République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider Biskra
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

N°d'ordre : .....
Série : .....

Mémoire

Présenté en vue de l'obtention du diplôme de

Magister en Informatique

Option : Data Mining et Multimedia

Thème

RÉSOLUTION DE PROBLÈMES D'OPTIMISATION PAR LES SYSTÈMES MULTI-AGENTS ET LES APPROCHES ÉVOLUTIONNAIRES

Par

Aziza BECHIR

Mohamed Chaouki BABAHENINI	MCA	Université de Biskra	Président
Kamal Eddine MELKEMI	MCA	Université de Biskra	Rapporteur
Abdelmalik BACHIR	PROF	Université de Biskra	Examinateur
Ammar LAHLOUHI	MCA	Université de Batna	Examinateur

Je dédie ce travail à :

Mes chers parents ...

Mon frère et mon mari...

Toute la famille ...

Et tous mes amies...

REMERCIEMENTS

Tout d'abord, je remercie Allah, le tout puissant, qui m'a donne la force,la patience et la volonte pour accomplir ce modeste travail.

Je tiens à remercier vivement mon encadreur, Kamal Eddine Melkemi qui, par son expérience et sa tolérance, a guidé mon travail tout en préservant mon esprit d'initiative, durant toute la période de la préparation de mon mémoire.

Ce stage m'a donné l'opportunité de me confirmer et de découvrir le domaine de la recherche.

J'adresse également mes remerciements à l'ensemble des professeurs qui m'ont enseigné durant mon année théorique, notamment, à Monsieur le chef du département, Dr Mohamed Chaouki Babahenini pour son accompagnement pédagogique et scientifique.

Je tiens à exprimer ma gratitude une double fois à l'égard du Président du Jury.

Je remercie également les membres du jury ; Mr Albdelmalik Bachir, Mr Ammar Lahlouhi pour leurs déplacement malgré leurs charges pédagogiques et scientifiques.

RÉSUMÉ

La plupart des problèmes d'optimisation combinatoire sont des problèmes très difficiles à résoudre et que l'on trouve dans plusieurs domaines à savoir la fabrication, l'énergie, les télécommunications, la médecine, la robotique etc ....

Malgré l'existence d'un nombre important de méthodes de résolution de ce type de problèmes, elles restent limitées et insuffisantes dans la plupart de problèmes, spécialement pour la résolution des problèmes d'optimisations dans les systèmes critiques.

Pour cela, nous essayons de proposer des solutions adéquates pour résoudre ces problèmes. Nous utilisons des approches hybrides distribués efficaces.

Dans ce mémoire, nous traitons le problème de l'optimisation combinatoire en faisant une hybridation entre les approches inspirées de la nature et les systèmes multi-agents. Nous appliquons cette approche pour la résolution d'un problème l'ordonnancement de tâches.

Dans un premier temps, nous proposerons une hybridation entre les algorithmes génétique et l'informatique quantique et par la suite nous intégrons cette hybridation dans un système multi-agent.

Mots clés: optimisation combinatoire , algorithmes évolutionnaires, algorithme génétique, Informatique quantique, les systèmes multi-agents , Systèmes embarqués.

ABSTRACT

Real optimization problems are found in several areas such as manufacturing, energy, telecommunications, medicine, robotics etc .... Most of these combinatorial optimization problems are complex and very difficult to solve.

Despite the existence of a huge number of resolution methods, the optimisation of such problems by using these methods, remains limited and insufficient, especially for critical systems.

In this manuscript, we propose an efficient hybrid distributed approaches based on quantum genetic algorithm for solving task scheduling problem. In this work, we design our system using a multi-agent system. Exeperimental results are presented in order to show the validity and the performance of our MAS compared with some classical methods. We apply JADE to acheive our MAS.

Keywords : combinatorial optimization, evolutionary algorithms, Genetic Algorithm, Quantum Computing, multiagent systems, Embedded Systems.

ملخص :

مَشَاكِل التَحسِين التوَافُقِية هي مَشَاكِل صَعبة الحَل وَالتِي نَجِدُهَا فِي عِدَة مجَالَات مِثلَ السيَارَات، وَ الالكترُونيَات، اَلطيرَان، وَ الرُوبُوتَات.اَلخ . بالرغم من أَن هنَاك عدَد كبِير من الطرق لمعَالجة هذه المَشَاكِل ولكن هذه الطرق غِير كَافية بالنسبة لبعض المَشَاكِل، خصُوصَا مَشَاكِل التَحسِين التوَافُقِية فِي الأَنضمة الخطِيرة. و لهذَا يتطلب منَا حل هذه المَشَاكِل بَاستخدَام أَسَالِيب قوية و دقِيقة. و فِي هذه المذكرة نقُوم بمعَالجة مشكلة التَحسِين بوَاسطة الدمج بِين النهج المستُوحَاة من وحِي الطبِيعة و الأَنضمة المتعدَدة الخدمَات. أَولَا نقترح دمج بِين الخوَارزميَات الجنِية و الاعَلَام الالِي الكمِي. و بعد ذلك نقُوم بادرَاج هذَا الدمج فِي الأَنضمة المتعدَدة الخدمَات.

كلمَات مفتَاحية : التَحسِين، اَلخوَارزميَات التطورية، اَلخوَارزميَات الجنِية، اَلاعَلَام الالِي الكمِي، اَلأَنضمة المتعدَدة الخدمَات، اَلأَنضمة المدمجة.

Abréviation

DM	Deadline Monotonic
EDF	Earliest Deadline First
GA	Genetic Algorithm
LLF	Least Laxity First
MAS	MuliAgent System
MAS-QGA	MuliAgent System based on Quantum Genetic Algorithm
QGA	Quantum Genetic Algorithm
Q-agent	Quantum Agent
RM	Rate Monotonic
SED	Système embarqué distribué

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

Les applications de l'optimisation sont nombreuses, où chaque processus peut-être amélioré et optimisé. Donc, il n'y a aucune entreprise ou société qui n'est pas impliquée dans la résolution de problèmes d'optimisation. En effet, de nombreuses applications difficiles ou complexes peuvent être modélisées comme des problèmes d'optimisation.

L'objectif de l'optimisation est de minimiser les temps calculs, le coût et le risque ou maximiser le profit ou le gain, la qualité et l'efficacité etc....

Dans la vie pratique, un grand nombre de problèmes d'optimisation dans la fabrication, la médecine, et les affaires sont complexes et difficiles à résoudre.

Ces problèmes ne peuvent pas être résolus par des méthodes exactes dans un laps de temps raisonnable. Alors, la principale alternative pour résoudre cette catégorie de problèmes est d'utiliser des algorithmes approximatifs.

L'objectif de ce mémoire consiste donc à étudier d'une part les approches inspirées de la nature (Algorithms inspired from nature [19]) utilisées pour résoudre ce type de problème d'optimisation combinatoire. D'autre part, le but est de proposer et de concevoir une nouvelle approche hybride distribuée et de l'appliquer à un des problèmes réels.

Nous exploitons les systèmes multi-agents (Multiagent system (MAS)) [26] comme outils de modélisation distribués dans la conception de notre approche, ainsi que les approches évolutionnaires.

Melkemi et al. [51] ont proposé un système MAS-GA pour la segmentation des images. Cette méthode utilise une perturbation des pixels dans l'image afin de réduire à la fois la taille de la population et le nombre d'itérations de l'algorithme génétique (Genetic Algorithm (GA)). Par ailleurs, Melkemi et Foufou [52] ont proposé une nouvelle approche distribuée de segmentation d'images basée sur une représentation floue des pixels dans une image. Deux architectures des MASs ont été proposée.

Dans ce travail, nous avons opté d'étudier le problèmes d'ordonnancement de tâches, en particulier dans un systèmes embarqué temps réels. Les approches évolutionnaires sont largement utilisées pour optimiser les problèmes complexes.

Plusieurs travaux ont été proposés sur des versions améliorées de ces méta-heuristiques en ajoutant quelques techniques afin de leurs permettre de résoudre les problèmes d'optimisation d'une manière plus efficace.

Actuellement, les problèmes d'optimisation deviennent de plus en plus complexe vus

le développement technologique. Malgré l'existence d'un grand nombre de méthodes approchées pour traiter ces problèmes, ces méthodes restent limitées.

Une tâche d'optimisation nécessite un certain temps pour la résolution et atteindre une solution raisonnablement proche de la solution optimale. Mais, en même temps ce temps peut être grande pour certaine instance de problèmes surtout dans les système critique dont le retard peut mener à des cas catastrophiques. Donc, il faut penser à proposer des approches efficaces rapides qui permettent un gain de performance qui se traduit par un temps de calcul raisonnable.

Dans ce mémoire, nous présentons une nouvelles approche hybride distribuées basée sur l'informatique quantique [38, 44], les approches évolutionnaires [30, 35] et les systèmes multi-agents (MASs) [26].

Cette approche est appliquée pour l'optimisation des problèmes d'ordonnancement des systèmes embarqué temps réel [59, 61]. Pour cela, nous introduisons au sein de chaque agent un algorithme génétique quantique efficace comme savoir-faire d'un agent. Nous appelons ce type d'agent *un agent quantique* (Quantum Agent (Q-agent)).

Cette approche permet une évolution naturelle [30, 35] et rapide vers une meilleure solution pour notre problème.

En effet, notre SMA quantique vise d'une part à réduire la taille de la population de notre AG d'une manière à pallier aux limitations des algorithmes génétiques classiques[35], et d'autre part à résoudre le problème dans un temps réel.

Dans ce travail, nous nous intéressons à perturber les données relatives aux individus en utilisant une inspiration de la physique quantique. Cette approche à montré sa robustesse [53, 54].

Dans notre travail, nous nous basons sur une approche hybride, qui consiste à combiner les GAs et les MASs, pour proposer une nouvelle approche distribuée d'optimisation pour l'ordonnancement. Dans le reste du manuscrit, nous appelons cette méthodes MAS-QGA (Multiagent system based on quantum genetic algorithm).

Afin de concevoir cette approche, d'abord nous suivons deux étapes principales :

Premièrement, nous commençons par une exploration et une réalisation d'un GA [30, 35]. Deuxièmement, nous proposons une amélioration en s'inspirant du concept du quantique dans le but de diversifier les individus afin de réduire le nombre d'itération de notre GA [35]. Un SMA est utilisé pour concevoir notre application.

Après cette introduction, Ce mémoire est organisé comme suit :

— Le chapitre 1 est une présentation des notions de base sur les problèmes combinatoires, l'optimisation, la théorie de la complexité, et enfin la classification en générale des méthode d'optimisation.

— Le chapitre 2 est dédié aux notions d'agents et des systèmes multi-agents suivis par les protocoles de communications entre agents.

— Le chapitre 3 est consacré à une introduction aux systèmes embarqués distribués temps réel, et à l'ordonnancement des tâches temps réel sur une architecture monoprocesseurs et multiprocesseurs ainsi que les principales classes des algorithmes d'ordonnancement.

— Dans le chapitre 4, nous focaliserons sur les GAs ainsi que leurs principales étapes, puis donner un état de l'art sur les travaux concernant l'application de ces derniers aux problèmes d'ordonnancement classiques et temps réel, en fin le concept des algorithmes génétiques quantiques est présenté.

— Le chapitre 5 présente l'approche proposée suivie par les résultats expérimentaux.

— A la fin de ce manuscrit, nous présenterons une conclusion générale 5.7 suivie par les perspectives envisageables.

CHAPITRE 1

L'OPTIMISATION COMBINATOIRE

Ce chapitre est une présentation des notions de base sur les problèmes combinatoires, l'optimisation et la théorie de la complexité. Par ailleurs, nous nous intéressons à la classification en général des méthodes d'optimisation en citant des méthodes existantes.

1.1 Introduction

Cela peut être une minimisation d'un coût de production d'une usine, dans l'optimisation du chemin d'un véhicule, etc....
Ces problèmes sont en général des systèmes complexes en vue de leur hétérogénéité et leurs diversités (exemple : des acteurs en jeux humains, appareils électroniques, etc.), à la masse importante des données, à la distribution des informations manipulées ainsi qu'à la dynamique des environnements dans lesquels ils se trouvent.

L'objet principal de ce travail consiste à étudier d'une part les phénomènes inspirés de la nature utilisés pour traiter le problème de l'optimisation à savoir l'évolution naturelle basée sur le principe de Charles Darwin "The survival of the fittest" [19, 30]. D'autre part, proposer et concevoir de nouvelles approches basées sur les systèmes multi-agents et les appliquer à des problèmes du monde réel.

Le principal objectif d'une optimisation d'un problème est de trouver une solution maximisant ou minimisant une fonction objective donnée. Les problèmes d'optimisation peuvent être combinatoires, continues ou mixtes.

Nous nous intéressons dans ce travail aux problèmes d'optimisation des problèmes combinatoires.

1.2 Concepts de base

1.2.1 Définition formelle de l'optimisation

Soit X l'ensemble des solutions du problème d'optimisation. X peut être un ensemble à n-dimensions, ou un ensemble binaire :
$X = \{0,1\}^n$ ou des valeurs réelles :
$X\ \mathbb{R}^n, \ldots$ etc. Le domaine X est souvent désigné par espace de recherche.

Considérons un problème d'optimisation défini par la fonction f appelée la fonction fitness (ou fonction coût, fonction erreur, fonction objective) et qui est définie pour évaluer la qualité (fitness) des solutions candidates dans X sur un problème donné :

$f : X \longmapsto \mathbb{R}$

Un problème d'optimisation consiste à trouver une solution maximisant (ou minimisant) la fonction objectif f de sorte que :

$\forall Y \in X : f(X) > f(Y)$ pour la maximisation.
$\forall Y \in X : f(X) < f(Y)$ pour la minimisation

— Les problèmes d'optimisation continue : optimiser une fonction objective sur un domaine continu.Plusieurs algorithmes polynomiaux existent pour de nombreux problèmes de cette famille.

— Les problèmes d'optimisation combinatoire : optimiser une fonction sur un domaine discret. Bien que décrit de façon concise, ce domaine (aussi appelé ensemble des solutions) est souvent de très grandes tailles.

— Les problèmes d'optimisation en variables mixtes : problèmes d'optimisation mixte comportent des variables de décision discrètes et continues. Si les contraintes et l'objectif du problème peuvent être exprimés de manière plus linéaire en fonction de ses variables de décision, alors on parle de Programmation linéaire mixte.

