, \ldots, N\}$, $j \in \{1, \ldots, N\}$ qui désigne les deux composants reliés par un fil connecteur.

6. Estimation de la longueur des fils (Distance de Manhattan)

La vitesse d'estimation à un effet considérable sur la performance de l'algorithme de placement, une bonne technique d'estimation est d'une importance totale pour tout algorithme de placement, l'estimation doit être la plus rapide possible. En réalité l'estimation des distances de fils dans les circuits électroniques utilise la géométrie Manhattan (figure 18), les ils sont horizontaux ou verticaux.

À chaque composant *i* on associe un couple de coordonnées(x_i, y_i), la distance Manhattan est calculée comme suite : $d\ (u_1, u_2) = |x_1 - x_2| + |y_1 - y_2|$

Où u_1 et u_2 sont deux composants électroniques.

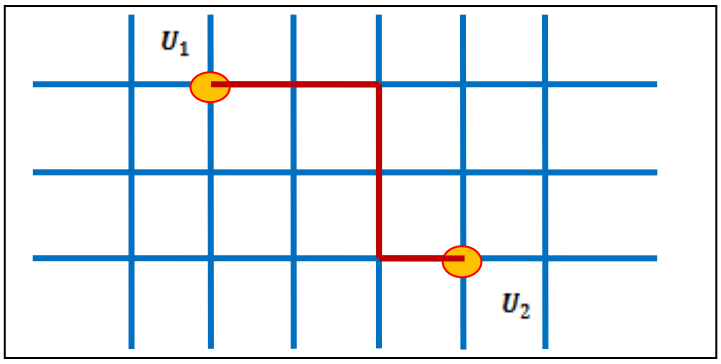

Figure 18 : Calcul de distance Manhattan.

7. Résoudre le problème de placement de composant électronique

Techniquement, dans ce problème on doit positionner les composants électroniques dans une grille de positions prédéterminées de façon à minimiser la quantité de fils nécessaires pour effectuer l'interconnexion des composants du circuit.

Généralement, pour avoir une bonne solution de placement avec un résultat satisfaisant, il faut trouver une méthode pour optimiser la solution de placement à partir d'une solution suffisante.

8. Conclusion

L'objectif n'est pas d'obtenir un optimum absolu, mais seulement une bonne solution. Pour atteindre cet objectif au bout d'un temps de calcul raisonnable, il est nécessaire de faire appel à des heuristiques permettant de trouver de bonnes solutions.

Dans le chapitre suivant, on va concevoir un système de résolution du problème de placement en utilisant les deux méthodes de recherche local étudiées dans le deuxième chapitre.

1. Introduction

Après avoir présenté le problème de placement de composants électroniques et les différentes méthodes de résolution heuristique dans les chapitres précédents, Nous arrivons dans le présent chapitre à la proposition de notre conception, qui est la proposition d'une modélisation de notre problème, en utilisant les méthodes de résolutions de problème les algorithmes génétiques et recuit simulé.

Nous commençons la description de notre conception par la présentation de l'objectif de notre travail. Ensuite, nous présenterons la description de la conception proposée et les couches internes, ainsi que leurs rôles, nous présentons aussi l'adaptation des méthodes de résolution du problème de placement. Enfin, la conclusion

2. Conception générale du système

On a essayé de concevoir un système qui représente l'outil de résolution du problème de placement de composants électroniques une architecture à trois couches représentée dans la figure suivante :

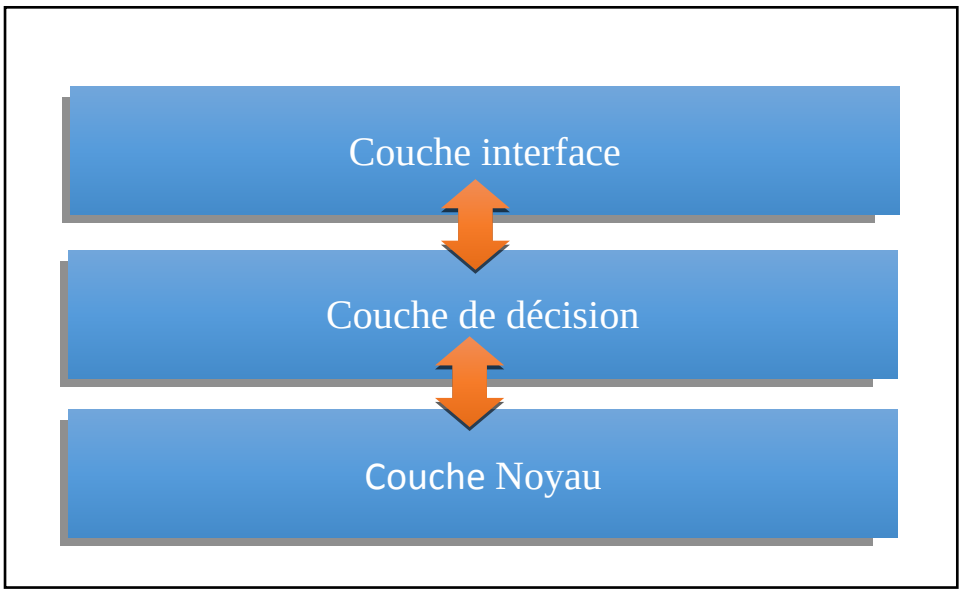

Figure 19 : Conception en trois couches.

3. Conception globale

Notre architecture est composée de 3 couches qui sont : Couche Interface, Couche de décision et couche noyau. On va détailler chaque couche de système dans l'architecture générale.

3.1. Couche interface

C'est l'IHM ou bien interface homme machine c'est la partie de visible l'application et interactive avec les utilisateurs. En informatique, à travers l'interface on choisit la méthode de résolution ainsi que ses paramètres de même on choisit le type de génération de circuits d'un coté et d'un autre coté on affiche les résultats obtenus après a résolution du problème

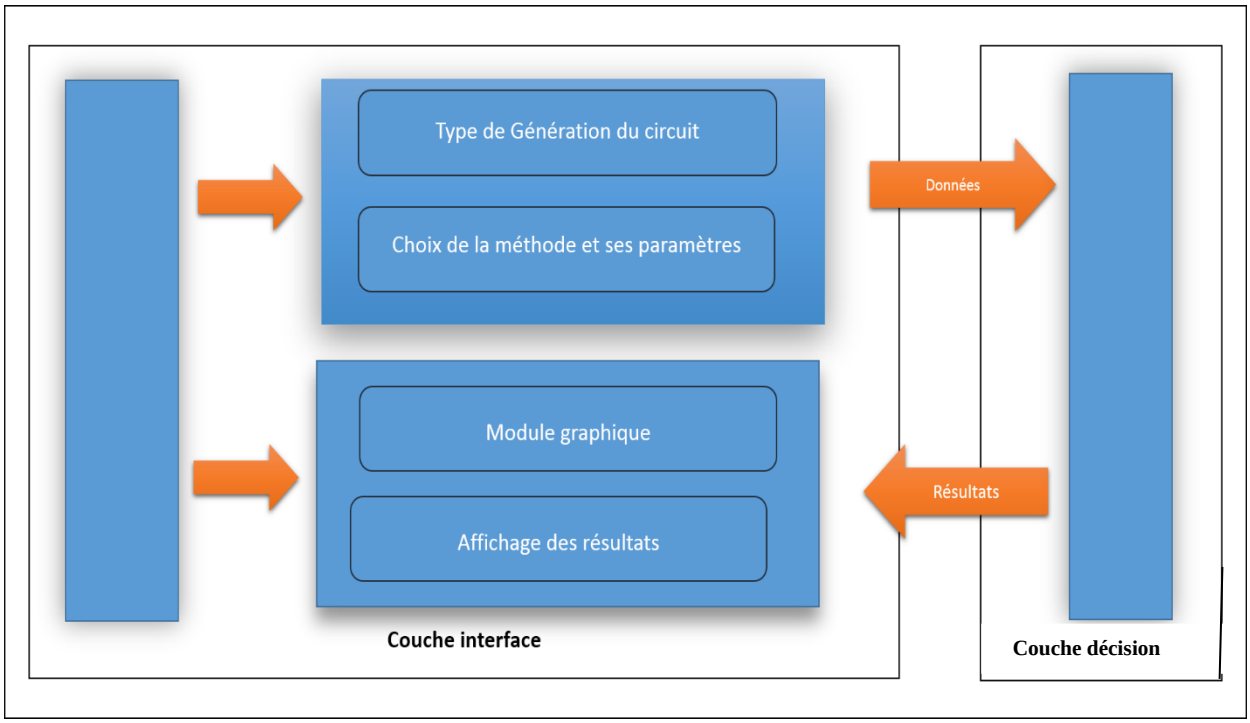

Figure 20 : Couche interface

À partir cette couche fournit les informations d'entrées vers la couche décision, elle contient deux modules :

3.1.1. Type de génération du circuit

- Génération aléatoire : L'utilisateur choisit cette génération Pour générer la taille de circuit qui se fait aléatoirement.
- Générer à partir d'un fichier : elle permet à l'utilisateur de choisir un fichier contenant les données d'un circuit définit.

3.1.2. Choix de la méthode et ses paramètres

- L'utilisateur doit choisir une méthode parmi la liste des méthodes et définir ses paramètres.

Après la résolution du problème, on va extraire quatre modules pour l'affichage, ce dernier est :

3.1.3. Affichage des résultats

Ce module reçoit les résultats (le temps d'exécution, le coût total de la solution trouvée) obtenus de la couche décision et les affichent sous forme numérique.

3.1.4. Module graphique

Ce module nous permet de représenter graphiquement le circuit électronique en affichant chaque composant à sa position ainsi que les fils qui les relient. Et afficher la courbe de la fonction de coût et du temps de calcule. En outre on afficher un tableau qui contient les valeurs de fitness de chaque itération.

3.2. Couche de décision

Cette couche, c'est une couche de décision entre la couche interface et la couche noyau

- Réception et standardisation des informations en provenance de la couche interface
- Transmission des informations à la couche noyau sous forme de structure de données.
- Réception les résultats puis transmission à la couche interface pour la l'affichage.