1.2.2 Complexité et NP-difficulté

La résolution du problème d'optimisation peut être considérée comme un problème de décision dont l'objectif est de décider s'il existe une solution qui satisfait la fonction objective (supérieure ; respectivement inférieure) ou égale à une valeur donnée). Donc la complexité d'un problème d'optimisation en temps ou en espace est équivalente à la complexité du problème de décision qui lui est associée. S'il existe un algorithme de temps polynomial pour résoudre Un problème, on dit que ce problème est dit résolvable ou facile.

Si aucun algorithme de temps polynomial existe pour résoudre le problème, dans ce cas ce problème est dit non traitable ou difficile. En général, on dit que le problème d'optimisation est NP-difficile si le problème de décision est classifié en tant que NP-complet, alors étant donné la notation grand-O O(...) est souvent utilisée. On peut dire pour Un algorithme qu'il est polynomial si et seulement si sa complexité est bornée par un polynôme P(n), i.e. f(n) =O(P(n)). Un algorithme est dit exponentiel si sa complexité ne peut pas être bornée par un polynôme.

$$P(n) = a^n$$

avec a > 1 une constante.Un aspect important de la théorie de complexité consiste à classer les problèmes en différentes classes.

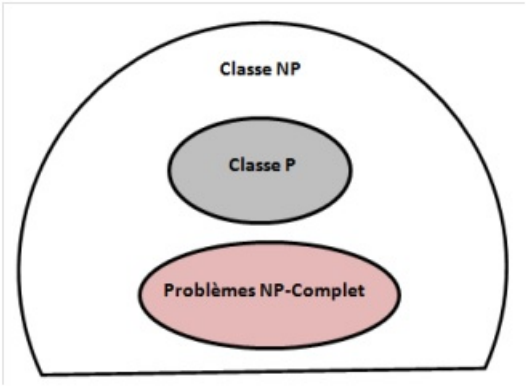


Figure 1.1 – Complexité des problèmes

Classe P : On trouve dans cette classe les problèmes de décision pour lesquels il existe des algorithmes polynomiaux qu'ils traitent.

Classe NP : Dans cette classe on trouve les problèmes de décision pour lesquelles il existe des algorithmes polynomiaux non déterministes qu'ils traitent.

Classe NP-complet : Un problème de décision A NP est NP-complet ssi B $\in$ NP/ BA peut se transformer plynômialement au problème A.

Le problème de l'optimisation combinatoire tient une place très importante dans des domaines nombreux. En effet, dans la littérature les problèmes d'optimisation combinatoire sont en général des problèmes très compliqués à résoudre et la plupart des problèmes de notre vie peuvent être représentés sous la forme d'un problème d'optimisation pour pouvoir les traiter. Un problème d'optimisation est représenté par un ensemble d'instances, dont chaque instance correspond à des données particulières du problème à résoudre. La plupart des problèmes d'optimisation combinatoire sont classifiés en tant que des problèmes NP-difficiles pour lesquels il n'existe pas d'algorithmes de résolution efficaces pour toutes les instances. De plus, dans nos jours l'étude de ces problèmes rend leur résolution plus facile.

Généralement, la résolution de tout problème combinatoire par une méthode d'optimisation, passe par deux étapes successives :

— L'analyse du problème qui englobe sa représentation informatique ou codage surtout le codage des solutions potentielles et codage d'opérateurs de recherche adaptés.

— Le choix d'une approche de résolution ou approche de recherche, qui prend en entré la représentation du problème dans un espace de recherche et en sortie elle

donne une solution ou un ensemble de solutions optimales par rapport à la fonction d'évaluation.

Avant de choisir une méthode d'optimisation, on doit d'abord établir un modèle de problème, c'est-à-dire choisir la représentation des solutions potentielles au problème puis de définir l'ensemble des propriétés requises qui vont permettre de sélectionner la méthode d'optimisation la plus adaptée pour résoudre le problème selon le codage choisi[64].

1.2.3 Les modèles d'optimisation classique

Un problème d'optimisation peut être défini par le couple (S, f), où S est l'ensemble des solutions réalisables et $f : S$ - R la fonction d'optimisation. La fonction objective attribue à chaque solution $s \in S$ de l'espace de recherche un nombre réel indiquant sa valeur. La fonction objective f permet de définir l'ordre total de relations entre n'importe quelle paire de solution dans l'espace de recherche.

Par conséquent, l'objectif principal est de trouver la solution optimale globale S^*. Pour obtenir plus de choix, le problème peut aussi être défini comme trouver toutes les solutions optimales globales.

Différentes familles de modèles d'optimisation sont utilisées pour résoudre les problèmes de prise de décision.

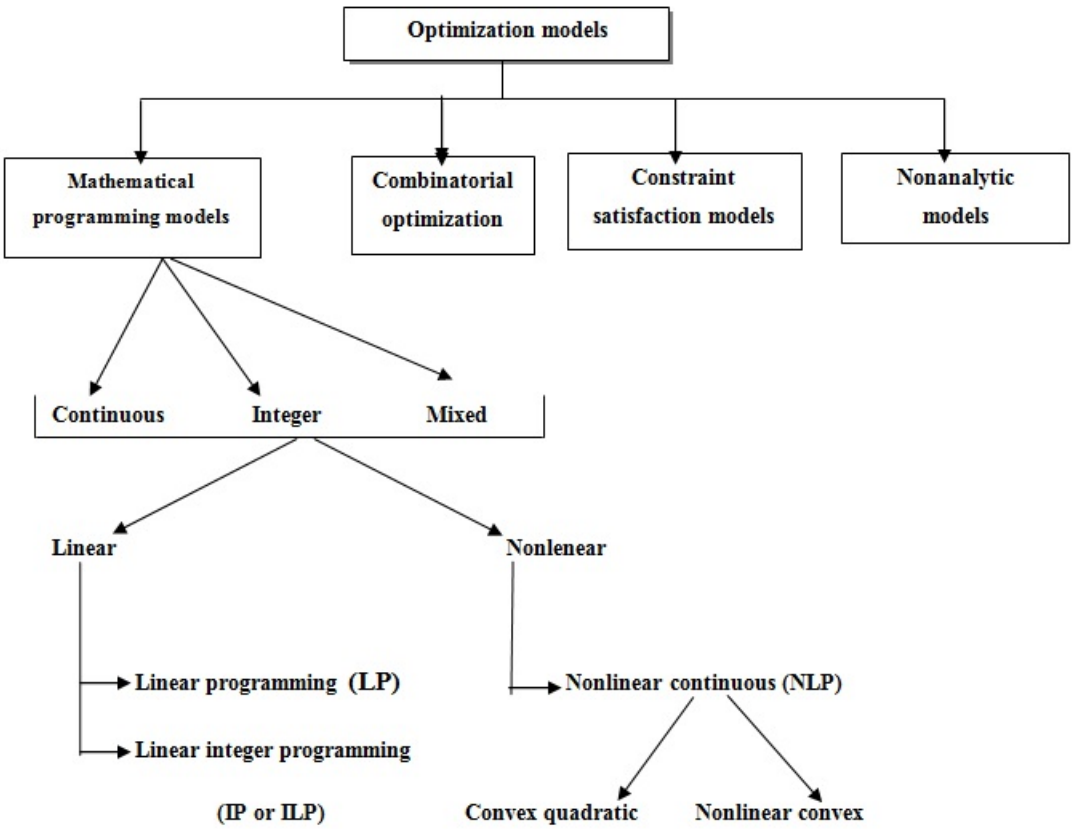


Figure 1.2 – Les modèles d'optimisation classique

Les modèles qui ont montré leurs succès sont basés sur la programmation mathématique et programmation par contraintes.

— Un modèle couramment utilisé dans la programmation mathématique est la programmation linéaire (LP), qui peut être formulée comme suit :
Min c • x
subject to

$$A \bullet x \quad b$$
$$x \quad 0$$

Où x est un vecteur de variables de décisions continues, et c , b (resp. A) sont des vecteurs constants (resp. matrice) de coefficients. En effet, pour les problèmes d'optimisation linéaire continue, les algorithmes exacts sont efficaces ,tels que les méthodes de type simplexe ou les méthodes existantes de point intérieur,donc il

n'existe pas de raison d'utiliser les méta-heuristiques pour résoudre les problèmes PL continus.

— Modèles de programmation non linéaire (NLP) ,les problèmes où la fonction objective et/ou les contraintes sont non linéaires. En général Un problème d'optimisation non linéaire continu consiste à minimiser une fonction f : S $R^n R$ dans un domaine continu. Les Modèles continus et non linéaires sont, cependant,beaucoup plus difficiles à résoudre, même s'il y a beaucoup de possibilités de modélisation qui peuvent être utilisées pour linéariser un modèle. Certains algorithmes exacts efficaces peuvent être utilisés pour résoudre des problèmes petits ou modérée. Malheureusement, certaines propriétés de problèmes tels qu'elles sont de haute dimension multimodale, épistasie (qui signifie l'interaction de paramètres), qui rendent ces approches traditionnelles impuissantes.

Les méta-heuristiques sont de bons candidats pour cette classe de problèmes pour résoudre les cas modérés et grands [64].

1.3 Classification des méthodes d'optimisation

Pour définir une méthode de résolution on doit d'abord étudier la complexité du problème, en général les problèmes peuvent être résolus par un algorithme exact ou un algorithme approximatif, dans les algorithmes exacts les solutions sont optimales et leur optimalité est garantie. Dans le cas des problèmes NP-complets, les algorithmes exacts ne possèdent pas un temps polynomial (à moins que P = NP). Les algorithmes approximatifs (ou heuristiques) génèrent des solutions de grande qualité dans un temps raisonnable, mais il n'y a aucune garantie de trouver une solution optimale globale.

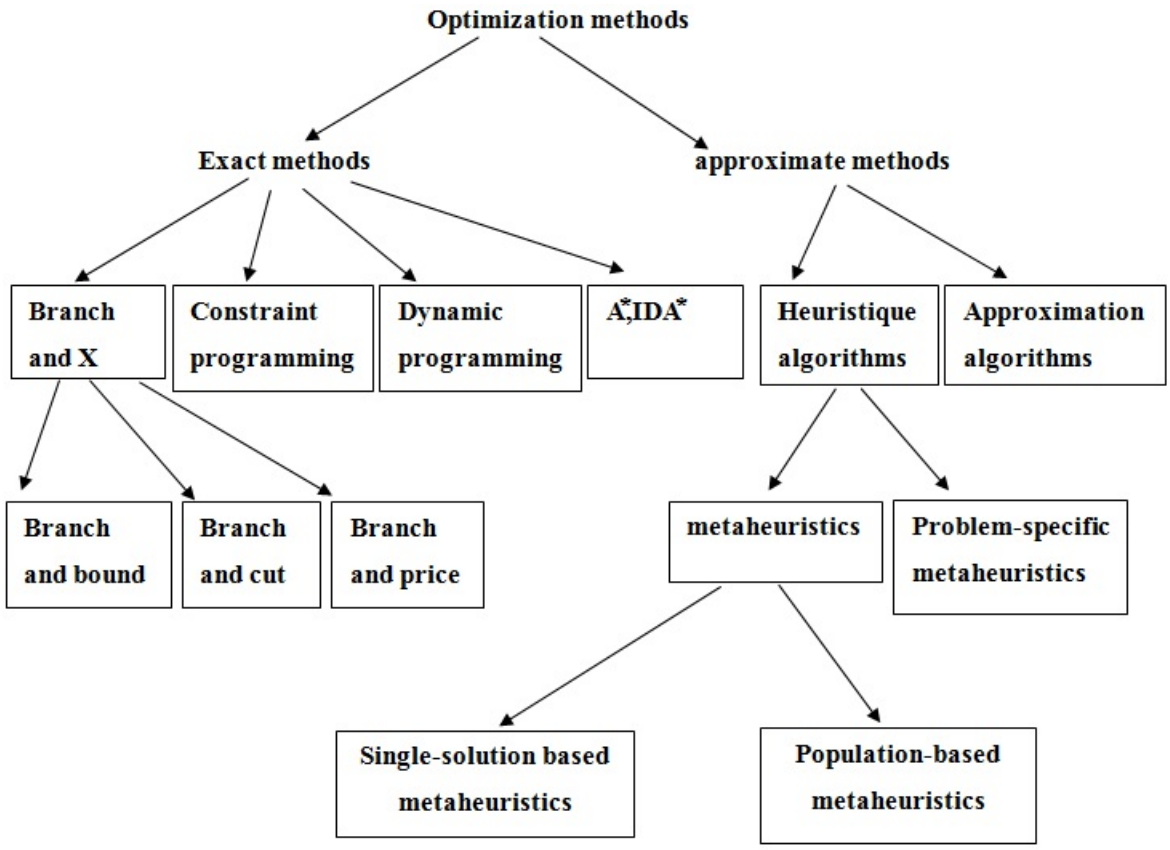


Figure 1.3 – Classification des méthodes d'optimisation classique

1.4 Les méthodes exactes

Dans Les méthodes exactes toutes les solutions de l'espace de recherche sont énumérées implicitement en utilisant des mécanismes qui détectent des échecs (calcul de bornes). Grâce à Ces méthodes on peut trouver des solutions optimales. Mais ces méthodes s'avèrent, malgré les progrès réalisés, plutôt inefficaces à mesure que la taille du problème devient importante. Dans cette classe des méthodes exactes,on peut trouver les algorithmes classiques suivants : programmation dynamiques ,la famille des algorithmes branche et X (branch and bound, branche et cut, branche et prix) , la programmation avec contrainte , et une famille de A^* d'algorithmes de recherche (A^*, IDA*).

1.5 Algorithmes approchés

On peut les appeler aussi incomplets à cause du fait qu'elles n'explorent intentionnellement qu'une partie de l'espace des configurations, par conséquent, elles peuvent ne pas trouver la solution optimale dans certains cas. Les algorithmes approchés, peuvent

être divisés en deux sous-classes d'algorithmes : les algorithmes d'approximation et les algorithmes heuristiques.

Les heuristiques se définissent par le fait de trouver des solutions "bonnes" sur les instances de problèmes de grande taille. Ils permettent d'obtenir des performances acceptables à des coûts acceptables dans un large éventail de problèmes. En général , heuristique ne dispose pas d'une garantie d'approximation sur les solutions obtenues.

Ils peuvent être classés en deux catégories : les heuristiques spécifiques et les méta-heuristiques. Les heuristiques spécifiques sont adaptés et conçus pour résoudre un problème spécifique et/ou une instance spécifique.

Les méta-heuristiques sont des algorithmes polyvalents qui peuvent être appliqués pour résoudre presque n'importe quel problème d'optimisation. Elles peuvent être considérées comme des méthodologies générales de niveau supérieur qui peuvent être utilisées en tant que stratégie de guidage dans la conception des heuristiques sous-jacentes pour résoudre les problèmes d'optimisation spécifiques.

1.5.1 Les Meta-heuristiques

Les métas-heuristiques sont souvent inspirées des systèmes naturels, tel qu'en physique (la méthode de recuit simulé), en biologie (les algorithmes évolutifs comme les GAs et les stratégies évolutionnaires) ou encore en étiologie (les algorithmes de colonies de fourmis).

On peut classer ces métas-heuristique en deux groupes :

— Les méthodes à solution unique ont tendance à intensifier la recherche en exploitant une partie de l'espace de recherche en effectuant une décente vers une solution optimale.

— Les méthodes à population de solutions ont plutôt tendance à la diversifier en explorant différentes parties de l'espace de recherche.

1.5.2 Méthodes basées population

L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité. Il existe plusieurs stratégies d'évolution de cette population amenant à des méthodes telles que les colonies de fourmis.

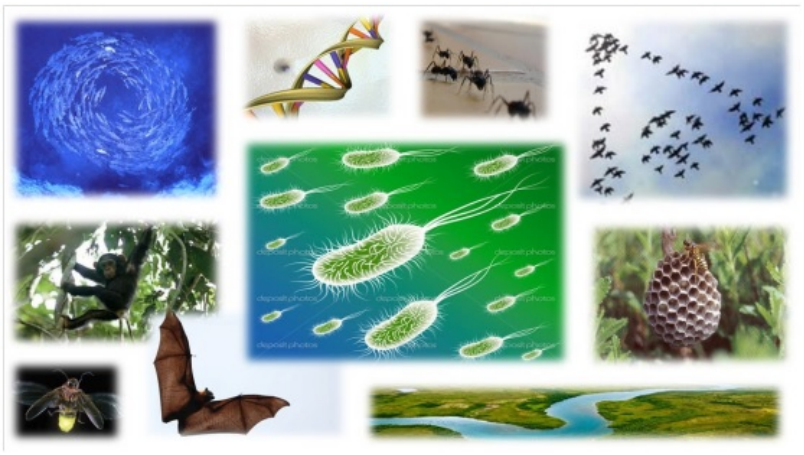


Figure 1.4 – Les approches inspirées de la nature

Aujourd'hui, les méta-heuristiques (les algorithmes évolutionnaires, l'optimisation par colonie de fourmis, l'optimisation par essaim de particules, les systèmes d'abeilles, . . . etc.), sont utilisés pour optimiser les problèmes complexes de grandes tailles. Contrairement aux méthodes exactes, ces méthodes offrent des solutions sous optimales acceptables dans des temps raisonnables, car il n'y a aucune garantie théorique que ces approches arrivent à la solution globale.

Par ailleurs, les méta-heuristiques sont très efficaces dans la résolution de nombreux problèmes à savoir la robotique, l'aérodynamique, l'automobile,et l'électronique et la bioinformatique,....etc.

Plusieurs classes de méta-heuristiques peuvent être identifiées :

— Les méta-heuristiques inspirées de la nature ; exemple les algorithmes évolutionnaires et des systèmes immunitaires artificiels de la biologie ;les colonies des fourmis et des abeilles , et l'optimisation par essaim de particules

— Les méta-heuristiques non inspirées de la nature (exemple le recuit simulé).

— Les méta-heuristiques avec mémoire.

— Les méta-heuristiques sans mémoire.

— Les méta-heuristique basé solution unique.

— Les méta-heuristiques basées population.

1.5.3 Les approches inspirées de la nature

Les paragraphes suivants présentent un aperçu général de quelques algorithmes d'optimisation inspirés de la nature basés sur l'évolution (GA, GP, Evolutionary Strategy) et

de quelques algorithmes d'optimisation inspirés de la nature basés sur l'intelligence distribuée (PSO, ACO, BFO, FA, AIS).

Voici un pseudo code d'une méta-heuristique
Algorithme illustre le modèle de haut niveau de P-méta-heuristiques

Algorithm 1: Un pseudo code d'une méta-heuristique

Debut ;
P = P0 ; / * Génération de la population initiale * /;
t = 0;
repeat
  Générer (P't) ; / * Génération d'une nouvelle population * /;
  Pt + 1 = Sélectionnez-population (Pt ∪ P't) ; / * Sélectionner nouvelle population * /;
  T=t+1 ;
until *l'arrêt critères satisfait*;
Sortie : La meilleure solution (s) trouvé;
Fin

1.5.3.1 Les approches d'optimisation basées sur l'intelligence distribuée

Figure 1.5 – Intelligence distribuée

L'optimisation par essaims particulaire :

L'optimisation par essaims de particules (en anglais Partical Swarm Optimization PSO) peut être considérée comme une métaphore de calcul et de comportement pour résoudre les problèmes distribués, l'origine de cet algorithme s'inspire du monde du vivant grâce à des observations faites lors des simulations informatiques de la nature fournis par des insectes sociaux comme les fourmis, les termites, les abeilles, les guêpes et aussi par les essaims, les troupeaux en se basant généralement sur la reproduction d'un comportement social. Elle est inventée par Russel Eberhart et James Kennedy [39] en 1995. Ces simulations sont basées sur la capacité des individus d'un groupe en mouvement à conserver une distance optimale entre eux et à suivre un mouvement global par rapport aux mouvements locaux de leur voisinage. Elle s'appuie sur le concept d'auto-organisation. Les particules peuvent converger progressivement vers l'optimum global grâce à des règles de déplacement très simples (dans l'espace des solutions).

L'intelligence distribuée possède les caractéristiques suivantes :

— **Autonomie :** Les individus contrôlent leur propre comportement.

— **Adaptabilité :** Lorsque les individus peuvent détecter des changements dans l'environnement dynamique ensuite adapter de manière autonome leur propre comportement à ces nouveaux changements.

— **Évolution :** La capacité d'utiliser des groupes composés de quelques-uns et ensuite évolués à des milliers d'individus ayant la même architecture de contrôle.

— **Flexibilité :** La capacité d'enlever ou remplacer ou ajouter dynamiquement un individu.

— **Robustesse :** Il n'existe pas de coordination centrale, ce qui signifie qu'il n'y pas de point de défaillance unique. Le système d'essaim permet la redondance, ce qui est essentiel pour la robustesse.

— **Le parallélisme massive :** Le fonctionnement du système d'essaim est massivement parallèle et distribué. Les individus accomplies les mêmes tâches au sein du groupe. L'architecture de "Intelligence Distribuée" peut-être considérée comme une architecture SIMD.

— **L'auto-organisation :** L'intelligence exposée n'est pas présente chez les individus, mais émerge plutôt d'une certaine manière sur l'ensemble de l'essaim.

Le principe de PSO :

La population dans L'optimisation par essaim de particules est appelée essaim, ces individus sont originalement placés aléatoirement dans l'espace de recherche, dans chaque itération de l'algorithme, les individus se déplacent dans l'espace de recherche en prenant en compte leur meilleure position et également la meilleure position de son voisinage [62] :

Avantages et inconvénients :

Quelques avantages et inconvénients de cet algorithme d'optimisation peuvent être résumés dans les points suivants :

— a) Avantages [58] :

  — PSO n'utilise pas de croisement ni de mutation. La recherche peut être effectuée simplement par la vitesse de la particule.

  — Le calcul de PSO est très simple. En comparaison avec les autres algorithmes, elle occupe la plus grande capacité d'optimisation et elle peut facilement être remplie.

— b) Inconvénients :

  — L'algorithme converge facilement vers l'optimum local, ce qui provoque le moins exact à la régulation de la vitesse et la direction.

  — PSO utilise plusieurs paramètres en entrée qui sont déterminé dans l'étape de L'initialisation et qui influent directement sur la convergence vers l'optimum global.

  — PSO est simple à utiliser.

  — La parallélisation du PSO peut être facile.

Algorithme de colonie de fourmis ACO

Dans L'optimisation par colonie de fourmis, les fourmis artificielles d'une colonie coopèrent pour trouver de bonnes solutions à des problèmes d'optimisation difficiles. La conception des algorithmes ACO est basée sur La coopération d'un ensemble d'agents simples qui se communiquent indirectement par la stigmergie. Les bonnes solutions sont une propriété émergée de l'interaction coopérative des agents. Une fourmi artificielle

ACO construit progressivement une solution en ajoutant des composants de solutions définies à une solution partielle en cours de construction [23].

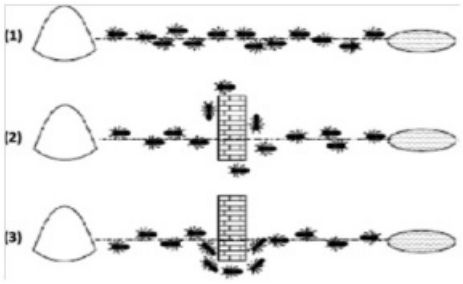


Figure 1.6 – Recherche des fourmis du plus court chemin de la nourriture vers le nid en suivant la phéromone.

On peut définir l'algorithme ACO par trois procédures : procédure de Construction de Solutions Fourmis, procédure de Mise-à-jour-Phéromones et Opérations-Démons.

Algorithme de base ACO
Construire la Solutions Fourmis
Mise-à-jour du Phéromones
Opérations Démons / optionnel
End

Optimisation par recherche de nourriture bactérienne BFO

Cette optimisation est inspirée par le sabot exposé par des comportements de recherche de nourriture bactérienne [65]. Les bactéries se rassemblent toujours dans les zones riches en éléments nutritifs par une activité appelée "Chimiotaxie". La méthode suivie par les bactéries pour nager est de tourner flagelles actionnés par un moteur réversible, incorporé dans la paroi de la cellule (E. coli à 8 à 10 flagelles placé de façon aléatoire sur le corps de cellule). A la fin, quand tous les flagelles tournent dans le sens antihoraire, ils forment un compact propulsant hélicoïdalement la cellule suivant un chemin hélicoïdale, qui est appelé "Courir" (Run). Dans le cas ou tous les flagelles tournent dans le sens horaire, ils tirent tous sur la bactérie dans des directions différentes, ce qui fait culbuter la bactérie (Tumble) [57].

1.5.3.2 Les approches basées sur les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des méta-heuristiques à population de solutions dont la population d'un problème d'optimisation est constitué par l'ensemble de solutions candidates de ce problème d'optimisation donc les solutions sont définies comme des individus d'une population. L'évolution de la population conduit à trouver de meilleures solutions. Certains mécanismes inspirés de l'évolution biologique sont appliqués pour l'évolution de la population [41].

Plusieurs méthodes évolutionnaires ont été proposées on peut citer les GAs, les stratégies d'évolution (ES), la programmation génétique (GP), la programmation évolutionnaire (EP) et l'évolution différentielle. Ces approches sont des algorithmes évolutionnaires très populaires [4, 28], une description de ces algorithmes est donnée dans ce qui suit.

Stratégies d'évolution ES

La stratégie d'évolution (ES) a été développé à l'université technique de Berlin par I. Rechenberg et H.-P. Schwefel dans les années 1960. Dans Les SE les caractéristiques de l'individu sont les paramètres à être optimisés. En outre, cet ensemble de caractéristiques est représenté par des nombres réels sous forme d'un vecteur de nombres réels de dimension fixe.

Les SE génèrent une population de descendants (de dimension μ) à partir de laquelle des individus sont sélectionnés aléatoirement afin de générer une population de descendants (de dimension $\lambda\mu$). A partir d'une population de parents (de dimension μ) en sélectionnant aléatoirement des individus. Ensuite on applique l'opérateur génétique classique mutations pour modifier ces individus , qui consistent à ajouter une valeur générée aléatoirement selon une fonction de densité de probabilité paramétrée. Dont l'évolution de cette fonction dans le temps se fait de la même façon que l'évolution des paramètres caractérisant les individus. Afin de former la nouvelle population, μ individus sont sélectionnés (approche (μ,λ)) parmi les μ meilleurs individus des λ descendants, ou (approche $(\mu+\lambda)$) parmi les μ meilleurs individus des μ parents et λ descendants. Des stratégies d'évolution imbriquées sont intégrées dans Les SE moderne[29].

Les algorithmes génétiques :

L'idée des GA est vaguement inspirée des chaînes d'ADN, qui composent tout organisme vivant. Leur but principal est d'améliorer le profit des solutions candidate dans

une population contre une fonction de coût du domaine du problème. Les GAs sont basés sur les opérateurs génétiques de recombinaison (croisement et de la mutation) appliqués sur les individus de la population, selon une fonction objective, ce qui amène à une amélioration des solutions [11] [40].

Les opérateurs génétique :

L'objectif principal de la sélection, le croisement et la mutation est d'améliorer les chromosomes (solutions) entre deux itérations pour parcourir l'espace de recherche[40]. Les étapes d'un GA son comme suivent :

1. Codage du système.
2. Génération de la population initiale. Répéter
3. L'évaluation des chromosomes, et calculer une valeur d'adaptation pour chaque chromosomes (fitness).
4. La sélection des meilleurs chromosomes se fait selon la fonction objective ,et fitness.
5. Appliqué les opérateurs de recombinaison (Croisement et mutation) des chromosomes parents.
6. testé si le critère d'arrêt est vérifié ou non, s'il est satisfait, nous quittons sinon, répéter à partir du point 3.

Dans ce mémoire on s'intéresse aux GAs donc il sera plus détaillé dans ce qui suit.

CHAPITRE 2

SYSTÈMES MULTI-AGENTS

Ce chapitre est consacré aux notions d'agents, des systèmes multi-agents et quelques protocoles de communication entre agents. Quelques méthodes de développement des systèmes multi-agents sont présentés également dans ce chapitre.

2.1 Introduction

Dans nos jours, les dispositifs de calcul deviennent de plus en plus puissants, et ils sont considérés comme des entités individuelles possèdent une certaine capacité de répondre à beaucoup d'exigences de l'être humain.

Cela ouvre de nouveaux défis considérables pour les concepteurs de logiciels et les utilisateurs. Par ailleurs, la complexité des problèmes rencontrés ne cessent d'accroître en fonction de ces nouvelles technologies.

En raison de la complexité croissante, il ne sera pas possible de concevoir explicitement et gérer ces systèmes dans les moindres détails et d'anticiper toutes les configurations possibles. Ainsi, et à cause de l'hétérogénéité dans ces problèmes, nos systèmes techniques devront agir de façon plus indépendamment, flexible et autonome donc on s'intéresse à développer de nouvelles machines basées sur la distribution du calcul et la décentralisation du contrôle plus adapté à ce genre de problème. L'objectif principal de ces machines est de réduire le temps de calcul grâce au parallélisme, et de distribuer des compétences et des connaissances.