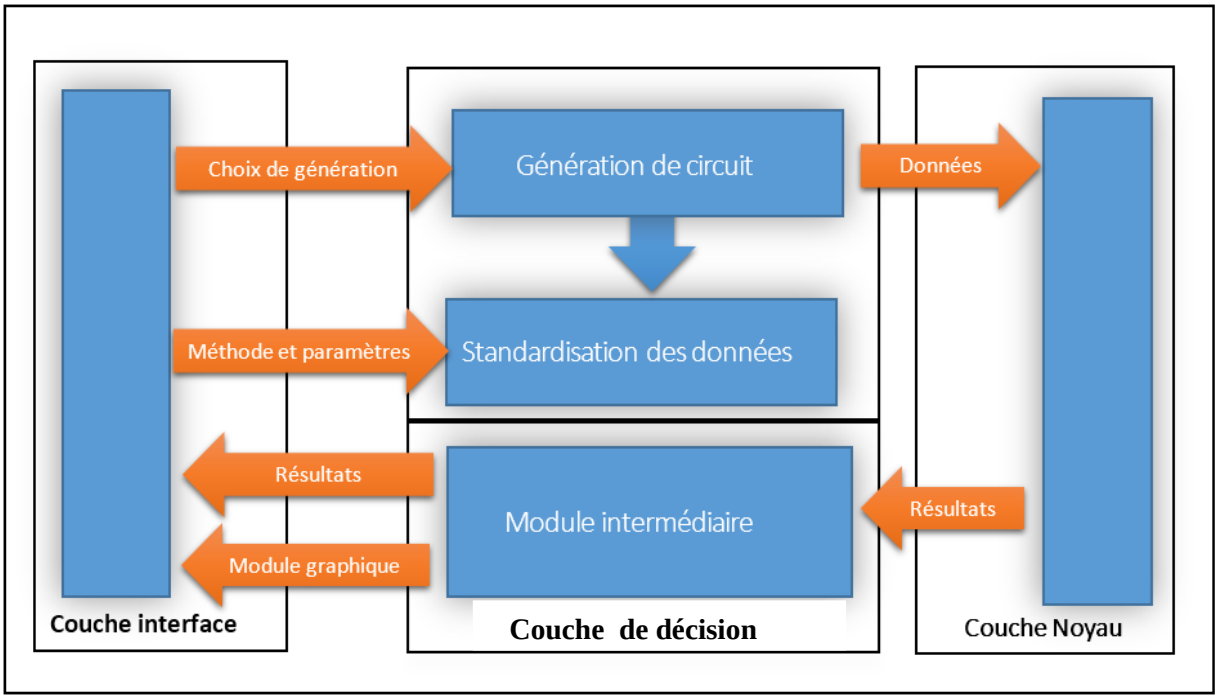

Figure 21 : Couche décision

La couche de décision contient trois modules :

3.2.1. Module de génération de circuit

Dans ce module le circuit est représenté en un graphe non orienté ou chaque nœud est un composant électronique du circuit et il est relié à un ou plusieurs autres composants (par des arcs non orientés) formant un circuit.

La génération de ce circuit peut être de deux manières :

- La génération de ce circuit c'est une génération aléatoire de la position des composants et les fils entre les composants du circuit dont la taille (nombre de composants) doit être spécifiée par l'utilisateur.
- Une génération à partir d'un fichier qui contient la taille du circuit, les liaisons de fils entre les composants ainsi que leurs identificateurs.

Ce module reçoit le choix de la manière de génération depuis la couche supérieure, et transmet les données du circuit généré au module de standardisation des données.

3.2.2. Module standardisation des données

Ce module crée la structure de données (position x, y des composants) depuis les données du circuit provenant du module de génération pour qu'elles soient standardisées et envoyées à la couche inférieure, ainsi que les informations sur le choix et les paramètres de la méthode en provenance de la couche supérieure.

3.2.3. Module intermédiaire

Ce module reçoit les résultats de calcul (temps de calcul et distance minimal calculée) depuis la couche inférieure et les envoient à la couche supérieure pour qu'elles soient affichées, il fait aussi appel au module graphique de la couche supérieure pour afficher le nouveau circuit après le calcul de solution.

3.3. Couche noyau

L'importance de cette couche est primordiale dans notre outil, elle représente l'implémentation des méthodes de résolution. Son rôle est d'exploiter les structures de données et paramètres en provenance de la couche échange pour résoudre le problème avec une méthode de résolution et de renvoyer les résultats obtenus.

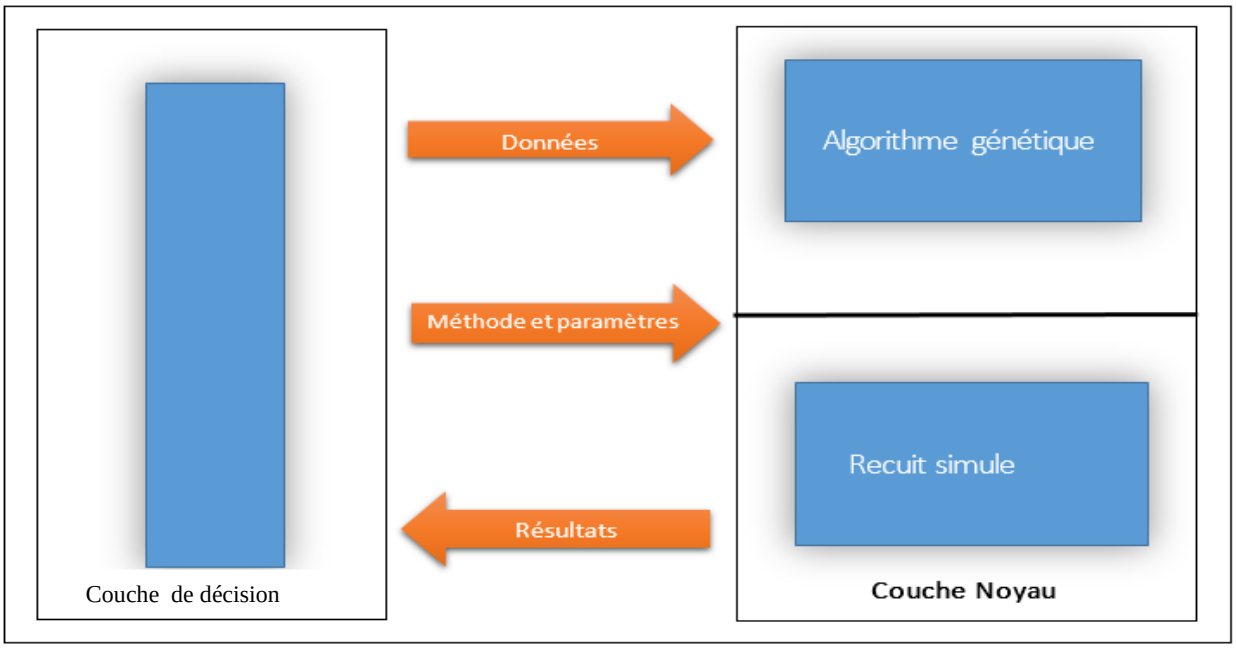

Figure 22 : Couche décision

3.3.1. Modélisation du problème de placement de composants électroniques

Pour résoudre notre résolution du problème de placement de composants électroniques on a choisi deux méthodes résolution des problèmes :

- L'algorithme génétique
- Le recuit simulé

La première méthode sont les algorithmes génétiques qui sont beaucoup utilisés pour les problèmes d'optimisations, ils permettent une recherche globale sur l'espace de recherche. Il utilise une approche qui part d'un ensemble de solutions qui sont diversifiées afin de rapprocher de la solution optimale.

Deuxième méthodes c'est le recuit simulé il permet une recherche locale sur l'espace de recherche, il est de plus utilisé pour résoudre notre problème.
De cette façon on assure une comparaison équitable entre les deux méthodes.

3.3.1.1. La solution

C'est le positionnement de tous les composants (pièces) du circuit où chaque composant à ses propres coordonnées (x, y). La figure suivante schématise un exemple de positionnement :

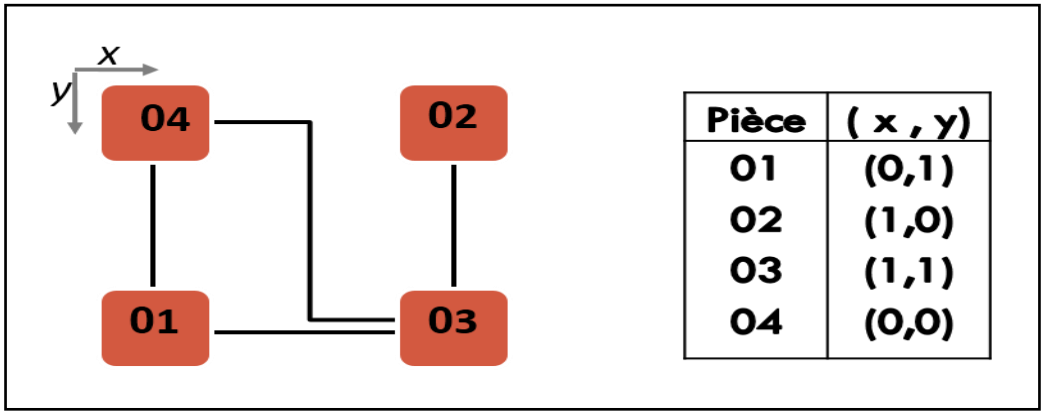

Figure 23 : Exemple de position

3.3.1.2. Le voisinage

Le voisinage représente l'ensemble de modifications (transformations) possibles appliquées à une solution pour obtenir des nouvelles solutions, dans notre cas c'est la permutation des positions de deux composants. La figure suivante schématise un exemple de voisinage :

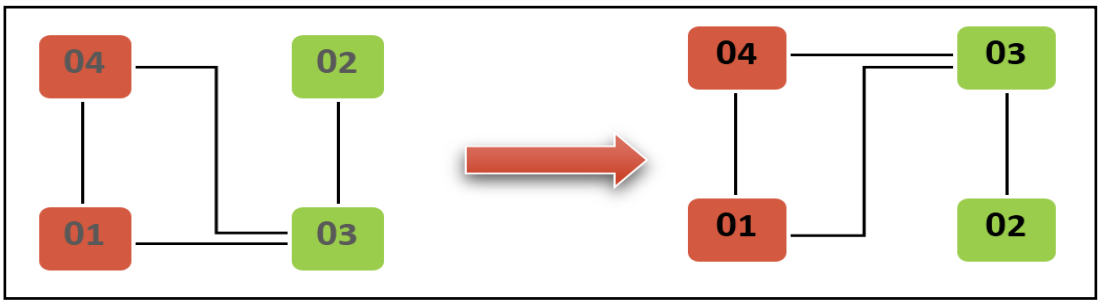

Figure 24 : Exemple de permutation.

3.3.1.3. Fonction de coût

Dans ce problème, la fonction de coût à optimiser *f(s)* est la somme de la longueur de tous les fils qui relient les composants du circuit entre eux. La longueur en étant calculée comme longueur de Manhattan vue dans le chapitre 2.

Soit deux composants C1 et C2 reliés entre eux,

La distance : $d(C1, C2) = |x1 - x2| + |y1 - y2|$

x1, y1 sont les cordonnées de placement pour le composant C1 ;

x2, y2 sont les cordonnées de placement pour le composant C2;

3.3.2. L'algorithme génétique

3.3.2.1. Le mécanisme d'algorithme génétique

L'idée de base des algorithmes génétiques est de suivre la loi de l'évolution naturelle. Chaque solution est représentée comme un individu et l'ensemble de solutions, comme une population. À chaque itération des opérateurs génétiques (sélection, croisement, mutation) sont utilisés pour diversifier la population.

Donc, concernant notre problème d'optimisation, nous tenterons de générer à partir d'un ensemble solutions (placements), une solution adéquate à l'utilisateur.

Nous distinguons une phase de création de la population initiale, suivie de la perturbation des éléments de cette population par l'application de différents opérateurs génétiques, et enfin une phase d'évaluation de cette population.

Ces mécanismes se répètent pendant un certain nombre de générations. Chaque génération est supposée contenir des éléments plus performants que ceux de la génération précédente. Intuitivement, plus le nombre de générations est grand plus la solution se raffinera et nous espérons ainsi obtenir une bonne solution, mais pas forcément la meilleure.