Beaucoup d'approches proposées qui doivent guider les concepteurs dans le processus de logiciels pour ces problèmes. Parmi les méthodes les plus efficaces, on trouve les MASs.

Un MAS est un concept relativement nouveau dans l'intelligence artificielle distribuée. Leurs objectifs est de réaliser un comportement collectif produit à travers les interactions entre plusieurs agents autonomes et flexibles pour assurer la parallélisation dans le comportement global du système [49].

2.2 Définitions

2.2.1 Agent et ses caractéristiques

Selon Jenings [36] un agent est défini comme un système informatique placé dans un environnement,et qui exécute ces tâches de façon autonome sur cet environnement pour satisfaire ses objectifs.

Ferber [26], définit un agent comme étant une entité physique ou virtuelle

— qui est situé dans un environnement.

— qui est en coopération avec d'autres agents.

— qui possède un ensemble d'objectifs individuels ou une fonction de satisfaction.

— qui possède ces propres ressources.

— qui possède une capacité de percevoir (mais de manière limitée) son environnement.

— qui dispose d'une représentation partielle de son environnement (et éventuellement aucune),

— qui est capable d'offrir des services en tenant compte de ces compétences.

— qui est capable à se reproduire,

— dont en fonction de ces ressources et ces compétences , et en tenant compte de sa perception, et de sa représentations tend à satisfaire ses objectifs.

Un agent a les caractéristiques suivantes :

— **L'autonomie :** Un agent autonome est un agent qu'est plus ou moins indépendant des autres agents et de l'utilisateur. Pour vérifier cette caractéristique, un agent doit avoir ses propres objectifs et être apte de décider pour résoudre des conflits internes [18].

— **La réactivité :**Des agents sont dits réactifs s'ils peuvent répondre aux stimulus externes qui se sont effectués dans leur environnement.

— **La communication :** Un agent doit être capable de communiquer avec les autres agents.

— **La pro-activité :** Des agents sont dits pro-actifs s'ils peuvent agir sans même que leur environnement ait changé.

— **Planification :** La capacité de construire un plan d'action à réaliser pour poursuive un certain but.

— **Intelligence :** On appelle agent intelligent un agent qui possède des capacités de raisonnement et d'apprentissage, et être apte de planifier ses propres actions, et aussi tenir compte de celles des autres agents.

— **Rationalité :** Un agent rationnel possède des critères d'évaluation de leurs actions, et afin d'atteindre un objectif, il choisit selon ces critères les meilleures actions.

— **Apprentissage :** Les agents doivent évoluer et améliorer leurs connaissances, ainsi, adapter ou changer leurs comportements face à des situations similaires et cela en fonction de leurs expériences passées. Cette caractéristique est principalement réservée aux agents qualifiés d'intelligents.

— **Prise de décision :** Un agent doit être capable de prendre une décision sur le choix du but à satisfaire en premier, et l'action qui permet de l'atteindre, ainsi que pour les autres buts.

2.2.2 Classes d'Agents

Il existe deux grandes classes d'agents qui sont [66] :

2.2.2.1 Les agents réactifs

Les agents sont très simples et l'intelligence est le résultat de l'interaction entre ses agents, leur réaction à des événements produits en entrées est rapide et il n'est pas nécessaire d'inclure un raisonnement de haut niveau.

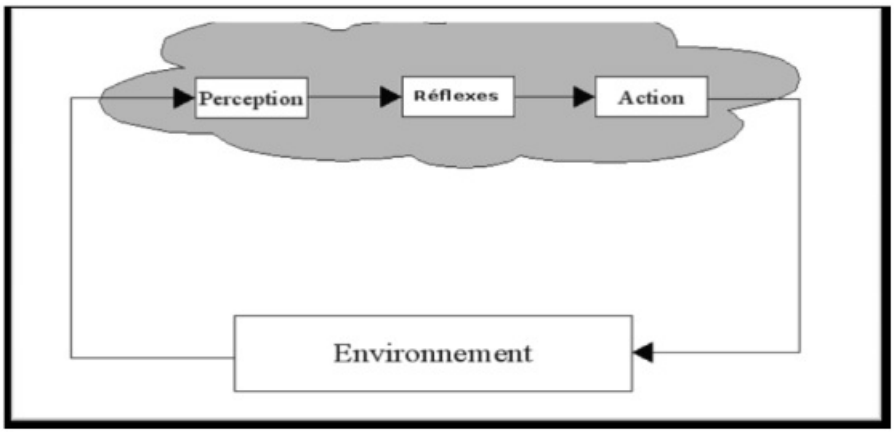


Figure 2.1 – Agent réactif [66]

2.2.2.2 Les agents cognitifs

Un système cognitif est basé sur la coopération des systèmes experts, qui sont capables de réaliser des tâches complexes [66] [32] [37]. Chaque agent possède ses propres fonctions : le raisonnement, aptitude à traiter des informations diverses liées au domaine d'application, et d'informations relatives à la gestion des interactions avec d'autres Agents et l'environnement, prendre des décision, l'apprentissage.

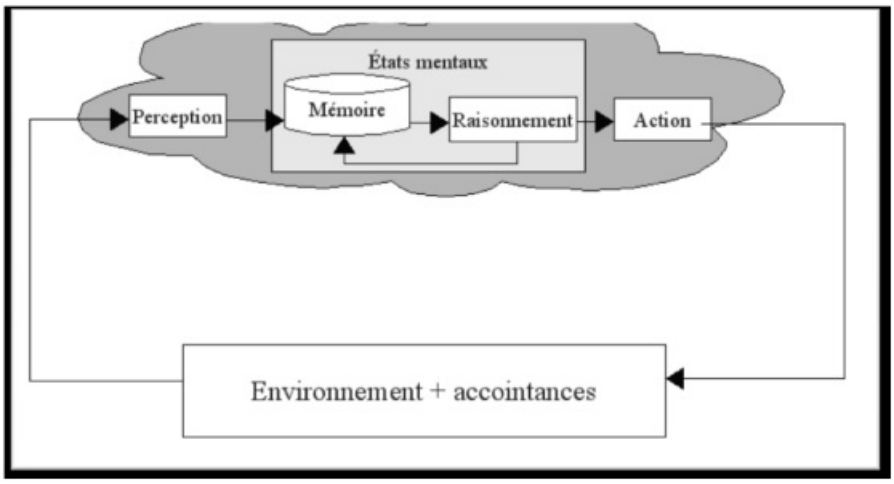


Figure 2.2 – Agent cognitif

2.3 Système multi-agents

Selon Ferber [26], un MAS est un système constitué de :

— Un environnement E commun.

— Un ensemble d'objets O, qui sont placés dans un environnement E, c'est-à-dire que chaque objet possède une position dans E. Ces objets peuvent être manipulé(modifié, perçus, créés, détruits) par les agents.

— Un ensemble A d'agents inclus dans E, et qui sont des objets spécifiques représentant l'entité active du système.

— Un ensemble de relations R qui inter-connectent entre objets et/ou agents.

— Un ensemble d'opérations Op qui aident les agents à manipuler les objets de O.

— Des opérateurs dont leur rôle est de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.

2.3.1 Domaines d'application

En pratique, il existe cinq grandes catégories d'applications des MAS : la résolution de problèmes au sens large, la robotique distribuée, la simulation multi-agents, la construction de mondes hypothétiques et la conception kénétique de programmes.

2.3.2 Intérêts des MAS

L'approche MAS est justifiée par les avantages suivants [25] :

- Fiabilité, due à la redondance.

- La distribution d'un système en sous-systèmes en coopération.
- La possibilité de réutiliser les systèmes déjà existant ,en intégrant avec des nouveaux systèmes
- La vitesse, avec le parallélisme par la représentation des connaissances de façon distribuée.
- La simulation de nombreux mécanismes et phénomènes naturels.

2.3.3 Interaction entre agents

L'intervention des schémas d'interactions sophistiquées est essentielle pour les MAS. Il faut noter que, sans interactions, la sociabilité des agents n'est pas possible. Les phénomènes d'interaction consistent à mettre en relation dynamique de deux ou plusieurs entités(agents, éléments de l'environnement) du système à travers un ensemble d'actions. Les interactions sont fondées sur la coopération, la coordination et la négociation.

2.3.3.1 La coordination

La coordination est définie par un ensemble d'actions supplémentaires qu'il est nécessaire d'accomplir pour réaliser une tâche commune, et peut-être considérée comme une négociation entre des agents possédants des buts différents[48].

2.3.3.2 La coopération

La coopération est la forme générale d'interaction ,elle est considérée comme un comportement adopté par les agents pour travailler en groupe. Elle permet à un agent de :

- Modifier les connaissances globales du système.
- Acquérir des informations venant des autres agents et les intégrer.
- Un agent peut intervenir pour aider les autres agents en interrompant son exécution[24].
- Partager les tâches qui sont difficiles à résoudre avec d'autre agents et prendre en compte localement les compétences des autres.

2.3.3.3 La négociation

Consiste à améliorer les accords et résoudre les conflits sur des points de vue communs des agents grâce au mécanisme de négociation c'est-à-dire l'échange structuré d'informations pertinentes.

2.3.4 La communication dans un MAS

Les agents cherchent à faire partager leurs croyances avec d'autres agents, grâce au principe d'interaction. Les agents peuvent interagir soit en communiquant de façon directe entre eux, soit de façon indirecte en modifiant leur environnement. Deux stra-

tégies fondamentales ont été exploitées pour exprimer la communication entre agents : Communication directe par envoi de messages ou communication via une structure de données partagées (base de données partagées appelée tableau noir ou “blackboard”) où on trouve les informations.
Il existe plusieurs types de protocoles de communication, les plus connus sont :

2.3.4.1 Protocole Knowledge query and manipulation language (KQML)

Le langage KQML [27] est un standard de communication de haut niveau qui a été proposé pour permettre la coopération entre les agents cognitifs. Ce langage est représenté par un ensemble de requêtes(messages) (appelés abusivement “performatifs”) et des règles de comportements proposés pour les agents qui reçoivent ces messages.

2.3.4.2 Protocole FIPA-ACL (Foundation for Intelligent Physical Agents)

Ces dernières années, pour les raisons que KQML manque cruellement de spécifications et de formalisation. Un autre langage plus riche sémantiquement ACL (pour Agent Communication Langage) à été inventé. C’est un standard FIPA(Fon-dation for Intelligent Physical agent), possède les mêmes caractéristiques de message que KQML, ACL est fondé sur la théorie du langage et a profité grandement des résultats de recherche de KQML.

2.3.4.3 La Plateforme Jade

Afin de rendre le développement des applications multi-agents plus facile, plusieurs plateformes ont été proposées. Jade (Java Agent developement framework) est un environnement d’implémentation et d’exécution des MAS, développée en Java par le laboratoire TILAB. Les MAS implémentés par Jade sont compatibles à la norme de FIPA [63]. Il est constitué de trois composantes fondamentaux :

— DF (DirectorFacilitor) fournit un service d’annuaire à la plate-forme.

— ACC (Agent Communication Channel) gère la communication entre les agents (Message Transport System du modèle de référence).

— AMS (Agent Management System) supervise l’enregistrement des agents, leur authentification, leur accès et l’utilisation du système.

Il faut mentionner que Toutes ses composantes sont des agents JADE.

1. **Les propriétés des agents implémentées sous Jade**

   Les agents Jade sont des classes héritées de la classe jade.core.Agent, et chaque agent possède, plusieurs comportements (élémentaires, cycliques, périodique, à retardement...). En général un agent ne peut vivre que dans l'un des conteneurs, il est caractérisé par six propriétés [1] :

   — **Autonomie**
   Chaque agent est indépendant des autres agents et possède son propre thread qui lui permet de contrôler ses actions et d'être capable de décider afin de réaliser ses propres buts.

   — **Réactivité**
   Chaque agent peut réagir et répondre aux stimulus externes de son environnement.

   — **Aspects sociaux**
   Un agent est capable d'interagir et de communiquer avec les autres agents de son environnement à travers le passage de messages asynchrones. La structure et la sémantique des messages sont définies par le standard FIPA.

   — **Dynamicité**
   La communication entre les agents se fait d'une manière dynamique en cours d'exéc-ution.

   — **Offre de service**
   Chaque agent possède un ensemble de services, qui peuvent être enregistré et modifié, et il peut aussi échanger les services dont il a besoin avec les autres agents.

   — **Mobilité**
   Chaque agent peut se déplacer d'un site à un autre en cours d'exécution pour être proche de données ou de ressources.

2. **Communication entre agents en JADE**

   Ces agents communiquent entre eux à travers le passage de messages asynchrones. Il n'y a pas de dépendances temporelles entre les agents. Un message est une instance de la classe ACLMessage appartenant au package jade.lang.acl sous la norme FIPA-ACL (Agent Communication Language) [20]. Chaque agent peut

émettre ou recevoir des messages,pour l'envoi de messages l'agent utilise la méthode Send (message), où message est un objet de la classe ACLMessage, et pour la réception du message il utilise la méthode Receive() comme suit :message = receive(). Un ACLMessage possède plusieurs propriétés les plus importantes sont les suivantes :

propriété	signification
sender	Expéditeur du message
receiver	Destinataire du message
content	Contenu du message
language	Description du langage du contenu de message
Ontology	Description du l'ontologie du contenu de message

Tableau 2.I – Quelques propriétés d'un message de la classe acl message du jade

3. **Comportements d'agents en JADE**

   La plateforme JADE est constituée d'un ensemble de "containers " pouvant être répartie sur un réseau et chaque container est composé d'un ensemble d'agents donc un agent peut effectuer différents comportements en parallèle. Un agent contient plusieurs comportements. On peut ajouter Un comportement à un agent par la méthode addBehaviour(). Et quel que soit le comportement on trouve toujours les deux méthodes : public void action() et public boolean done(). La première méthode qui contient les opérations à exécuter par le comportement, et la deuxième retourne un booléen indique si le comportement a terminé son exécution ou non. Avant l'exécution de la méthode action()on doit d'abord appeler La méthode onStart() et la méthode onEnd()et appelée après la méthode done().

2.4 L'avantage de la plate-forme multi-agents Jade

La plate-forme Jade présente plusieurs avantages dans l'implémentation d'un ordonnanceur de tâches dans un système embarqué :

- Jade possède certaines abstractions de comportement dans la modélisation des tâches qu'un agent peut effectuer et les agents instancient leurs comportements selon leurs besoins et leur capacité. Alors l'abstraction est garantie par Jade.

- Jade utilise une technique d'envoi de messages ACL entre les agents très efficace, flexible et transparente.

2.5 Conclusion

Dans la conception des MAS notre principal défi est de répartir l'ensemble des tâches à effectuer sur plusieurs entités appelées agents dont le travail se fait de façon collaborative pour réaliser un but commun. Les agents interagissent entre eux et effectuent une certaine coordination et regroupement de leurs solutions partielles pour fournir la solution globale, l'action et l'interaction dans Les MAS sont considérés comme des éléments moteurs de la structuration d'un système dans son ensemble.

CHAPITRE 3

SYSTÈMES EMBARQUÉS DISTRIBUÉS TEMPS RÉEL

Ce chapitre est consacré à une introduction aux systèmes embarqués distribués temps réel et l'ordonnancement des tâches temps réel sur une architecture mono-processeur et multiprocesseurs ainsi que les principales classes des algorithmes d'ordonnancement.

3.1 Introduction

La tendance actuelle dans des systèmes embarqués temps réel qui sont actuellement de grande importance et que l'on trouve dans l'automobile, l'avionique, la robotique, etc....

Le problème est de valider non seulement la correction des résultats mais surtout les dates de délivrance de ces résultats. Ces dates doit être respecté, car le retard est considéré comme une erreur et peut mener à des conséquences catastrophiques.

Afin de réaliser un tel système nous sommes obligés à résoudre les problèmes d'ordonnancement et d'allocation des tâches temps réel dures pour des architectures multiprocesseur hétérogènes et distribuées. Cependant, le problème est qualifié de problème NP-difficile.

Dans ce travail, nous nous intéressons au système de tâche sporadique. Plus précisément, nous proposons une nouvelle conception efficace de ces systèmes qui nécessitent des méthodes robustes, puissantes et précises pour optimiser leurs performances.

3.2 Définitions

3.2.1 Système embarqué

Un système embarqué [59] est une combinaison hardware et software qui ne possède pas des unités d'E/S et qui est destiné à réaliser une tâche. Il doit obéir aux contraintes suivantes :

— Le poids.

— La fiabilité (résistance aux pannes).

— Le coût.

— La consommation d'énergie.

— Temps de réponse (temps réel).

3.2.2 Système temps réel

On défini un système informatique temps réel [60] comme un un système informatique qui réalise sa tâche en temps réel, i.e son temps de réponse doit être inférieur au seuil défini par le constructeur.

3.2.3 Système distribué

Un système distribué est une ensemble d'ordinateurs qui apparaît du point de vue utilisateur comme un unique système informatique cohérent. Ces ordinateurs sont connectés en réseau ayant comme objectif l'exécution d'une tâche globale à laquelle chaque

ordinateur participe par ses propres traitements et les communications qu'il entreprend, sans même qu'il ait une représentation globale du système[28].

3.2.4 Systèmes embarqués temps réel

Ces systèmes embarqués temps réel peuvent être considérés comme étant des systèmes critiques pour la sécurité dans des applications telles que par exemple l'automobile, les équipements médicaux,... etc.

En général, la validité des résultats ne dépend pas seulement de la correction des traitements réalisés mais également de la date à laquelle les résultats de ces traitements sont délivrés.

3.2.5 Systèmes embarqués distribués

Un systèmes embarqués distribués est une variante décentralisée d'un système embarqué centralisé possédant un ensemble de contrôleurs (noeuds) indépendants qui sont inter-connecté entre eux soit par un seul bus partagé, une hiérarchie de bus ou soit par un réseau de communication généralement avec trois couches : la couche physique, la couche liaison et la couche application.

3.3 Systèmes embarqués distribués temps réels

3.3.1 Définition des systèmes embarqués distribués temps réels

Un systèmes embarqués distribué est dit un système temps réel s'il est capable de satisfaire ses contraintes temporelles.

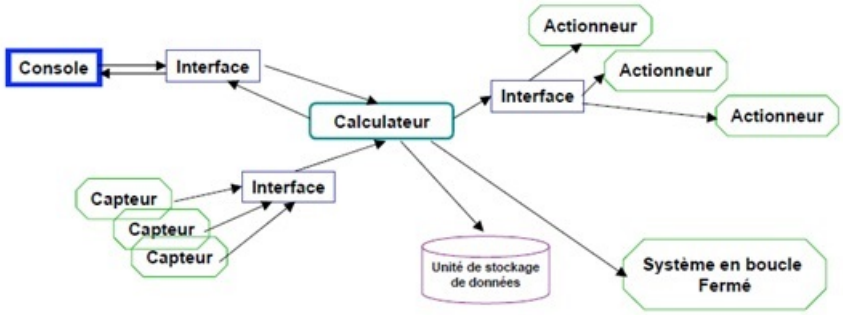


Figure 3.1 – Un modèle de système embarqué temps réel

- Le capteur : Son rôle est de transformer les caractéristiques physiques de son environnement en signaux électriques.
- L'actionneur : Est une machine dont ses entrées sont les sorties de l'ordinateur qui transforment ces signaux électriques en certains phénomènes physiques et par la suite l'applique sur son environnement [47].

3.3.2 Le deadline dans les systèmes embarqués temps réel

Contraintes de temps : La contrainte la plus importante est ce qu'on appelle l'échéance ou "deadline" (en français le délai), qui représente l'instant auquel la tâche doit terminer son exécution en bon terme et produire les résultats [47].

3.3.3 Systèmes temps réel durs et souple

- Systèmes temps réel à contraintes dures : Le non-respect d'une contrainte temporelle (temps de réponse $>$ deadline) peut avoir des résultats catastrophiques.

  Exemple : Le système de conduite de drones.

- Systèmes temps réel à contraintes souples : Le non-respect des contraintes temporelles ne peut occasionner des conséquences catastrophiques.

  Exemple : Le système de téléconférences.

Généralement, un seul système peut avoir une hybridation entre des sous-systèmes temps réel durs et souples à la fois [8].

3.4 Ordonnancement temps réel

3.4.1 Définition d'ordonnancement temps réel embarqué

Le problème d'ordonnancement des tâches est un problème d'affectation des tâches et qui détermine l'ordre d'allocation du processeur, d'une manière permettant d'optimiser un ou plusieurs objectifs. Tout en respectant les contraintes temporelles.

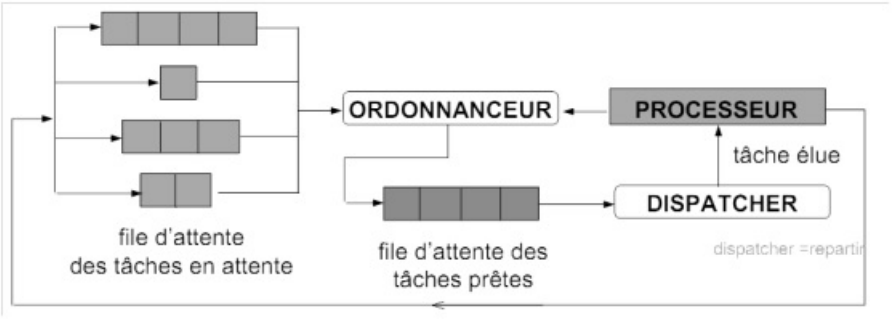


Figure 3.2 – L'ordonnancement des tâches temps-réel

Ordonnanceur : Alloue le processeur aux différentes tâches.

Dispatcheur : Implémente l'ordonnanceur (élection des tâches prêtes)[56].

La différence majeure entre l'ordonnancement dans les systèmes classiques et l'ordonnancement temps réel, est que dans le premier cas le but est d'augmenter le nombre de tâches qui consomme leur temps d'exécution par unité de temps et/ou de réduire leur temps d'attente moyen. Tandis que dans le deuxième cas le but est de respecter le deadline par toutes les tâches.

3.4.2 Un modèle de système

Un modèle de système est caractérisé par :

1. Un modèle de tâches exprimé par un ensemble de tâches $T = \{\tau 1 \dots \tau n\}$ ou n est le cardinale de T. Dont chaque τi possède un : deadlines (Di) ,une période (Pi) ,et un temps d'exécution (ci). Chaque tâche peut être soit périodique ou apériodique ou sporadique.
2. Un modèle de contraintes : temporelles(dure, souple), matériel(ressources), contrainte de précédences.
3. Un ou plusieurs processeurs.

3.4.3 Ordonnancement temps réel mono-processeur

Dans un système mono-processeur une seule tâche est exécutée à la fois et pour s'exécuter ,chaque tâche doit attendre la fin d'exécution des tâches qui la précède. Alors, on parle d'ordonnancement mono-processeur.

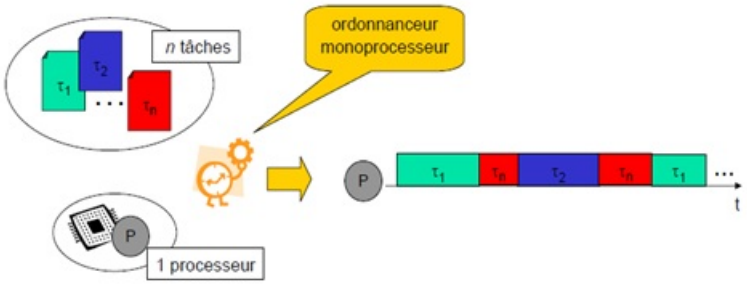


Figure 3.3 – Ordonnancement mono-processeur

3.4.4 Modèle de tâches temps réel

Un système est caractérisé par un ensemble T de n tâches temps réel, $T = \{\tau 1 \dots \tau n\}$, (voir [31]).

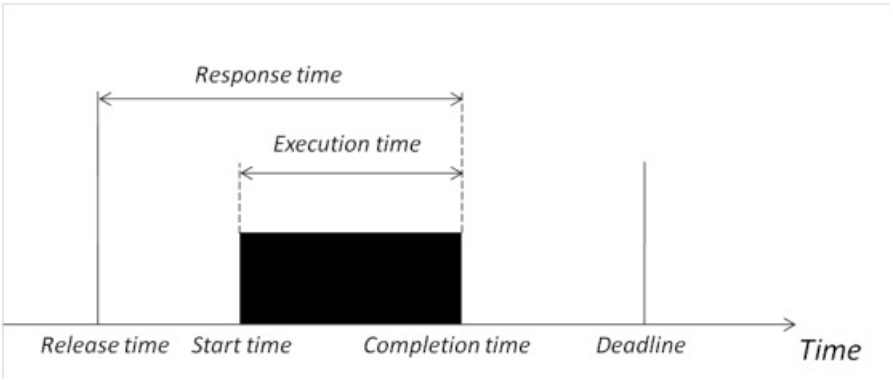


Figure 3.4 – Les attributs temporels de base d'une tache τi.

3.4.4.1 Caractéristiques d'une tâche temps réel

Une tâche est composée d'une séquence ordonnée indivisible d'instructions qui vont s'exécuter sur un processeur. Fréquemment les caractéristiques des tâches temps réel les plus utilisées sont : (tous ces paramètres sont illustrés sur la Figure 3.5) :

- Ci (Computing time) :Durée d'exécution maximale d'une tâche ti. Ce paramètre est considéré comme le pire cas des temps d'exécution (WCET pour Worst Case Execution Time).
- Di (Deadline) : C'est l'échéance ou la date au plus tard. Il signifie le temps avant lequel la tâche doit terminer son exécution et produire des résultats et dont le non respect provoque une violation de contrainte temporelle. Deux types de deadlines

existent :Deadline absolu D (ti) +R (ti) (à compter de la date 0) ,Deadline relatif D (ti) (à compter de la date d'arrivée)

- Ti (Task period) : C'est la durée d'exécution d'une tâche,c.a.d la séparation entre les temps d'arrivées successives générés par la tâche(valide uniquement pour les tâches périodiques).
- Ri (Ready time ou Release time) : C'est l'instant auquel la tâche Ti est prête à être exécutée.
- Si (Start time), Ei (End time) : Sont respectivement la date de début et de fin d'exécution sur le processeur.
- li (Laxity) : C'est la plus grande durée pendant laquelle la tâche peut ne pas s'exécuter sur le processeur sans rater son échéance.
- Temps de réponse : Est la durée de temps qui sépare entre l'activation de la tâche et la terminaison.
- Relations de précédence entre les tâches : Une tâche précède une autre tâche, si la deuxième tâche ne doit commencer son exécution que si et seulement si la première termine. Les précédences entre les tâches sont impliquées par des attentes de messages, ou de synchronisation.
- Contraintes de ressources : Autres que le processeur, tels que les périphériques d'entrées/sorties, les réseaux, des fichiers et des bases de données.
- Partage des données : Les tâches doivent souvent partager leurs résultats entre eux [62].

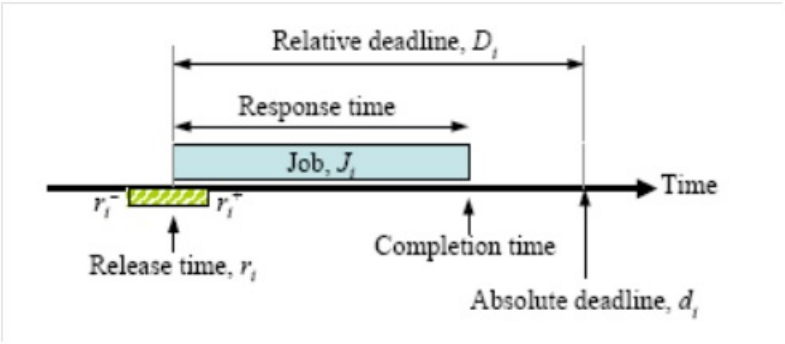


Figure 3.5 – Le deadline absolu et le deadline relative

3.4.4.2 Types de tâches temps réel

Il existe trois types de tâches temps réel : périodiques, apériodiques et sporadiques :

- Tâche périodique : Comme le nom périodique indique une tâche périodique est celle qui est activée (libérée) régulièrement à un taux fixe (périodes). Normalement, les tâches périodiques ont des contraintes qui indiquent que des instances d'elles doivent être exécutée une fois par période : $Ti(\Phi i, Ci, Pi, Di)$, où $\Phi i \geq 0$, $Ci > 0$ et $Ci \leq min(Pi, Di)$, $Pi > 0$, et $Di > 0$. Ci son temps d'exécution au pire cas ou WCET (Worst Case Execution Time), Pi sa période d'exécution et Di son échéance relative, c'est-à-dire le temps maximal dont on dispose pour exécuter la tâche.
  On distingue deux types de tâches périodiques :

- Tâches périodiques synchrones :l'instant de début d'exécution pour toutes les tâches est le même(départ simultané, très souvent au temps t=0).