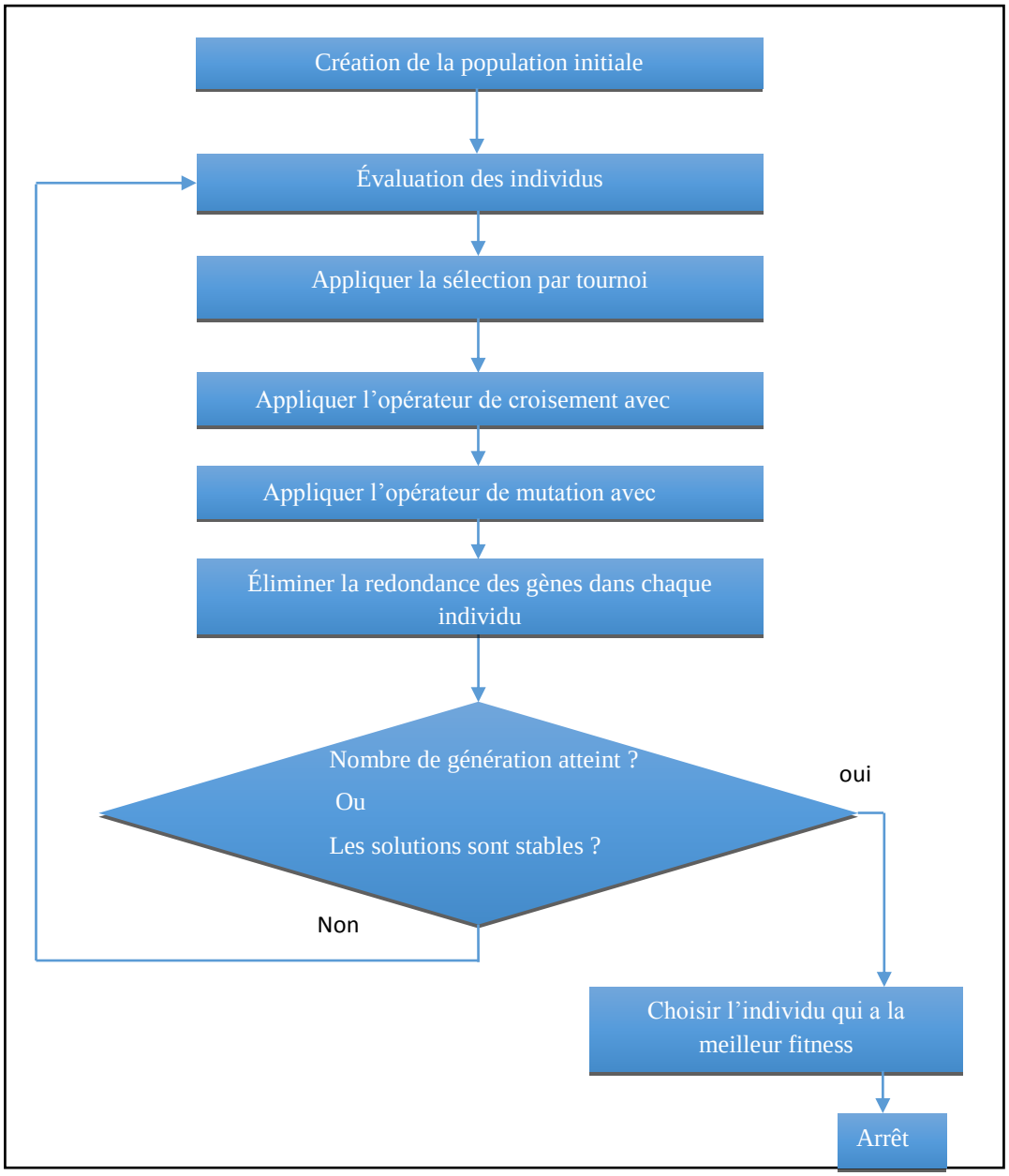

Figure 25 : Organigramme de l'algorithme génétique

3.3.2.2. Individu ou solution

C'est le positionnement de tous les composants (pièces) du circuit où chaque composant à ses propres coordonnées (x, y). Notre idée est de représenter la solution sous forme d'un vecteur de taille n (n représente le nombre de composants à placer), dont chaque élément de vecteur signifie le placement d'une pièce électronique dans le circuit. La figure 26 illustre un exemple de codage d'une solution (placement).

3.3.2.3. Codage des individus

Nous avons retenu seulement sur le codage par valeur, où chaque individu représente les caractéristiques d'un tel placement de composants électroniques. Les éléments de la chaine représentant un individu sont des entiers. Ces derniers correspondent aux différentes pièces électroniques qui sont numérotés de 1 à n, n est le nombre de composants électroniques dans le circuit.

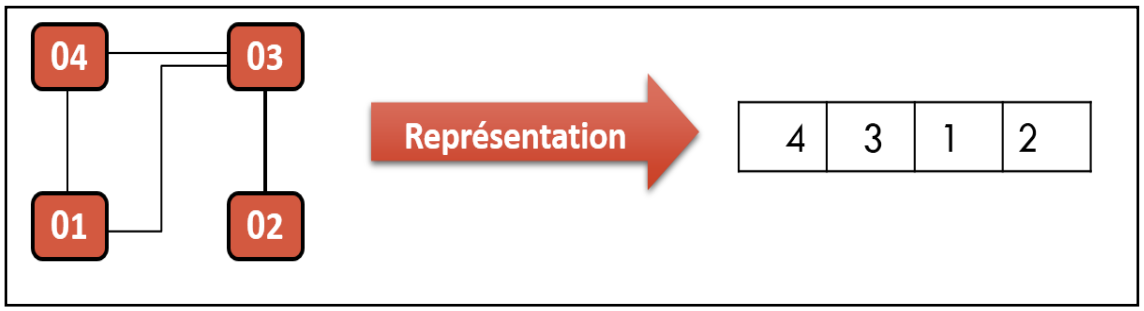

Figure 26 : Codage d'une solution.

3.3.2.4. Population initiale

La génération de solutions se fait aléatoirement où chaque élément de tableau, Ça peut être un entier quelconque entre 1 et un maximum prédéfini par le nombre de composants existe dans le circuit. À condition que les valeurs de tableau ne doivent pas être répétées.

3.3.2.5. Reproduction et évaluation

À chaque itération de l'algorithme, la population est reproduite afin diversifier les solutions. Trois opérateurs génétiques sont utilisés: la sélection, le croisement et la mutation.

3.3.2.5.1. L'opérateur de sélection

Sert à choisir deux solutions qui vont être croisées pour créer de nouvelles solutions. La sélection choisie pour cet algorithme est celle par tournoi (figure 27). Elle a été décrite dans le chapitre 1. L'avantage de cette méthode de sélection est qu'elle permet de choisir n'importe quel individu sans le risque de négliger les mauvaises solutions. Ce qui est souhaitable puisque la reproduction avec de mauvaises solutions peut guider vers des solutions meilleures.

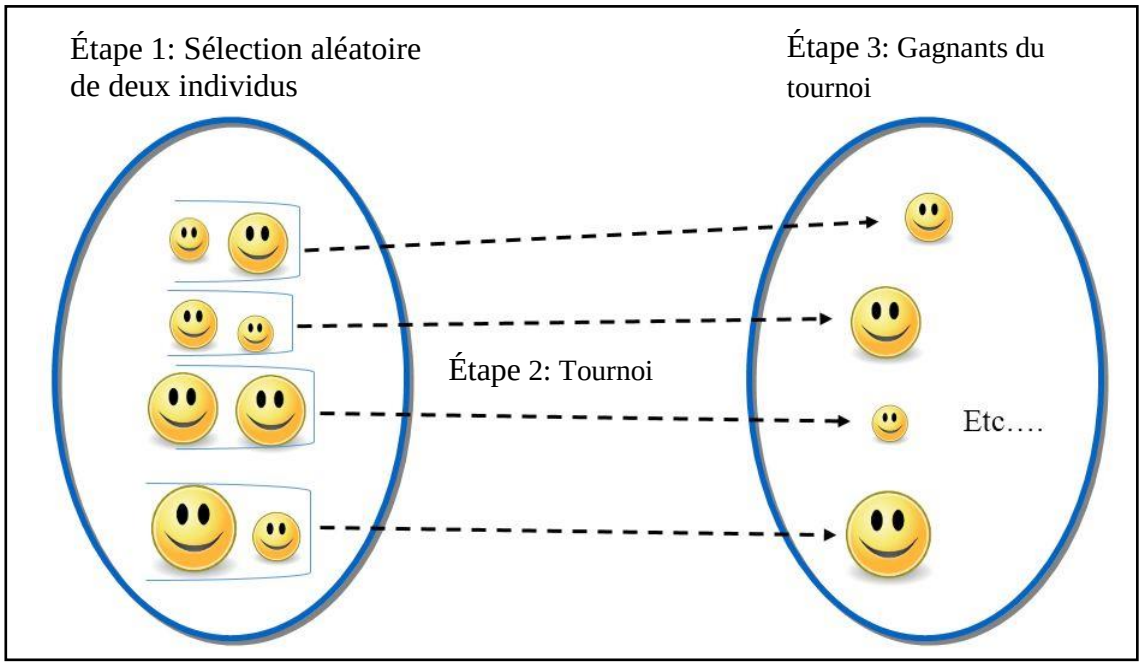

Figure 27 : Représentation d'une sélection de tournoi

3.3.2.5.2. L'opérateur de croisement :

Pour chaque couple d'individus sélectionnés par la stratégie de tournoi, on applique un point de coupure aléatoire sur la représentation des deux individus et on génère les deux enfants. À prendre en compte par l'opérateur de croisement:

- **Héritabilité :** c'est la caractéristique principale du croisement. Les enfants doivent hérités les propriétés génétiques de leurs parents.
- **Validité :** les enfants (individus générés) doivent appartiennent à l'espace de recherche.
- **Le croisement est probabiliste :** (P_c) Probabilité de croiser deux parents : P_c Appartient à [0, 1]

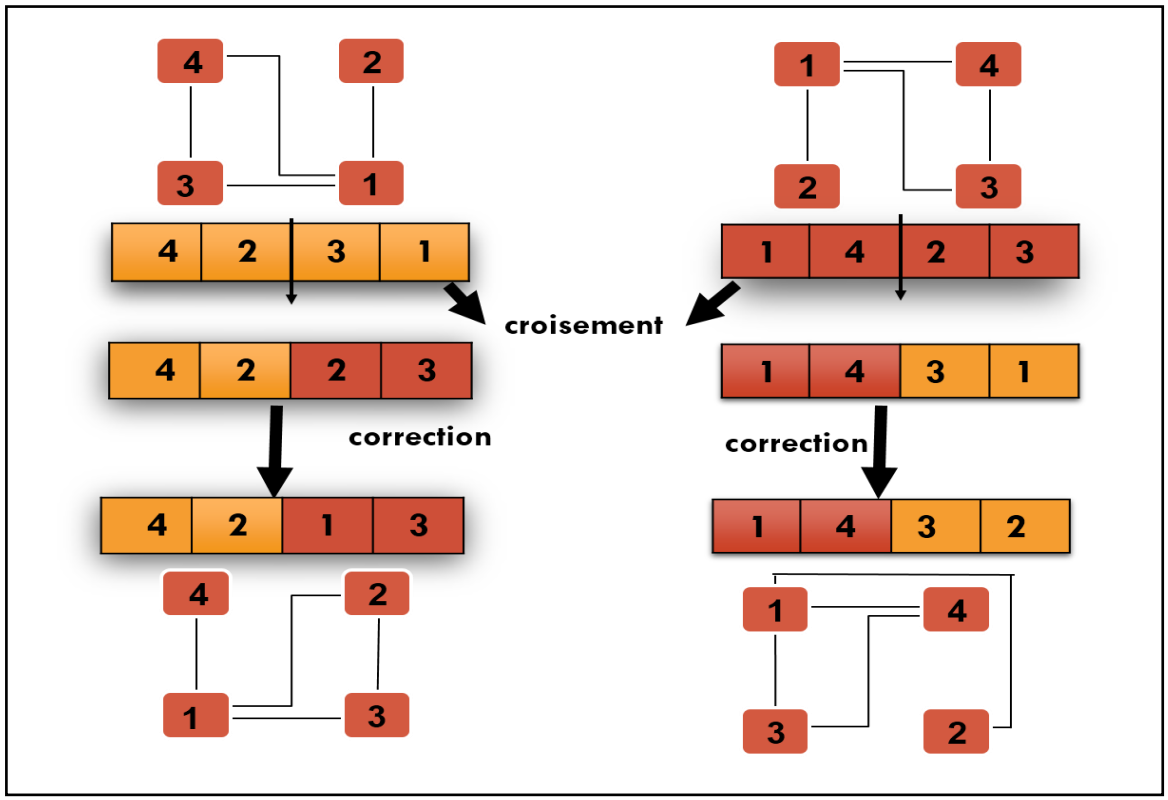

Figure 28: Représentation de croisement de deux individus.

3.3.2.5.3. L'opérateur de mutation

La mutation est considérer comme un effet (petit changement) sur un individu sélectionné de la population (figure 29).

À prendre en compte par l'opérateur de mutation:

- **Localité :** un changement minimal et local.
- **Validité :** produire des solutions valides.
- **La mutation est probabiliste :** (p_m) probabilité de muter chaque élément de la représentation.

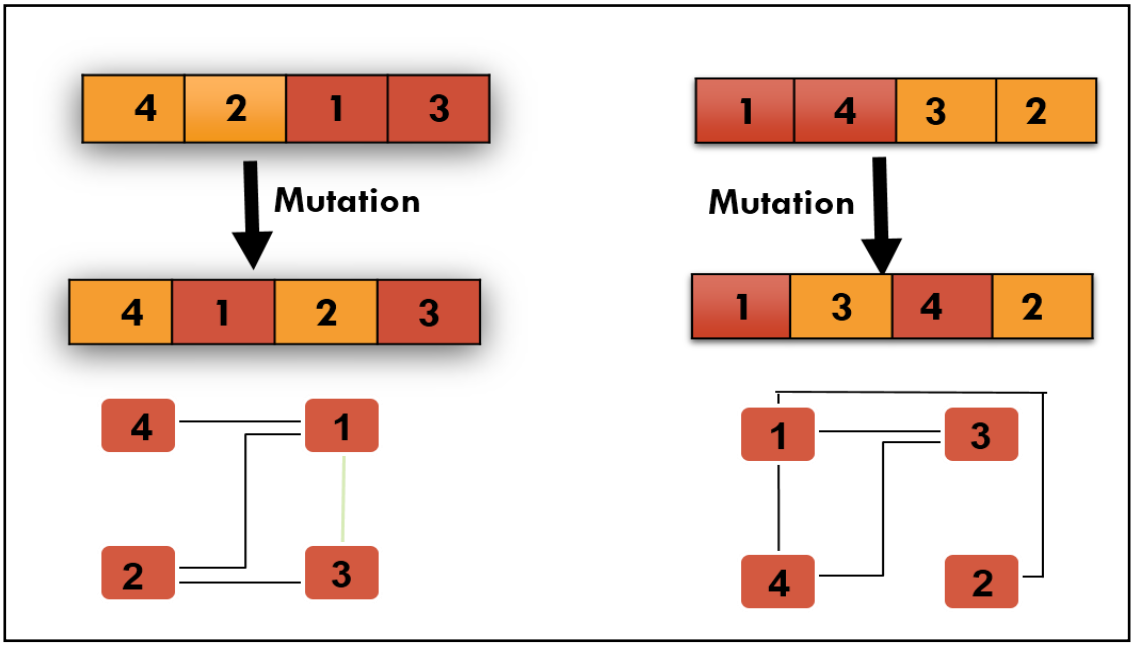

Figure 29: Exemple de mutation.

3.3.2.5.4. L'opérateur de remplacement :

C'est la sélection de survivants entre les populations des parents et des enfants. À chaque génération, on remplace les mauvaise individus trouvées dans la population parent par les meilleurs de la population enfant.

3.3.2.6. Le critère d'arrêt :

Notre algorithme génétique s'arrête dans l'un des deux cas suivants :

Le premier cas : Si le nombre de génération atteint le nombre défini préalablement.

3.3.3. Le recuit simulé

Cette méthode s'avère être bien adaptée pour la résolution du problème de placement de composants, elle donne de bonnes solutions approximatives pour ce problème.

Premièrement l'algorithme recuit simulé commence à partir d'une solution initiale et température T élevée. À chaque itération ce permutant les positions de certains composants du circuit choisis aléatoirement. Si la fonction diminue, on accepte la nouvelle solution sinon on l'accepte avec la probabilité, $e\text{^}(-(\text{Ei}+1-\text{Ei})/T)$. C'est-à-dire qu'on accepte l'augmentation seulement si le tirage aléatoire d'un nombre dans l'intervalle [0,1] est supérieur

à $e^{(-(Ei+1-Ei)/T)}$, L'algorithme continue ainsi jusqu'à ce que la température atteigne un certain niveau minimal, l'algorithme s'arrête alors et le meilleur positionnement trouvé tout au long de la recherche sera la solution approximative de notre problème.

3.3.3.1. L'algorithme de recuit simulé

Début

Choix de la température de départ T
Générer une solution quelconque S
Choix d'un coefficient a : 0<a<1 ;
Calcul de l'énergie E en fonction de S
Répéter
Choix d'une solution S_{new}
Calcul de l'énergie E_{new} en fonction de Snew
Valide=0
Si Enew <E alors
Valide =1
Sinon
Tirage d'un nombre p : 0<p<1
Si p> $e^{-\frac{Ei+1-Ei}{T}}$ Alors
Valide =1
FinSi
FinSi
Si valide=1 Alors
E ← E_{new}
T ← T*a
FinSi
Jusqu'à (T ≤ *t*)

Fin

4. Conclusion

Dans ce chapitre, nous avons présenté la conception la conception d'un système qui représente l'outil de résolution du problème de placement, ainsi nous avons adapté les méthodes de résolution choisies, La conception proposée comporte les concepts nécessaires pour assurer les exigences que nous avons pris en compte afin d'assurer le bon fonctionnement du système proposé.

Alors les couches internes sont illustrées, les mécanismes d'interaction, et de décision sont discutés.

Dans le chapitre suivant, nous allons décrire la réalisation du système conçu dans ce chapitre, et nous allons effectuer des séries de tests sur les deux méthodes implémentées, les résultats seront analysés et discutés.

1. Introduction

Dans cette dernière partie nous allons d'abord décrire la mise en œuvre de notre système citer les choix techniques et détailler les structures de données utilisées, ensuite nous allons analyser les résultats obtenus d'un ensemble de tests pour trouver rapidement les plages de variation de ces paramètres de contrôle de chaque méthode de résolution qui permettent une bonne convergence des algorithmes. Afin d'effectuer une comparaison entre les méthodes implémentées dans notre outil de résolution.

2. Environnement de développement

Notre système est développé sous l'environnement suivant :

- Micro-portable Sony vaio i3 (CPU 2.13GHz, RAM 4Go).
- Système d'exploitation Microsoft Windows 7.

Pour développer notre application on a choisi l'environnement builder C++ pour la programmation.

C++ Builder est un outil RAD ou Rapid Application Développent, c'est-à-dire tourné vers le développement rapide d'applications sous Windows. C++ Builder permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows,

3. Langage de programmation

Le langage de programmation utilisé pour le développement de notre système est le C++. C++ est un langage de programmation crée par Bjarne Stroustrup et normalisé en 1998. Le ++ est l'opérateur d'incrémentation (qui fais +1) : le C++ est donc une amélioration de l'ancestral langage C.

4. Implémentation

4.1. Illustration de l'outil de résolution

Notre outil de résolution consiste trois interfaces qui sont :

4.1.1. Interface principale

À partir cette interface on choisit la méthode de résolution (algorithme génétique et recuit simulé) qu'on va exécuter pour trouver la solution du meilleur placement de ce circuit.

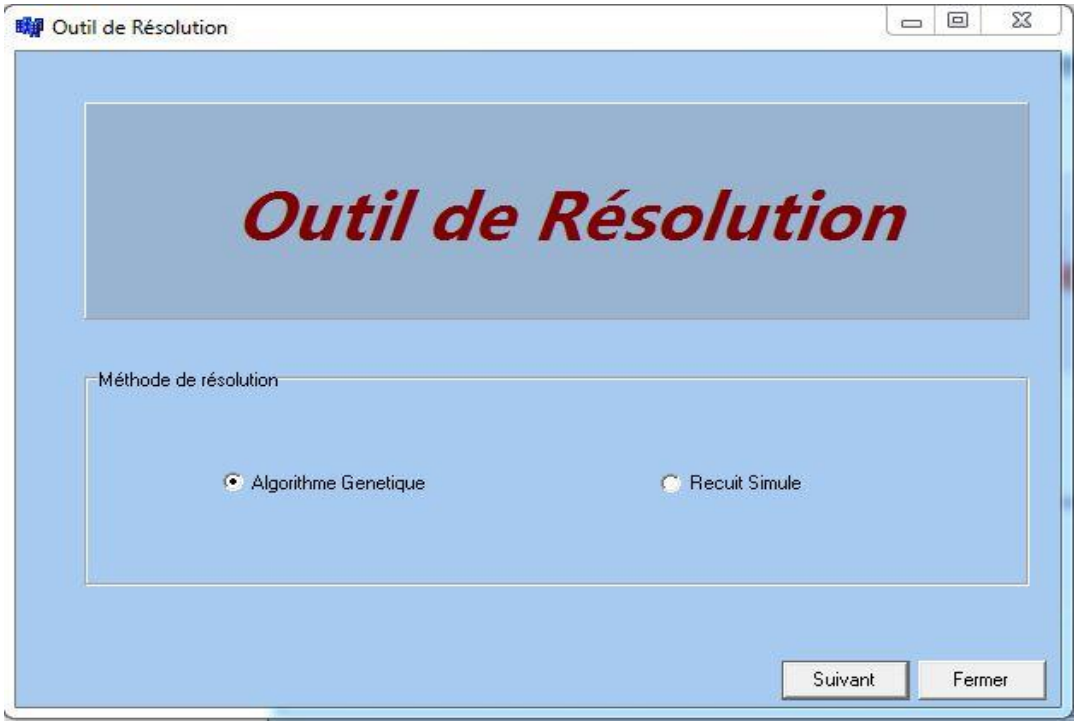

Figure 30 : L'interface principale de l'outil de résolution.

4.1.2. Interface recuit simulé

À partir cette interface on commence par le choix de la génération du circuit électronique. Ensuit cliquer sur le bouton générer pour créer des fils de liaison entre les composants électronique puis nous entrons les paramètres de recuit simule (la température initiale et facteur de décroissance) puis on exécute l'algorithme de résolution. Dans le coté adroite on affiche un graphe qui représente la meilleur position des composants trouvées et affiché la courbe de la fonction de coût et du temps de calcule.