- Tâches périodiques asynchrones : les premières instances sont différentes (départ différé).

Le facteur d'utilisation de toutes les tâches périodiques $Ut(T)$ d'un système est donné comme suit :

$$U_t(\tau) = \sum_{i=1}^{n} U_i \tag{3.1}$$

Ou Ui est Le facteur d'utilisation de la tache périodique Ti qui est représenté par le rapport entre son pire temps d'exécution et sa période : $Ui = Ci/Ti$.

- Tâche apériodique : L'activation d'une tâche apériodique est un événement aléatoire, habituellement déclenché par une action extérieure au système (exemple : il peut être un clic sur bouton). Tâches apériodiques ont également des contraintes de temps qui leur sont associées. Le délai entre deux arrivées successives d'une tâche, et la fréquence ne sont pas connus. Le temps d'arrivée est généré à la suite par exemple d'une distribution de Poisson.

- Tâches sporadiques : Ces tâches sont des tâches en temps réel qui sont activées de façon irrégulière avec certain taux borné connu. Le taux borné est caractérisé par une période inter-arrivée minimum, qui est, un intervalle minimum de temps entre deux activations successives. En général les tâches sporadiques sont semblables aux tâches périodiques, à l'exception que la période est remplacée par, la séparation minimum entre les arrivées successives de la même tâche. Elle est caractérisée par un triplé $Ti(Ci, gi, Di)$,ou le temps d'exécution au pire cas d'une instance de la tâche égale à Ci et la période inter-arrivée minimum est gi ce paramètre implique que si une tâche sporadique se produit, l'instance suivante ne peut pas se produire avant au moins un temps égale a gi unités de temps, Di dénote l'échéance.
  Un système de tâche sporadique est constitué d'un ensemble fini de tâches sporadiques. Par exemple, dans un robot, une tâche qui est générée pour traiter un obstacle qui apparaît soudainement est une tâche sporadique [61] [55] [6] [15] [5].

3.4.4.3 Caractéristiques de l'ordonnancement

Les algorithmes d'ordonnancement possèdent différentes caractéristiques en fonction des caractéristiques du problème et les tâches à être ordonnancé.

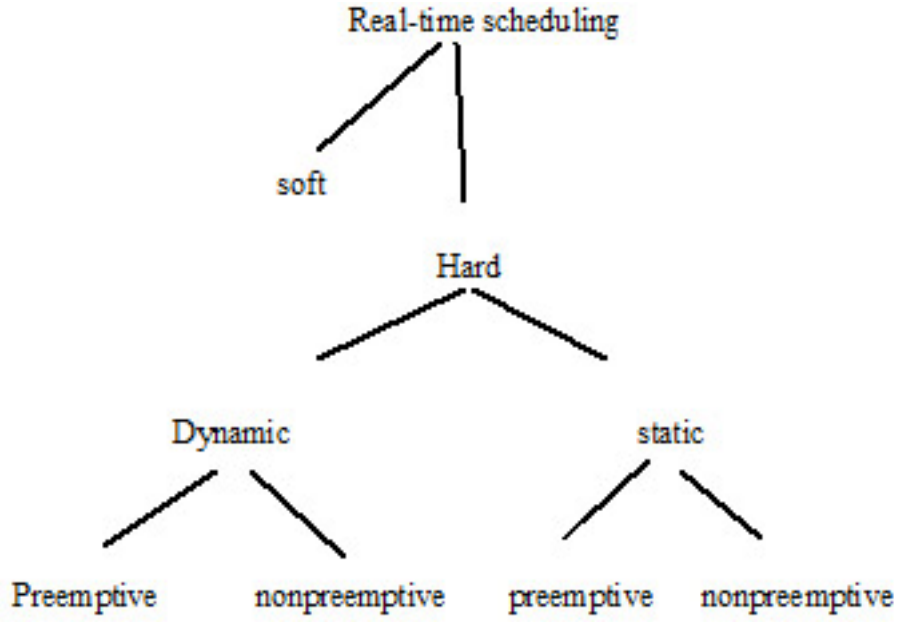
3.4.4.3.1 Ordonnancement optimal/ non optimal

Un algorithme d'ordonnancement optimal est celui qui peut ne pas respecter un deadline uniquement si aucun autre algorithme d'ordonnancement peut respecter cette deadline. Par conséquent, un algorithme d'ordonnancement réalisable est optimal s'il n'y a pas d'autre algorithme réalisable avec des conditions plus souples [55] [10].

3.4.4.3.2 Ordonnancement Préemptif / Non préemptif

En informatique, la préemption est la capacité d'un système d'exploitation multi-tâches à exécuter ou stopper une tâche planifiée en cours.

Si l'interruption d'une tâche en cours d'exécution sur les processeurs au profit d'une autre tâche plus prioritaire, alors on dit que l'ordonnancement est préemptif et par contre si l'exécution d'une tâche ne peut pas être interrompue une fois cette dernière entame son exécution, on dit que l'ordonnancement est non préemptif.



3.4.4.3.3 Ordonnancement hors-ligne / en-ligne

Hors-ligne : toutes les décisions concernant l'ordonnancement sont prises avant le démarrage du système et l'ordonnanceur a une connaissance complète de toutes les tâches avant l'exécution

En-ligne : les décisions sur la façon d'ordonnancer des tâches sont effectuées au cours de l'exécution du système(basées sur les priorités des tâches qui lui sont assignées soit dynamique ou statique).

3.4.4.3.4 Ordonnancement centralisé / distribué

Si toutes les tâches arrivent au niveau de chaque processeur de façon indépendante, et que chaque processeur prend ses propres décisions d'ordonnancement par un algorithme

local dans un système temps réel distribué alors l'ordonnancement est distribué.

Par contre , l'ordonnancement est centralisé, si toutes les tâches arrivent à un processeur central (répartiteur), qui prend toutes les décisions d'ordonnancement et affecte les tâches à d'autres processeurs disponibles[31].

3.4.4.3.5 Ordonnancement statique/dynamique

Les deux principales catégories d' ordonnancement sont l'ordonnancement des tâches statiques et dynamiques. Dans L'ordonnancement de tâches statique ou déterministe les contraintes de précédence et les relations entre les tâches sont bien connues et assignées avant l'activation des tâches et restent stables durant toute la période de l'exécution.

Tandis que,dans l'algorithme d'ordonnancement dynamique ou le non déterministe ces informations ne sont pas connues à l'avance. Et l'algorithme ne possède aucune connaissance sur les nouvelles tâches arrivant (uniquement l'ensemble des tâches actuellement actives).

L'ordonnancement de tâches statiques est réparti en deux classes : basé sur les heuristique et les algorithmes basé sur la recherche aléatoire guidée.

Les techniques de recherche aléatoires utilisent un choix aléatoires pour se guider à travers l'espace de problème. L'algorithme génétique appartient à la catégorie de Recherche aléatoire.

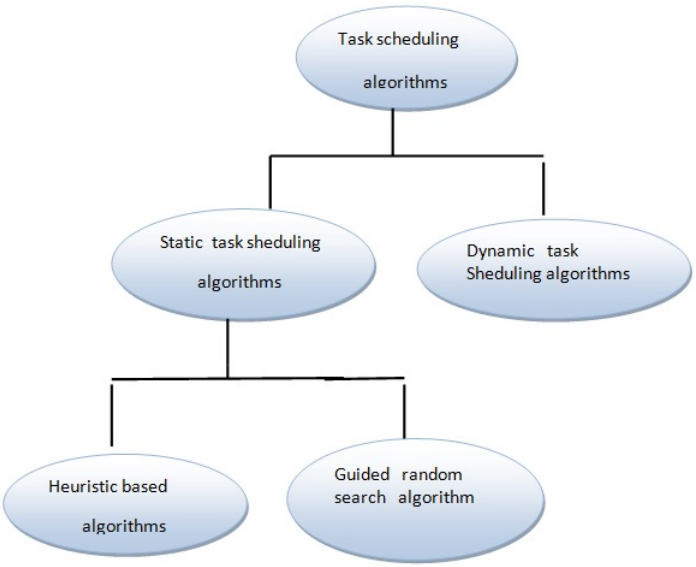


Figure 3.6 – Classification des algorithmes d'ordonnancement de tâches

3.4.4.4 Ordonnancement des tâches périodiques indépendantes

Si les tâches ne partagent pas des ressources entre eux et ne possèdent pas des contraintes de précédences alors se sont des tâches indépendantes.

3.4.4.4.1 Ordonnancement à priorités fixes (statiques)

Plusieurs algorithmes existent dans le cas où les tâches du système sont toutes périodiques, et indépendantes les unes des autres et avec priorités statiques.

Rate monotonic (RM) : RM est considéré comme un algorithme d'ordonnancement mono-processeur à priorités statiques, c.à.d sont déterminées avant l'exécution, dans RM toutes les tâches reçoivent leur priorité statique proportionnelle inversement à leur période. Dans ce sens les tâches avec les périodes plus courtes obtiennent les plus hautes priorités RM[31].

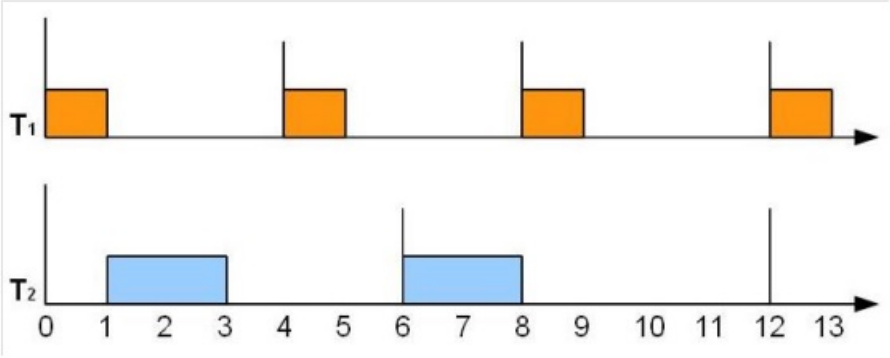


Figure 3.7 – Exemple de l'algorithme RM

L'algorithme taux monotone possède plusieurs conditions doivent être réalisées. Ces conditions sont formalisées par Liu et Layland.

- Le temps d'exécution des tâches périodiques est constant et connu ,et les tâches sont prêtes à être exécutées au début de chaque période (T).
- L'algorithme RM est optimal pour les algorithmes à priorités fixes et pour les systèmes à échéance sur requête ,et les tâches sont à départ simultané.
- Deadline(D) pour les tâches sont à la fin de chaque période c.à.d Di=Pi.
- Les tâches sont indépendantes, c.a.d, il n'y a aucune ation de précédence entre les tâches et donc elle ne se bloquent pas mutuellement.
- La contrainte d'ordonnancement est donnée ainsi :

$$\sum_{i=1}^{n} U_i = \sum_{i=1}^{n} (C_i/P_i) \leq 1 \tag{3.2}$$

Où :Pi et Ci sont respectivement la période de temps et le pire des cas d'exécution de la tâche Ti,n est le nombre de tâches, Ui est le facteur d'utilisation de la tâche Ti, $Ui = Ci/Pi$.

En général un ensemble de n tâches périodiques indépendantes, peut-être ordonnancé par la politique de taux monotone si et seulement si :l'utilisation totale du processeur n'est pas supérieur à $n(2^{1/n} - 1)$.

Deadline monotonic (DM) : Est un algorithme d'ordonnancement similaire à RM à la différence que date limite(deadline) détermine la priorité de la tâche , c.à.d. la tâche périodique avec le plus court deadline est attribué la plus haute priorité. DM est un

algorithme optimal et à départ simultané, utilisé lorsque Di<Ti.Ainsi La contrainte d'ordonnancement est similaire à celle de RM[31].

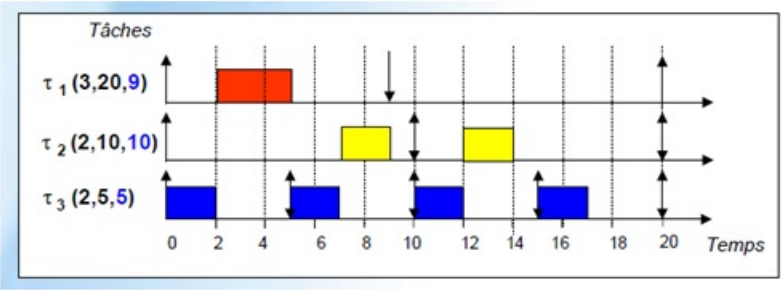


Figure 3.8 – Exemple de l'algorithme DM

3.4.4.4.2 Ordonnancement à priorités dynamiques mais fixes dans un emploi

Earliest Deadline First (EDF) : Introduit par Liu et Layland [] ,est un algorithme d'ordonnancement à priorités dynamiques pour les tâches périodiques(ou sporadiques). L'affectation des priorités aux tâches dans EDF réalisée en fonction de leurs échéances absolues, c.à.d. lorsque l'échéance absolue de la tâche est plus petite (proche) ,la priorité est plus forte. Cet algorithme est utilisé généralement avec préemption. L'algorithme EDF a été prouvé d'être optimal dans le sens de faisabilité entre tous les algorithmes d'ordonnancement [67].

Ainsi, une condition de l'ordonnancement dans EDF est :

$$\sum_{i=1}^{n} e_i/p_i = \sum_{i=1}^{n} U_i \leq 1 \tag{3.3}$$

Où : Ui est le facteur d'utilisation de la tâche Ti, tel que : $Ui = Ci/Pi$, n est le nombre de tâches.

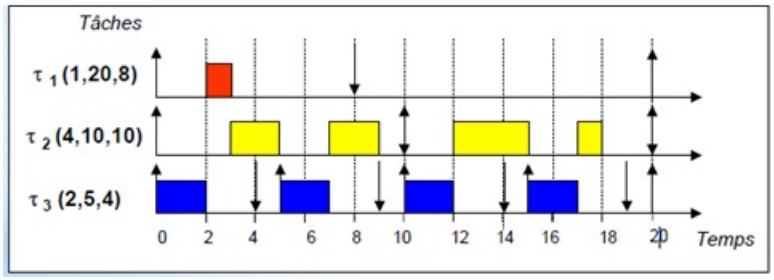


Figure 3.9 – Exemple de l'algorithme EDF

3.4.4.4.3 Ordonnancement à priorités dynamiques

Least Laxity First (LLF) : La laxités d'une tâche est l'intervalle de temps maximum pendant lequel la tâche peut ne pas avoir le processeur sans rater leur échéance. Dans cet algorithme, une forte priorité est affectée à la tâche ayant la plus petite laxité.