En outre affiché la longueur totale des fils et temps d'exécution temps de calcule.

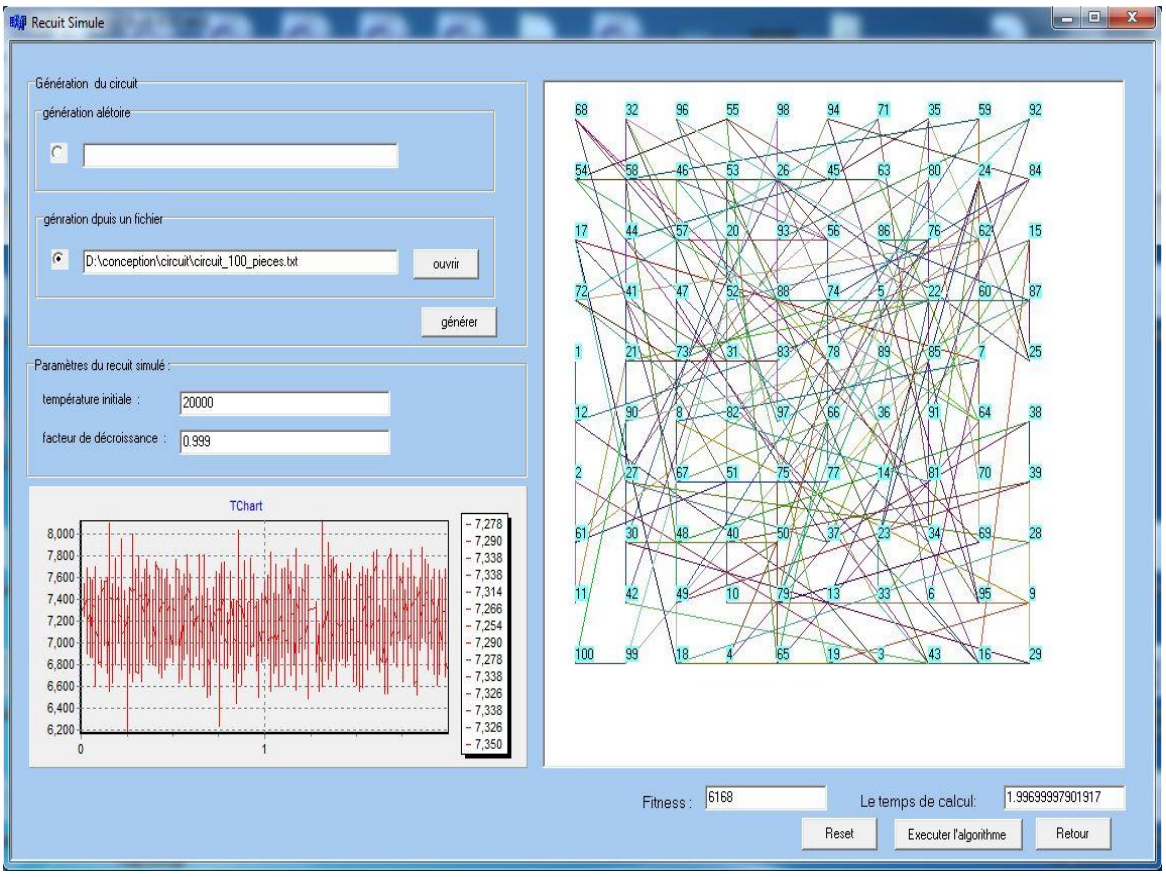

Figure 31 : L'interface recuit simulé de l'outil de résolution.

4.1.3. Interface de l'algorithme génétique

La même chose pour l'algorithme génétique en ce qui concerne la génération de circuit et l'affichage des résultats. La différence entre les deux interfaces c'est les paramètres de chaque méthode. Et dans l'algorithme génétique s'affiche un tableau qui contient la population initiale et la population finale.

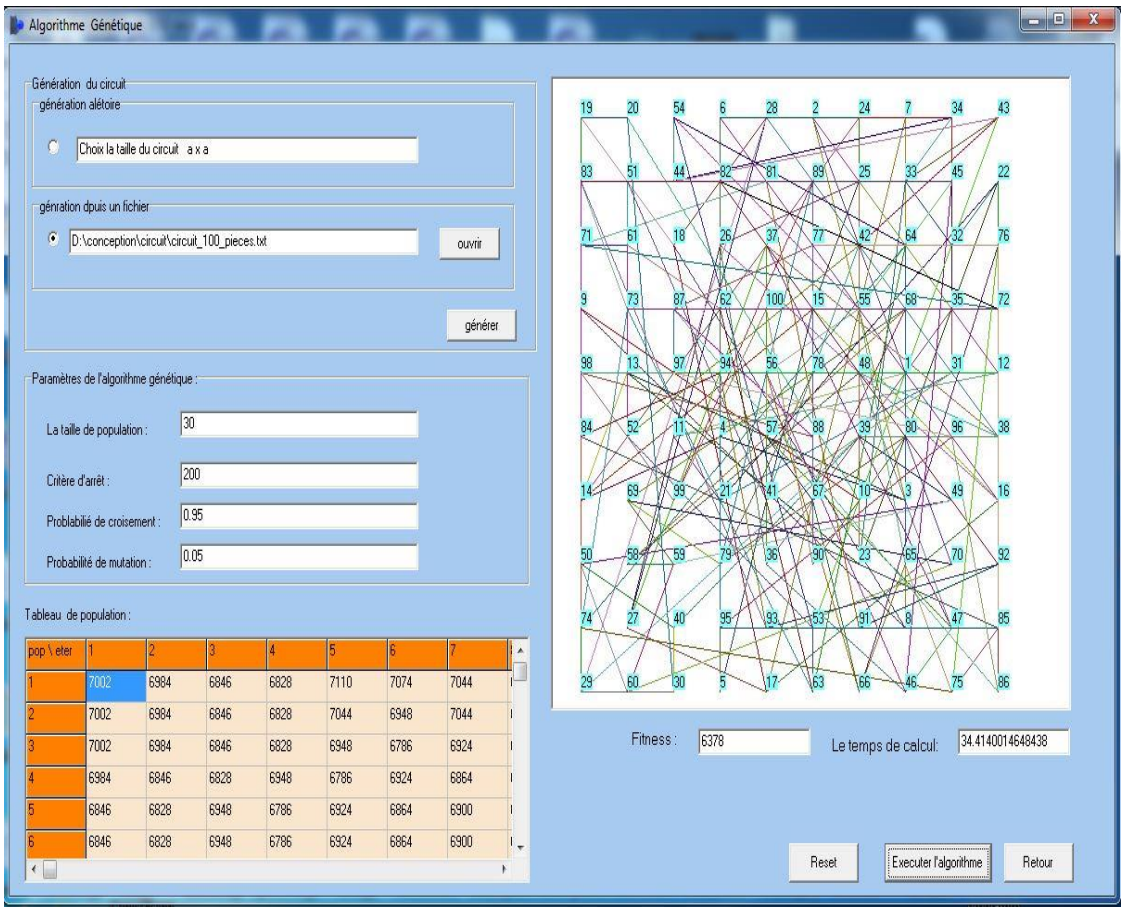

Figure 32 : L'interface de l'algorithme génétique de l'outil de résolution.

4.2. Structures de données utilisées

On présente dans ce qui suit les différentes structures de données utilisées

- **Circuit :** c'est une structure de données (vecteur) qui représente l'ensemble des composants chaque composant électronique elle contient deux variables entières (x, y).
- **T.C** : c'est un vecteur de composant utilisé pour sauvegarder les cordonnées de chaque composant du circuit.
- **Y :** c'est un vecteur de composant utilisé pour sauvegarder la meilleure solution trouvée.
- **chart1 :** c'est un vecteur utilisé pour sauvegarder le temps de calcul et la fitness de chaque itération, l'objectif de ce vecteur est de dessiner une courbe de l'algorithme du recuit simulé
- **Matrice A :** cette matrice utilisée pour sauvegarder les liaisons entre les composants du circuit

- **Matrice D :** utilisé pour sauvegarder les positions des composants du circuit
  En plus pour le dessin graphique du circuit.
- **Vecteur X :** c'est un vecteur de vecteur, qui contient les vecteur T.C, elle représente la population
- **Vecteur Sel1 :** utilisé pour garder deux chromosomes sélectionnés
- **Vecteur Sel1 :** utilisé pour garder deux chromosomes sélectionnés
- **Vecteur Sol :** c'est un vecteur qui contient tous les fitness de la population final, l'objectif de vecteur **sol est de** sélectionner la meilleure solution trouvée en plus de donner l'indice du meilleur vecteur de position

4.3. Génération du circuit électronique

À travers l'outil de résolution nous choisissons le mode de génération de circuit électronique, en utilisent deux méthodes :

- **Mode génération aléatoire :** cette mode est utilisée pour générer aléatoirement la connectivité (les arcs) entre les composants électronique
- **Mode génération depuis un fichier :** sur un fichier binaire, le nombre de ligne ou la colonne elle représente le nombre des composants électronique, si il contient 1 en ligne i et en colonne j alors les composants i et j sont reliés et 0 sinon. Après la lecture on copie ligne par ligne dans une matrice A.

4.4. Critères de comparaisons

Nous avons retenu deux critères principaux pour exécuter l'étude comparative. Ces critères sont :

- **Qualité de la solution :** la qualité du résultat final découverte par chacun des algorithmes, le minimum de la fonction de coût en terme de la longueur totale des fils du circuit électronique.

- **Temps d'exécution :** le temps CPU utilisé par un algorithme pour résoudre un problème de placement. Ce critère est dépendant de la machine utilisée, pour cette raison tous les tests sont réalisés sur la même machine.

4.5. Étude et analyse des paramètres de contrôle

Pour analyser les des paramètres de contrôle de chaque méthode stochastique, on a choisi de faire un test. Cela permet de trouver rapidement les plages de variation de ces paramètres qui permettent une bonne convergence des méthodes.