LLF est aussi, comme EDF, optimal dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que $Di \leq Ti$ a condition que le coût de changement de contexte soit ignoré.

3.4.4.5 Ordonnancement des tâches Sporadiques

- Une méthode simple consiste à considérer les tâches sporadiques comme les tâches périodiques avec une période égale à leur temps inter-arrivée minimum.

- Une autre approche consiste à définir une tâche périodique possédant la plus haute priorité et d'une certaine période fictive d'exécution choisie. Pendant le temps que cette tâche est ordonnancé pour s'exécuter sur le processeur, le processeur est disponible pour exécuter n'importe quelle tâches sporadiques qui peut être en attente de service. En dehors de ce temps, le processeur est chargé des tâches périodiques. Cette méthode est la plus simple pour le problème.

- Le différé Server est une autre approche, ce qui gaspille moins de bande passante. Ici, chaque fois que le processeur est ordonnancé pour exécuter des tâches sporadiques et ne trouve pas de telles tâches en attente de service, il commence à exécuter les tâches périodiques par ordre de priorité. Toutefois, si une tâche sporadique arrive, il préempte la tâche périodique et peut occuper un temps total jusqu'au temps alloué pour les tâches sporadiques [31].

3.4.4.6 Ordonnancement des tâches dépendantes

Dans les systèmes en temps réel plus réalistes les processus(tâches) interagissent afin de satisfaire aux exigences de l'ensemble du système , ces interactions prennent des formes variées (partager des ressources logicielles (accès mémoire) ou matérielles (capteurs)). L'ordonnancement de tâches dépendantes doit satisfaire les contraintes de synchronisation et les relations de précédence entre les tâches : [12]

3.4.4.6.1 problèmes de synchronisation

En effet, La principale difficulté, avec les ressources critiques ,est qu'une tâche de haute priorité peut être bloquée par une tâche de priorité inférieure (problème blocage) pour une durée indéterminée, ce phénomène est appelé l'inversion de priorité. Il existe aussi Un autre problème causé par la présence des ressources au sein des systèmes temps réel est l'inter-blocage.

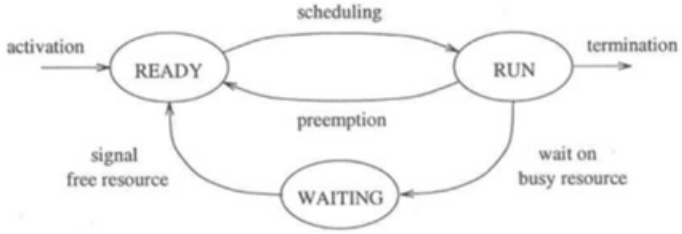


Figure 3.10 – État d'attente causé par une contrainte de ressource

3.4.4.6.2 Relations de précédence

En général pour capturer les dépendances entre les tâches, on utilise des relations de précédence,c'est-à-dire certaines tâches pour s'exécuter, elles doivent attendre la terminaison d'autres tâches. Les relations de précédence sont présentés par un graphe dirigé acyclique (DAG), les tâches sont décrites par des noeuds et les relations de précédence par des flèches.

Ja <*Jb* signifie que *Ja* est un prédécesseur de *Jb*.

Ja-> *Jb* signifie *Ja* est un prédécesseur immédiat de *Jb*

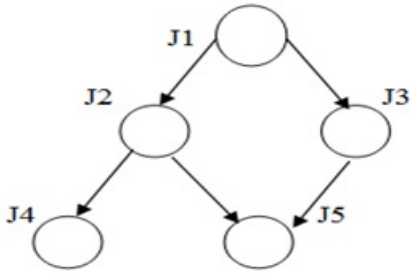


Figure 3.11 – Relations de précédence a travers cinq tâches

3.4.5 Ordonnancement temps réel multiprocesseur

En général les algorithmes d'ordonnancement des tâches temps réel pour les systèmes mono-processeur, peuvent être appliquées au systèmes multi-processeurs si l'on considère chaque noyau du multiprocesseur un noyau isolé, qui est un système mono-processeur. Dertouzos et al [22] ont prouvé que les algorithmes qui sont optimales pour des systèmes à processeur unique ne sont pas optimales pour l'augmentation du nombre de processeurs. Cependant, en mode multi-processeurs, la difficulté de vérifier l'exécution de différentes tâches sur plusieurs noyaux et de ne pas interférer les uns avec les autres et aussi de déterminer quelles tâches doivent être accordées à un certain noyau augmente la complexité considérablement par rapport à l'ordonnancement mono-processeur.

Néanmoins, l'affectation optimale des tâches à des processeurs est considérées comme un problème NP-difficile [55].

Par ailleurs, l'algorithme d'ordonnancement multiprocesseurs implique souvent des heuristiques pour simplifier la tâche afin de trouver un ordonnancement faisable.

L'ordonnancement multi-processeurs est composé de deux phases :

— Allocation des tâches « organisation spatiale » : la répartition judicieuse des tâches (déterminer processeur sur lequel la tâche va être exécutée)

— Ordonnancement des tâches « organisation temporelle » : déterminer l'ordre d'exécution des tâches sur chaque processeur.

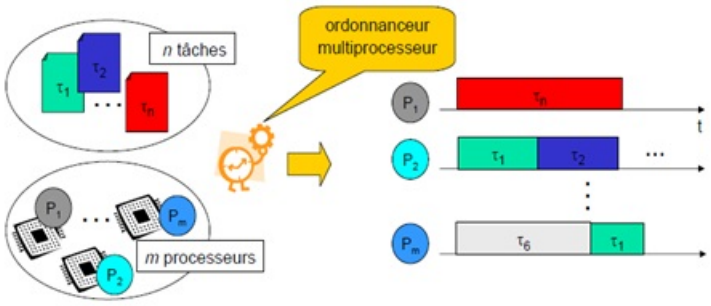


Figure 3.12 – Ordonnancement multi-processeur.

3.4.5.1 Ordonnancement par partitionnement

Chaque tâche est affectée à un processeur avant de débuter leur exécution et exécuter exclusivement sur ce processeur donc,les tâches d'une même partition sont affectées au même processeur et elles resteront sur le même processeur durant toute leur exécution. Cela se traduira par le fait qu'au lieu d'être un problème de multi-processeurs, il sera un ensemble de problèmes mono-processeur. Après l'allocation, l'algorithme d'ordonnancement mono-processeur peut être utilisé pour ordonnancé les tâches sur chaque processeur (par exemple, RM ou EDF), ce qui est un avantage. La migration des tâches d'un processeur à un autre n'est pas autorisée [55].

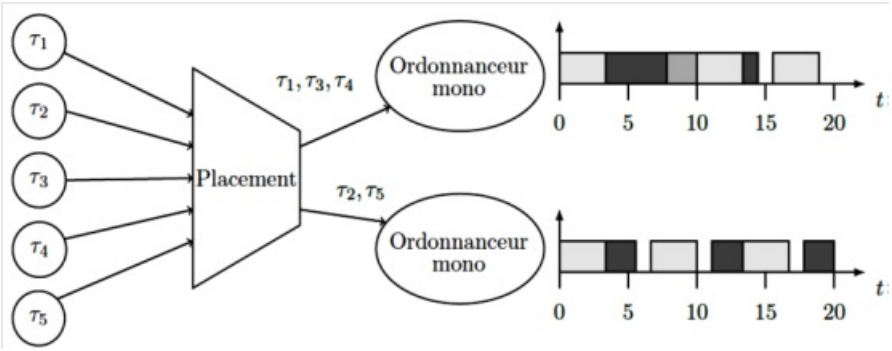


Figure 3.13 – Ordonnancement par partitionnement [17]

3.4.5.2 Ordonnancement global

Les tâches prêtes a être exécutées sont mise dans une file d'attente partagée entre tous les processeurs. Les tâches sont triées en fonction de la priorité. La tâche, avec actuellement la plus haute priorité, qui est la première dans la file d'attente, est sélectionnée par l'ordonnanceur et exécutée sur l'un des processeurs et peut être interrompue ou migré si nécessaire [13].

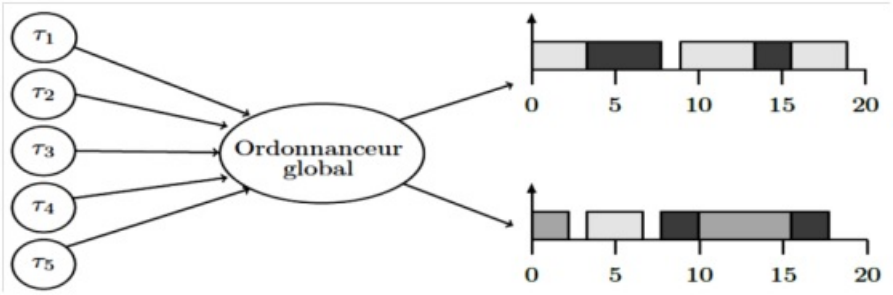


Figure 3.14 – Ordonnancement global [17]

3.5 Conclusion

Les algorithmes d'ordonnancement pour les systèmes embarqués temps réel ont été largement étudiés mais leur réalisation reste toujours un défi et plusieurs problèmes sont rencontrés et qui nécessitent des recherches plus approfondies. Cependant, la majorité des problèmes d'ordonnancement des tâches dans des systèmes temps réel avec plus de deux processeurs son NP-difficiles, alors le problème d'optimisation des performances est un facteur très important dans ces systèmes. Les approches pour traiter ces problèmes sont beaucoup, mais ces approches sont insuffisantes pour quelques problèmes. Donc, il faut utiliser des méthodes puissantes et précises par hybridation entre plusieurs approches puissantes l'algorithme génétique inspiré quantique et les systèmes multi-agents sont parmi ces approches. C'est ce qu'on va détailler dans le prochain chapitre.

CHAPITRE 4

LES ALGORITHMES GÉNÉTIQUES QUANTIQUES ET PROBLÈMES D'ORDONNANCEMENT CLASSIQUES ET TEMPS RÉEL

Ce chapitre présente les GAs classiques et un état de l'art sur les travaux concernant l'application de ces algorithmes aux problèmes d'ordonnancement classiques et temps réel. La fin de ce chapitre est consacrée au concept des GAs quantiques.

4.1 Introduction

Les phénomènes physiques et biologiques ont été la source d'inspiration de nombreux algorithmes à savoir les algorithmes évolutionnaires, les algorithmes du système immunitaire artificiel, les systèmes fourmis et les systèmes abeilles, etc...

Les algorithmes évolutionnaires qui sont inspirés du principe de l'évolution naturelle "The survival of the fitest" [19] a fait un grand pas dans les domaines de la résolution de problèmes complexes. Parmi les plus populaires approches, on trouve les GAs.

En fait, les GAs [30] sont des techniques d'optimisations globales utilisées dans de nombreuses applications de la vie réelle.

Dans ces GAs, ils ne s'agissent pas de trouver une solution analytique exacte, ou une bonne approximation numérique, mais de trouver des solutions satisfaisant au mieux à différents critères, souvent contradictoires. Ils manipulent un ensemble de plusieurs solutions simultanément et cherchent des solutions à des problèmes d'optimisation. La recherche de la solution se fait d'une manière itérative, elle commence par une population initiale et se déroule pendant certain nombre d'itérations pour terminer par le retour de la solution optimale si un certain critère d'arrêt est satisfait. L'aspect itératif de ces algorithmes permet d'explorer l'espace de recherche et améliorer la solution à chaque itération afin d'obtenir la solution optimale. Pour augmenter l'efficacité, certains concepts de la mécanique quantique ont été associés aux GAs classiques afin d'améliorer leurs performances, cette nouvelle technologie est appelée les algorithmes génétiques inspirés-quantiques (QGA).

4.2 Algorithme génétique classique

Les GAs ont été développés par Holland [33] à l'université du Michigan, USA. Holland a commencé par la simulation des processus d'adaptation des systèmes naturels. Ensuite, il est arrivé à les explorer dans l'optimisation et l'apprentissage automatique dans les années 1980. Ces GAs sont devenus les approches évolutionnaires les plus utilisées. Par la suite Goldberg [30] un de ces étudiants se servira de ce modèle dans ses recherches et publiera en 1989, un ouvrage de vulgarisation des GAs en développant ces modèles. Grâce à ces recherches deux idées principales ont été ajoutées aux GAs. [14].

— Le codage ou la représentation : Un individu dans la population est lié à un environnement par son code.

— La fitness : La liaison entre une solution et un problème est l'indice de qualité.

4.2.1 Le principe de base des GAs

Les GAs sont des métaheuristiques stochastiques itératives basé population qui simulent l'évolution naturelle des individus d'une population.

L'objectif des GAs est de guider des individus au seins d'une population pour obtenir une solution approchée à un problème d'optimisation donnée.

Ces GAs sont appliqués lorsqu'on peut pas déterminer une solution est impossible ou difficile de l'approcher en un temps raisonnable. Les GAs utilisent les principes de sélection naturelle, le croisement et la mutation en les appliquant à une population de solutions potentielles à un problème donné. La solution est approchée itérative ment.

Traditionnellement, les GAs sont associés à l'utilisation d'une représentation binaire. On peut trouver des GAS qui utilisent d'autres types de représentation (Scatter search). Un GA applique les opérateurs de croisement et de mutation afin de promouvoir la diversité. Les GAs utilisent une sélection probabiliste [3].

Pour mettre en oeuvre un GA, il est nécessaire de disposer :

— Une représentation génétique du problème, c'est-à-dire un codage de solutions utilisé sous la forme de chromosomes.

— Un mécanisme de génération de la population initiale. Ce mécanisme est indispensable pour construire une population d'individus non homogène.

— Une fonction qui permet d'évaluer l'adaptation d'un chromosome à son environnement, ce qui offre la possibilité de comparer des individus. Cette fonction est construite à partir du critère que l'on désire optimiser. L'application de cette fonction à un élément de la population donne sa fitness.

— Un mode de sélection des chromosomes à reproduire. Cette sélection est basée sur la reproduction et sur le codage génétique, qui stocke les informations décrivant l'individu sous forme de gènes.

— D'opérateurs de croisement et de mutation permettant de diversifier la population au cours des générations et d'explorer l'espace d'état.

— De paramètres qu'utilise l'algorithme : taille de la population, probabilité de croisement et de mutation, nombre total de générations.