4.5.1. Paramètres du recuit simulé

L'algorithme de recuit simule comprend un nombre réduit de paramètres. Leurs significations sont rappelées ci-dessous :

- la température initiale T(le nombre d'itérations),
- Facteur de décroissance (le coefficient de réduction de température),

Les paramètres influents pour recuit simulé sont les suivants :

- Pour l'étude de l'influence de la température initiale on propose les choix suivants :
  - on fixe le facteur de décroissance (q= 0.999),
  - la taille de circuit (a=49 composants),
- Pour l'étude de l'influence du facteur de décroissance on propose les choix suivants:
  - on fixe la température initiale (T = 10000),
  - la taille de circuit (a=49 composants)

4.5.1.1. Tests sur la température initia

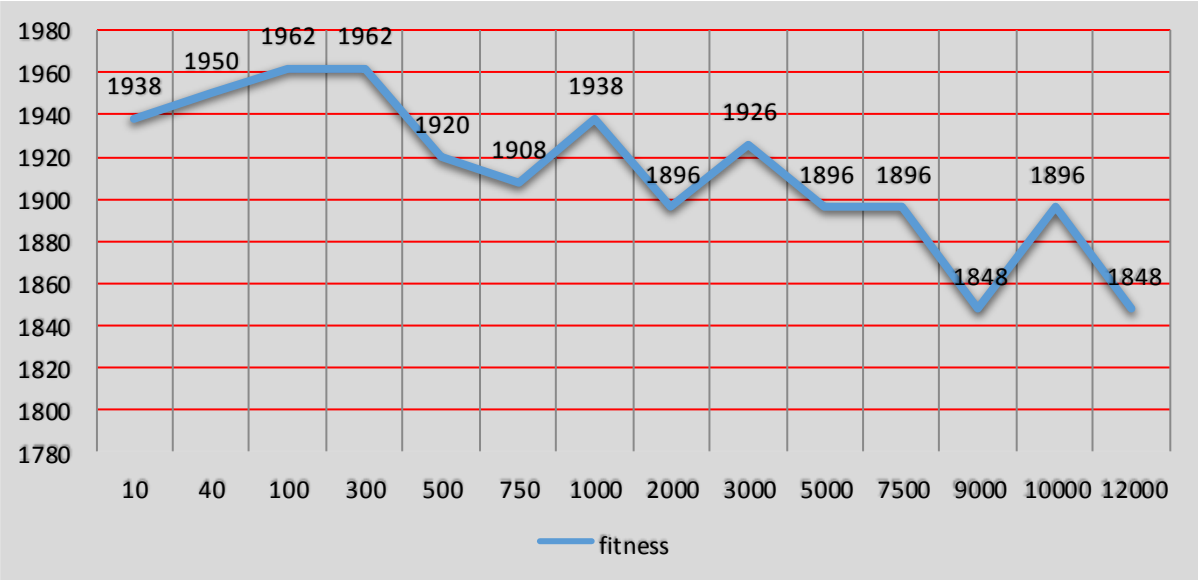

Figure 33: variation de la fitness en fonction de la température initiale

À travers la figure 33 on remarque dans l'intervalle [2000 à 12000] de la température, Nous obtenons des bonnes fitness, Et on constate que la fonction du coût se rapproche de l'optimum avec l'augmentation la température initiale.

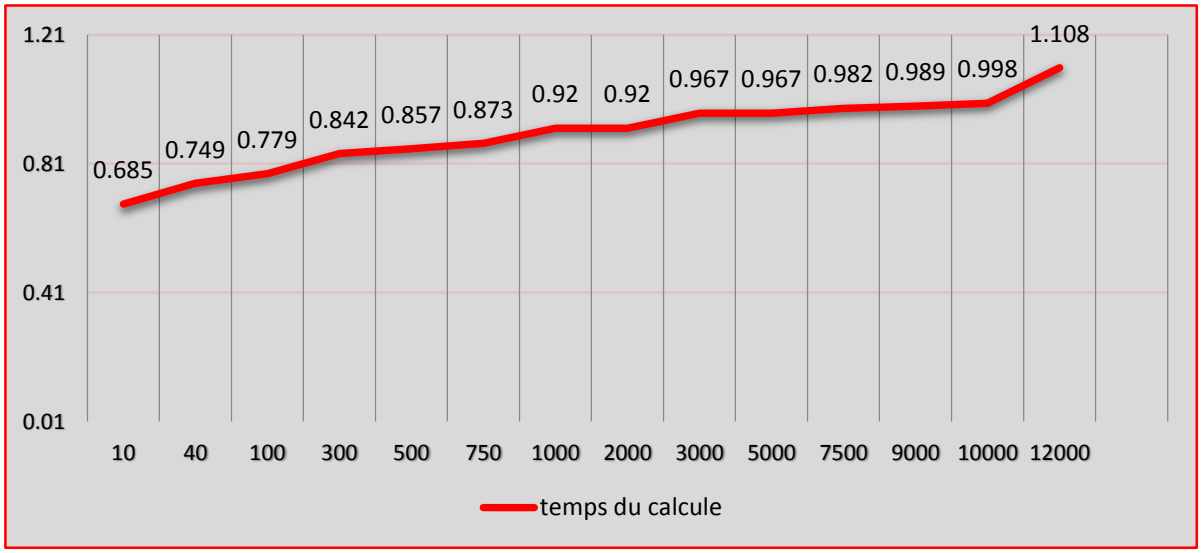

Figure 34: variation de temps de en fonction de la température initiale

La figure 34 démontre comment s'évolue le temps du calcul avec l'augmentation de la température. Grâce à la forme générale de la figure 34 on peut déduire que la température n'a pas une grande influence sur le temps de calcul.

4.5.1.2. Tests sur le facteur de décroissance

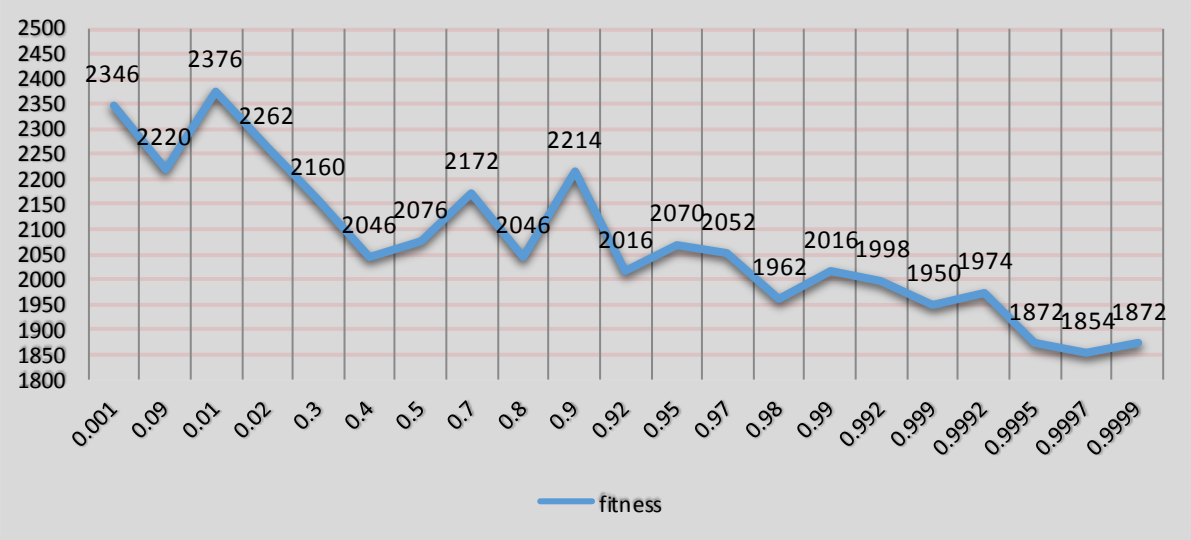

Figure 35 : variation de la fitness en fonction de facteur de décroissance

À partir la figure 35 on remarque dans l'intervalle [0.98 à 0.99] du facteur de décroissance, L'outil de résolution donne des bonnes fitness par rapport aux fitness après cette intervalle, et d'une manière générale la fitness se rapproche de l'optimum avec l'augmentation du facteur de décroissance.

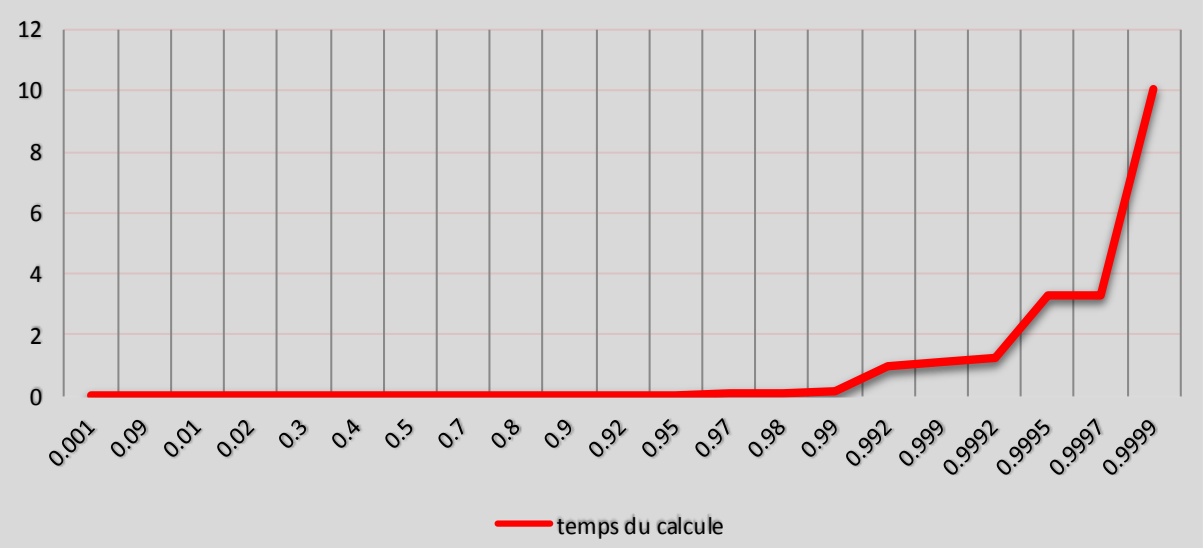

Figure 36 : variation de temps de calcule en fonction de facteur de décroissance

À partir figure 36 on remarque Dans l'intervalle [0.001 à 0.99] du facteur de décroissance, le temps de calcule égal 0 et après cette intervalle le temps est augmenté. Et on constate que le facteur de décroissance a une influence importante sur le temps de calcul, par exemple (facteur de décroissance=0.9999, temps de calcule =10.031s).

4.5.2. Paramètres de l'algorithme génétique

La solution obtenue avec l'algorithme génétique dépend d'un nombre réduit des paramètres qui sont :

- la taille de population (C),
- Critère d'arrêt (crt),
- Probabilité de croisement,
- Probabilité de mutation,

Les paramètres influents pour algorithme génétique sont les suivants :

- Pour l'étude de l'influence de la taille de population on propose les choix suivant:
  - ❖ on fixe le critère d'arrêt (crt= 50),
  - ❖ la taille de circuit (a=49 composants),
  - ❖ Probabilité de croisement (Pc=0.95),
  - ❖ Probabilité de mutation (Pc=0.95),
- Pour l'étude de l'influence du critère d'arrêt on propose les choix suivant:
  - ❖ on fixe la taille de population (*C = 50)*,
  - ❖ la taille de circuit (a=49 composants),
  - ❖ Probabilité de croisement (Pc=0.95),
  - ❖ Probabilité de mutation (Pc=0.05),

4.5.2.1. Tests sur la taille de population initiale

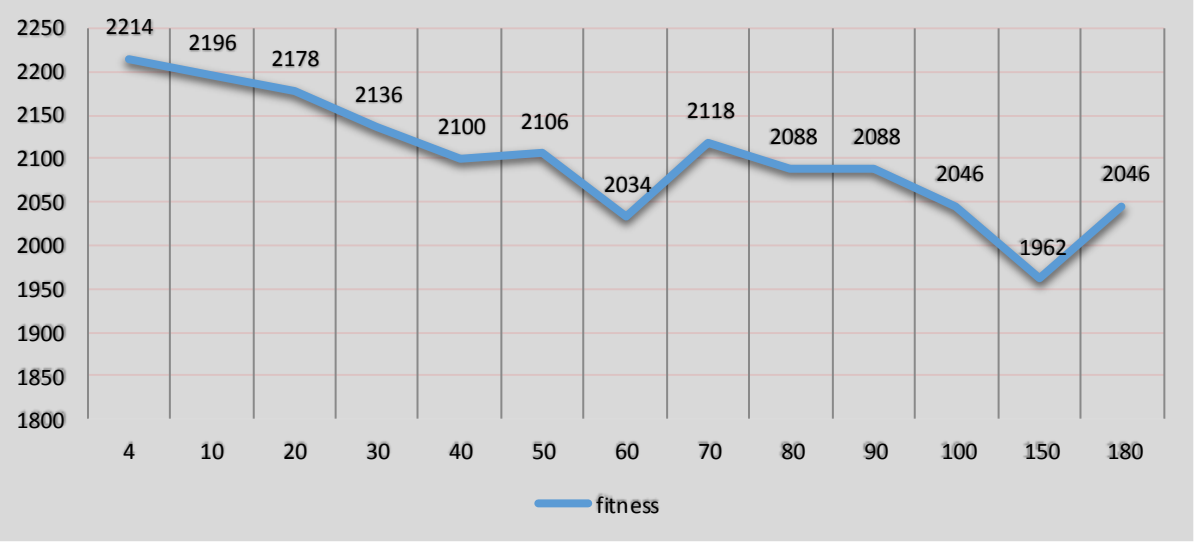

Figure 37 : variation de la fitness en fonction de la taille de population initiale

À travers la figure 37 on remarque dans l'intervalle [60 à 180] de la taille de population, Nous obtenons des bonnes fitness, et on constate que la valeur de fonction du coût se rapproche de l'optimum avec l'augmentation la taille de population.

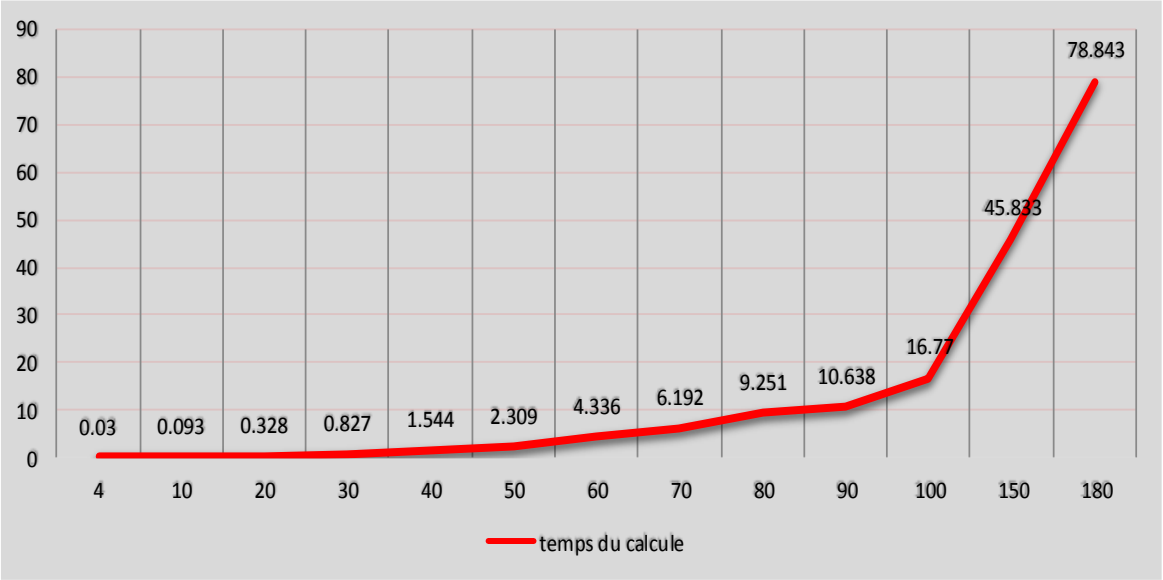

Figure 38 : variation de temps de calcul en fonction de la taille de population

La figure 38 illustre l'influence de la taille de population sur le temps de calcule, Grâce à la forme générale de la courbe on peut déduire que la taille de population a une grande influence sur le temps de calcul. Par exemple les valeurs de l'intervalle entre 100 et 180, le temps de calcule [16.77s, 78.893s].

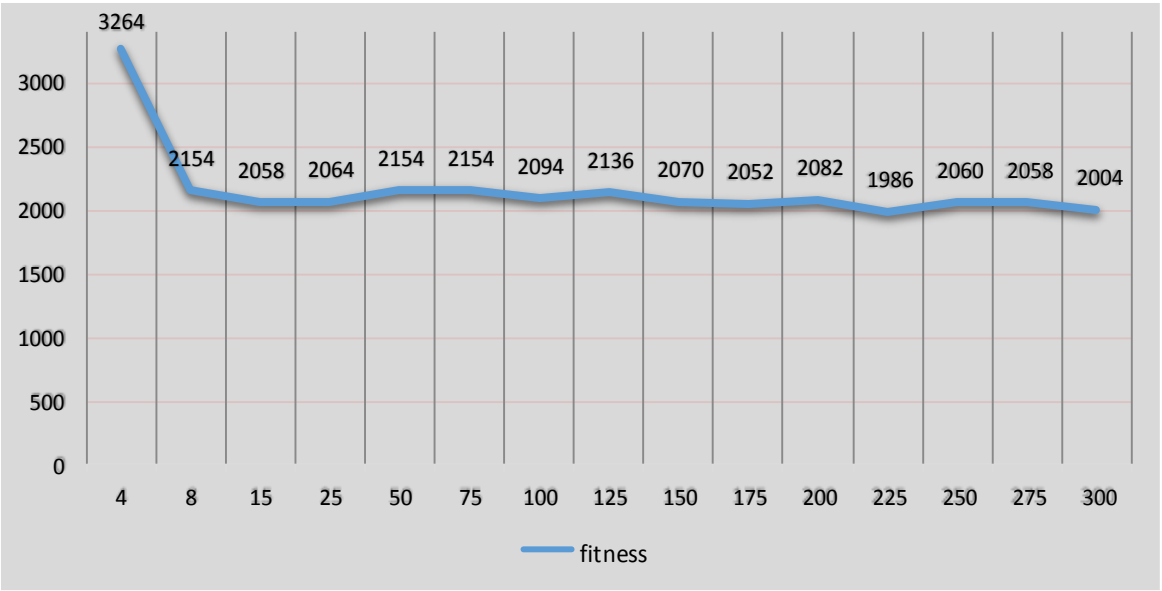

Figure 39 : variation de la fitness en fonction du critère d'arrêt

À travers la figure 39 on constate que la fonction du coût se rapproche de l'optimum avec l'augmentation du critère d'arrêt.

4.5.2.2. Tests sur le critère d'arrêt

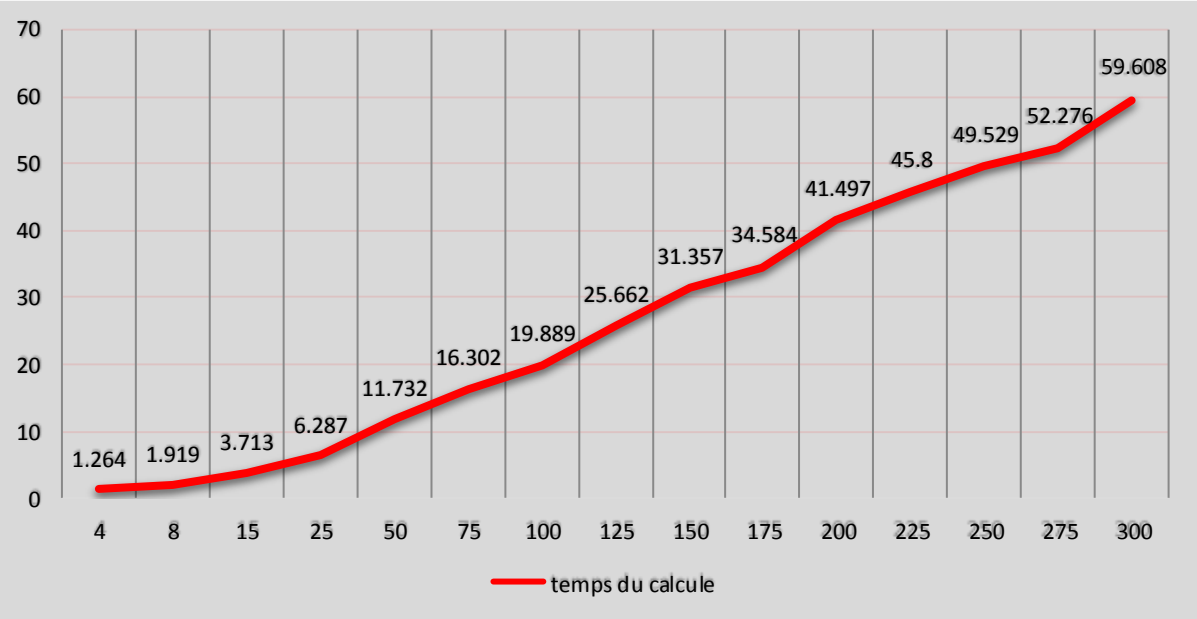

Figure 40 : variation de temps du calcul en fonction du critère d'arrêt

La figure 40 de montre comment s'évolue le temps du calcul avec l'augmentation de le critère d'arrêt. Grâce à la forme générale de la figure 40 on peut déduire que le critère d'arrêt a une grande influence sur le temps du calcul.

Grâce à ces tests, on peut dire que pour donner une plus grande liberté d'exploration de l'espace de recherche, on choisit la température initiale T et le facteur de décroissance qui doivent être tout les deux élevés, On met par exemple la température initiale T=10000 et le facteur de décroissance q = 0.9999, de cette manière le recuit simulé est efficace.

En ce qui concerne l'algorithme génétique le choix de la taille de population est très important, si la taille de population est petite on risque d'engendrer de mauvais résultats, donc pour éviter le risque on choisit la taille de population élevée. En outre on choisit la probabilité de mutation et la probabilité de croisement et le critère d'arrêt qui permet d'explorer l'espace de recherche.

À travers les tests sur l'influence des paramètres sur la qualité de la solution et du temps du calcule pour déterminer les meilleures valeurs de paramètres qui vont être utilisées pour les tests finaux, On a donc retenu les paramètres suivants pour chaque test :

Méthodes et paramètres		La taille de circuit							
		3x3	4x4	5x5	6x6	7x7	8x8	9x9	10x10
RS	Température initiale	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
	Facteur de décroissance	20000	20000	20000	20000	20000	20000	20000	20000
AG	Taille de population	40	40	40	40	40	40	40	40
	Critère d'arrêt	300	300	300	300	300	400	400	400
	Probabilité de mutation	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
	Probabilité de croisement	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05

Tableau 1 : Les paramètres fixés pour les tests de chaque problème

4.6. Les résultats et les analyses

4.6.1. Fonction de coût

Nous présentons dans ces tableaux les résultats comparatifs du recuit simulé et de l'algorithme génétique Pour obtenir ces résultats, chaque algorithme est exécuté 15 fois sur chaque circuit.

Méthodes	Fonction de cout	La taille de Problème							
		3x3	4x4	5x5	6x6	7x7	8x8	9x9	10x10
RS	Minimum	72	252	534	1056	1734	2856	4290	6054
	Moyenne	72	251.2	562.4	1076.4	1842.4	2725.6	4357.2	6161.6
	Maximum	72	258	588	1122	1884	3018	4434	6264
AG	Minimum	72	246	528	1032	1836	2898	4284	5826
	Moyenne	91.2	262.4	584.8	1120.4	1908.8	3006	4398.4	6189.6
	Maximum	114	276	618	1158	1974	3090	4500	6378
AG-RS	Minimum	0	-6	-6	-24	-102	42	-6	-228
	Moyenne	19.2	11.2	22.4	44	66.4	280.4	41.2	28
	Maximum	42	18	30	36	90	72	66	114

Tableau 2 : comparaison de deux méthodes

La première colonne du tableau 2 contient le nom de méthode de résolution. Les colonnes (3, 4, 5, 6, 7, 8, 9 et 10) représentent respectivement la valeur minimale, moyenne et maximale de fitness. Enfin, les 3 dernières lignes présentent l'écart du coût minimal, moyen et maximal entre l'algorithme génétique et le recuit simulé. En outre les valeurs colorées représentent les meilleurs fitness trouvées.

La figure suivant illustre les changements du coût minimal, moyen et maximal des deux méthodes dans les 8 problèmes :

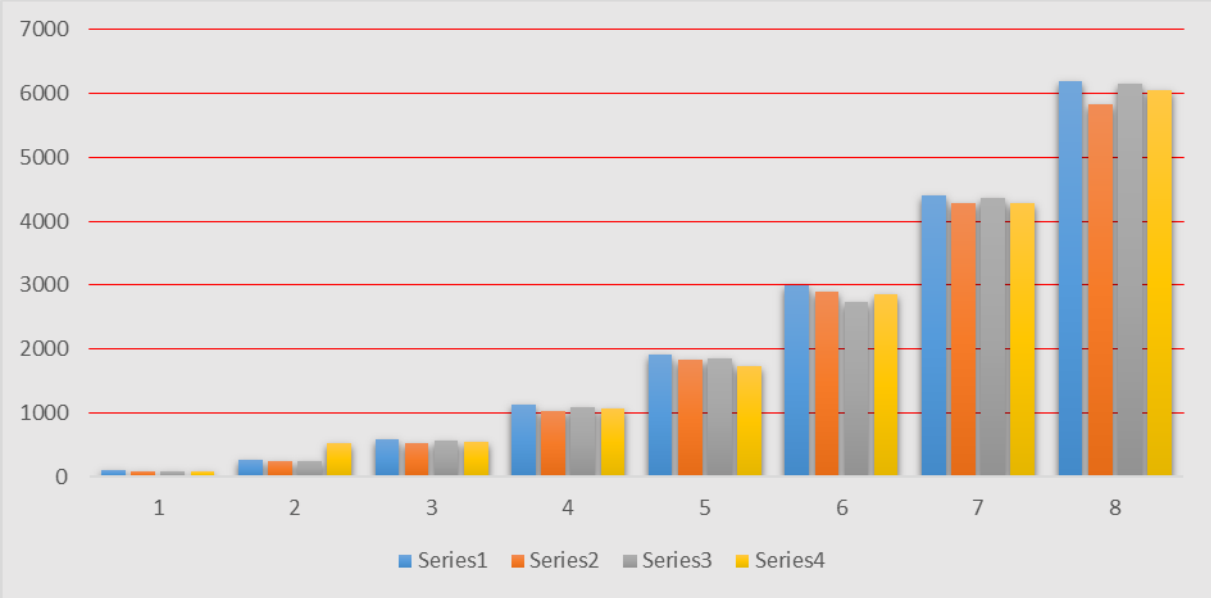

Figure 41 : Résultats comparatifs du recuit simulé et l'algorithme génétique

Sur la base des résultats obtenus, on constate que l'algorithme génétique toujours nous fournit une bonne solution par opposition au Recuit simule qui nous a donné que deux bonnes solutions dans les problèmes (circuit 7x7 et circuit 8x8). Mais cela ne signifie pas que le recuit simulé ne donne pas des bons résultats. En plus tous les écarts du coût moyen montrent que tous les coûts moyens de recuit simulé sont meilleurs par rapport à tous les coûts moyens de l'algorithme génétique.

4.6.2. Le temps d'exécution

Le tableau suivant le temps de calcul moyen des deux algorithmes lors de nos tests

Temps de	La taille de Problème							
calcul moyen	3x3	4x4	5x5	6x6	7x7	8x8	9x9	10x10
RS	6.323	7.102	8.105	9.691	10.748	11.86	15.97	20.251
AG	10.758	17.183	23782	33.219	46.478	66.439	103.884	115.658

Tableau 3 : Temps de calcul moyen en secondes du deux méthodes

La figure suivant illustre les changements du temps de calcul moyen des deux méthodes dans les 8 problèmes :

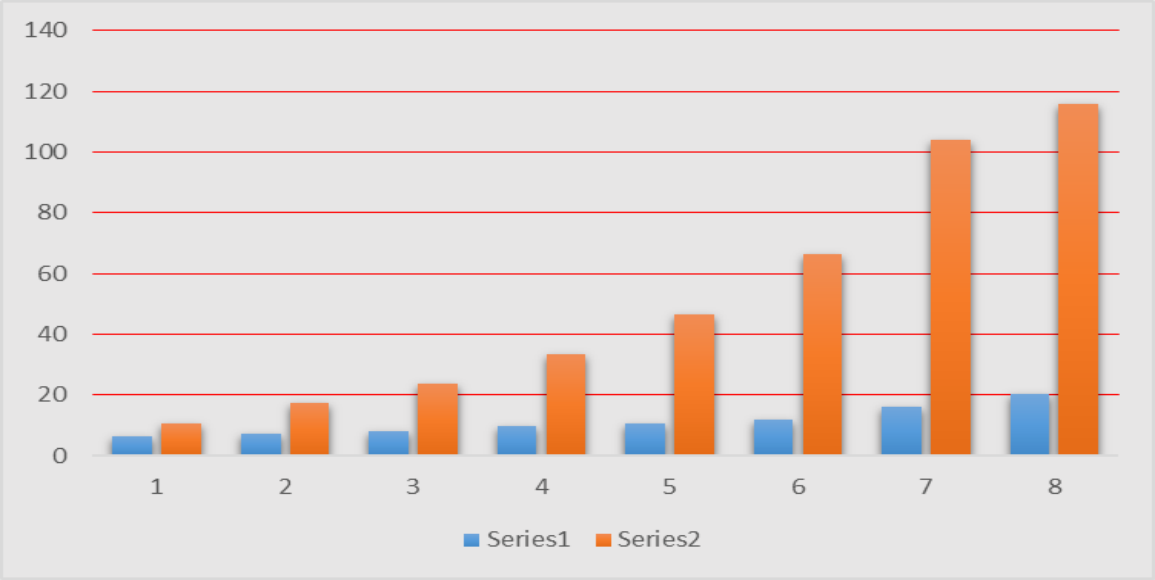

Figure 42 : Temps de calcul moyen des deux méthodes en fonction de la taille du problème.

À partir les résultats représentés dans le Tableau 3 et la figure 43 on remarque que dans tous les problèmes, le temps de calcul moyen de l'algorithme génétique est très grand par rapport à celui de recuit simulé. Puis on ne constate que le temps de calcul moyen des deux algorithmes augmente avec l'augmentation de la taille de problème, mais le temps de calcul de l'algorithme génétique augmente d'une façon plus importante.

4.7. Discussion final

Après avoir analysé les résultats des tests des deux algorithmes de Le recuit simulé et l'algorithme génétique, nous abordons la discussion sur ces deux algorithmes

Recuit simulé

Dans la résolution des problèmes de petite taille, le recuit simulé donne des résultats variés mais pas forcément insatisfaisants, cette variation est due à la large intervention du hasard dans l'algorithme du recuit simulé.

Dans l'algorithme du recuit simulé, le choix de configurations voisines à chaque itération est de manière aléatoire, c'est ce qui explique le temps de calcul raisonnable par rapport à l'algorithme génétique

Algorithme génétique

Algorithme génétique donne de meilleurs résultats quand la taille du problème est grand ou un problème de taille raisonnable. Et le temps de calcul de l'algorithme génétique est très grand par rapport le recuit simulé.

5. Conclusion

À partir des résultats expérimentaux obtenus précédemment, on conclut les points essentiels suivants :

- En terme de bonnes fitness, en général Algorithme génétique est meilleure.
- En terme de temps de calcul, Le recuit simulé est le plus préformant.

Conclusion générale

Dans ce travail, nous avons effectué une étude expérimentale de méthodes de résolution heuristiques. En particulier, nous avons présenté deux algorithmes fondés sur le recuit simulé et l'algorithme génétique pour la résolution du problème de placement de composants électroniques.

Des expérimentations comparatives des deux algorithmes ont été réalisées sur des tests de tailles variées. Les résultats ont montré les points suivants :

- Un bon choix des paramètres d'une méthode est très important, le réglage des paramètres est crucial et conditionne la qualité des résultats obtenus. Cependant, un réglage optimal reste une tâche difficile.

Les perspectives issues de ce travail sont :

Le recuit simulé quand il vient à lui, il peut être amélioré en ajoutent un mécanisme de paliers de températures dynamiques qui assurent la diversification et l'intensification dans la recherche de solution.

L'algorithme génétique peut être amélioré en spécifiant des critères d'arrêts qui lui conforme dont le but est de trouver des meilleurs solutions, ou bien ajouter des contraintes ou des systèmes complexes pour filtrer une bonne population initiale ce qui va aider de trouver une solution optimale.

Bibliographie

[1] Abdesslem LAYEB, Utilisation des Approches d'Optimisation Combinatoire pou La Vérification des Applications Temps Réel. Thèse de Doctorat, Université Mentouri de Constantine 2010

[2] Mostepha, R : Résolution de problèmes d'optimisation combinatoire par systèmes artificiles auto-organisés. Thèse de magister, Université Mentouri de Constantine ,2008.

[3] Palpant M. : Recherche exacte et approchée en optimisation combinatoire : schémas d"intégration et applications. Thèse de Doctorat, Université d"Avignon, 2005.

[4] Reeves, C.: Modern Heuristic Techniques for Combinatorial Problems. Advances topics in computer science. Mc Graw-Hill, 1995.

[5] Omessaad, H : Contribution au développement de méthodes d'optimisation Stochastiques application à la conception des dispositifs électrotechniques, Thèse de Doctorat, Université De Lille France 2003.

[6] Lambert veller sylvain , lechevalier david,quirico tommy «Problème de ramassage dans une ville virtuelle - Algorithme Tabu Search »Université de Bourgogne 2010-2011

[7] Aissa Boulmerka, « Adaptation des métaheuristiques à l'ordonnancement hors-ligne d tâches temps réel à contraintes strictes en environnement monoprocesseur », Mémoire de magister en informatique, 20 Avril 2009.

[8] Alain Hertz, 'L'optimisation Combinatoire', École Polytechnique, Canada, 2006

[9] Rachid Chelouah, 'L'optimisation combinatoire', INA Institut d'informatique appliquée, Suisse, 2003.

[10] http://khayyam.developpez.com/articles/algo/genetic/

[11] *Souquet Amédée* « Algorithme génétique » *Radet Francois-Gérard*

[12] Ratnesh Kumar, Wai Kit Leong, Robert J. Heath, 'An Integer Programming Approach to Placement and Routing in Circuit Layout', université du kentucky, 2006.

[13] Fouad Bennis 'Méthode générique pour l'optimisation d'agencement géométrique et fonctionnel', Institut de Recherche en Communications et Cybernétique de Nantes, janvier 2010

[14] Sataj Sahni, Atul Bhatt, Raghunath Raghavan, « the complexity of design automation problems », université de Minnesota, 1980.