Les niveaux d'organisation d'un GA :

Dans un algorithme génétique, les informations sont organisées en cinq niveaux, dans la population initiale on trouve un ensemble des individus, et un individu est constitué

d'un génotype doté d'un ou plusieurs chromosomes. Le chromosome est représenté par un ensemble de gènes dont chaque gène correspond à un paramètre qui dépend du problème d'optimisation à résoudre, il est codé sur un bit [14].

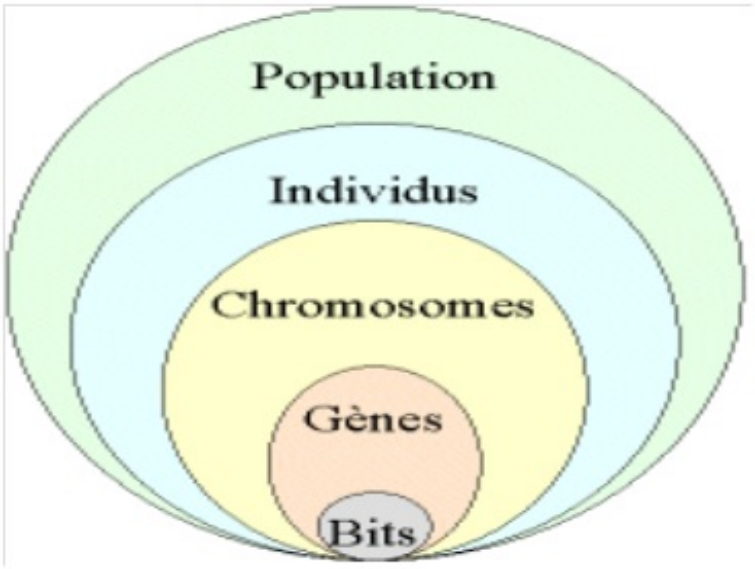


Figure 4.1 – Les niveaux d'organisation de l'information dans les GAs

4.2.1.1 Étapes de l'algorithme génétique

La première étape dans l'algorithme génétique consiste à générer une population initiale de Psize individus,et ensuite nous calculons leurs fitness et nous sélectionnons les individus par une méthode de sélection. Les opérateurs génétiques (croisement, mutation) seront appliqués sur Ces individus selon les probabilités Pcross, Pmut. Ces opérateurs de sélection et de recombinaison (croisement et mutation) permettent la génération de nouvelle population d'individus, qui ont de bonnes chances d'être plus forts que ceux de la génération précédente. Les individus résultant de l'étape de mutation seront insérés et remplacés par une méthode d'insertion dans la nouvelle population. Pour chaque nouvelle génération, la puissance des individus de la population agrandir. Et à la fin pour décider quand arrêter l'algorithme, un test d'arrêt sera effectué. La figure 4.2 présente un schéma de fonctionnement général de l'algorithme génétique.

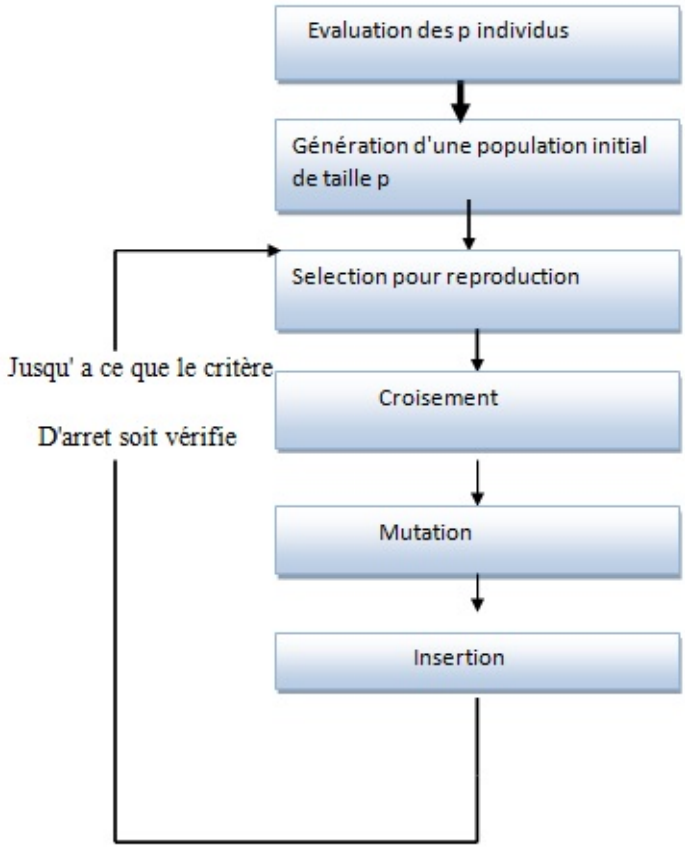


Figure 4.2 – Fonctionnement général d'un GA

4.2.1.1.1 Codage des chromosomes

Le codage des chromosomes ou la création d'une solution(chromosome) est la première étape dans la réalisation des GAs. Chaque point de l'espace de recherche est codé par un chromosome. Le choix du codage d'un chromosome est très important il peut rendre un algorithme génétique plus ou moins efficace. Le codage en vecteurs est la structure du codage la plus utilisée pour les GAs. Plusieurs types de codage sont utilisés pour coder les individus, on distingue :

— **Le codage binaire :** C'est le premier codage utilisé dans les GAs [35] retenu par John Holland. C'est le codage le plus utilisé, le codage de chaque chromosome de la population est fait par une chaîne de bits 0-1 de longueur fixe.

Un inconvénient à ce type de codage qui entraîne certains problèmes est que ce codage est généralement peu naturel par rapport à un problème donné.

— **Le codage réel :** Le codage de chaque gène se fait par une valeur réelle.Ce type de codage possède plusieurs avantages. Parmi ces avantages est que la marge de valeurs possibles des paramètres fournis par le codage réel est plus grande[35].

— **Le codage de gray :** Ce codage est utilisé pour éviter l'inconvénient que deux éléments voisins dans l'espace de recherche en codage binaire ne codent pas toujours deux solutions proches. Les nombres binaires dans le codage de Gray sont redéfinis de sorte que la distance de Hamming entre deux éléments consécutifs égale à 1, c'est-à-dire, un seul bit diffère [50] [42] [2].

4.2.1.1.2 Initialisation de la population

La population initiale doit être choisie de manière à éviter la convergence vers l'optimum global, donc la répartition de la population sur tout le domaine de recherche devient essentielle. Une méthode simple consiste à initialiser aléatoirement les valeurs des individus initiaux. Il faut bien noter que le choix de la population initiale peut dégrade l'algorithme, une grande taille de la population,implique l'augmentation du temps de réponse et espace de mémoire important. Tandis qu'av-ec une petite taille la solution obtenue n'est pas satisfaisante.

4.2.1.1.3 Evaluation des individus

Les individus d'une population dans un algorithme génétique, sont évalués afin de calculer leur fitness, en utilisant une fonction d'adaptation. Le choix des meille-urs individus se fait grâce au résultat fourni par la fonction d'adaptation. Donc cette fonction consiste à éliminer les individus moins performants et conserver les individus les plus performants [21].

4.2.1.1.4 Les opérateurs génétiques

Afin de garantir le principe de la reproduction. Il faut appliquer des opérateurs génétiques qui sont : la sélection, le croisement et la mutation.

1. **Sélection :**

   Le but de la sélection dans un algorithme génétique est de choisir les individus qui doivent se reproduire après avoir réalisé l'évaluation des individus. Les individus sont choisis sur la base de leurs fonctions d'adaptation qui vont servir de parents dans l'étape suivante. Il existe deux parties dans la sélection : la première consiste à déterminer qui seront les individus, qui vont se reproduire entre eux, et former des enfants. Et la deuxième partie consiste à décider quels sont ceux qui vont survivre [50].

   de nombreux opérateurs de sélection existent les plus utilisés : la sélection à la roulette, par tournoi, par élitisme et par rang.

   — **Sélection par roulette biaisée** : C'est la méthode la plus connue et la plus utilisée. Peut-être représentée par une roue découpée en secteurs, et chaque chromosome est représenté par un secteur. L'angle de ce secteur est proportionnel au fitness de l'individu qu'il représente. Les meilleurs individus ont plus de chances d'être obtenus une fois la roue est lancée. Donc cette méthode, permet à chaque lancement de la roulette de sélectionner les individus les plus adaptés au problème.

   $$P_s(x_i) = \frac{fitness(x_i)}{\Sigma_{(j=1)}^{N} fitness(x_j)}$$

   — **Sélection par élitisme** : Introduit par Kenneth De Jong en 1975, dans cette méthode les individus sélectionnés pour se reproduire sont choisis selon leurs fitness, c'est-à-dire ,sélectionner les premiers meilleurs individus. La convergence des solutions est plus rapide. Les meilleures solutions au cours d'évolution ne sont pas perdues.

   — **Sélection par tournoi** : Dans Cette méthode un tirage avec remise de paires d'individus est effectué, et nous le faisons combattre. Ensuite on sélectionne le gagnant parmi ces paires, l'individu qui a la meilleure fitness, pour se reproduire avec une probabilité $P \in [0.5, 1]$. Le processus est répété n fois afin d'obtenir les n individus qui serviront de parents.

   — **Sélection par rang** : Cette méthode consiste à attribuer à chaque individu un rang proportionnel à son fitness, d'une manière à attribuer l'individu le

plus mauvais le rang 1 et le meilleur aura le rang N tel que N est la taille de la population,ensuite, une roulette est implémentée en fonction de ces rangs. Ainsi la sélection d'un individu x se fait par une probabilité : [50] [34]

$$P_s(x_i)=\frac{Rang(x_i)}{\Sigma_{(j=1)}^{N} Rang(x_j)}$$

2. **Croisement** :

   De nouvelles chaînes (enfants), sont créées par l'opérateur par l'échange d'informations entre deux chaînes (parents). Pour créer une nouvelle génération, une sélection de parents est nécessaire afin qu'ils puissent se reproduire. Généralement et si les enfants héritent les meilleures caractéristiques ils peuvent être meilleurs que les parents, ce qui permet une meilleure exploitation de « l'espace de recherche ». Le processus de croisement consiste à associer deux individus en une paire avec une probabilité Pc, plus cette probabilité est élevée, plus la taille de la nouvelle population augmente, et cela dû au nombre grandissant des individus qui apparaissent. Habituellement le croisement peut être classé en deux classes ; celui appelé « en un point » et celui appelé « à des points multiples ».

   — **Croisement en un point** : Une sélection aléatoire d'un point pour chaque paire d'individus est effectuée, après on effectue la permutation des séquences entre ses individus, la figure suivante illustre le principe de fonctionnement de cette classe de croisement :

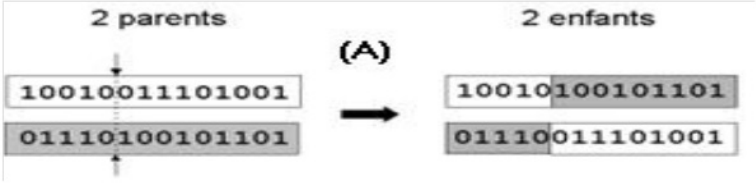


Figure 4.3 – Le mécanisme de croisement à un point

— **Croisement en multi points** : Contrairement au croisement en un point, plusieurs points sont sélectionnés aléatoirement et l'échange entre les parents s'effectue selon les différentes parties des séquences cernées par ces points.

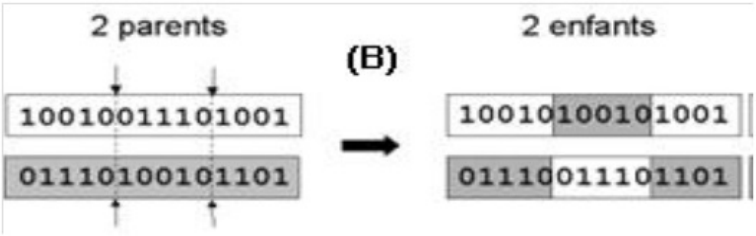


Figure 4.4 – Le mécanisme de croisement à multi points

— **Croisement uniforme** : Un autre type de croisement peut être recensé, nommé le croisement uniforme, ce mécanisme consiste à définir un masque de bits 0,1 dont la longueur est égale au nombre d'individus parents sur lesquels il sera appliqué. Si le bit du masque égale à «1 » dans la position « i » alors l'échange entre les parents dans la même position est possible, « 0 » ne désigne pas d'échange [14].

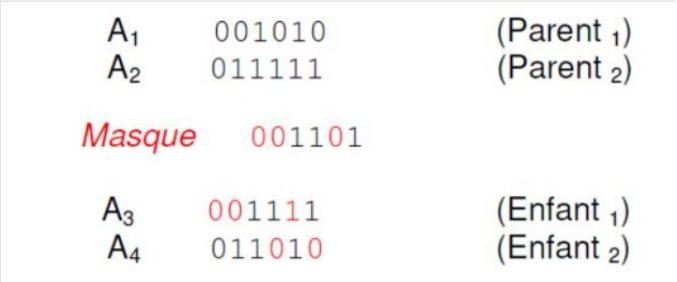


Figure 4.5 – Exemple d'un croisement uniforme

3. **Mutation** :

   L'opérateur de mutation est appliqué sur un seule individu. Le principe de la mutation consiste à changer la valeur allylique d'un gène pris au hasard avec une probabilité Pm très faible(généralement comprise entre 0.01 et 0.001). de 1 à 0 et vice versa.Cet opérateur important, permet d'introduire de la diversité dans GA et peut aider à s'échapper des optimum locaux c'est-à-dire permet d'explorer l'espace de recherche[21].

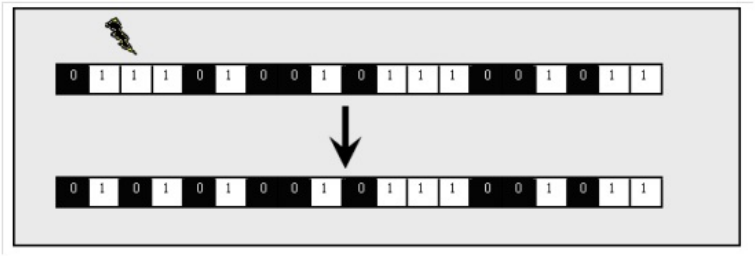


Figure 4.6 – Opérateur de mutation

4. **Insertion et remplacement** :

   Les nouveaux individus obtenus après l'application des opérateurs de croisement et de mutation, sont réintroduits dans la population en fonction de la valeur de fitness associé aux enfants. Donc la nouvelle population est constituée d'individus sélectionnés pour continuer à participer son amélioration. Une méthode d'insertion est utilisée.

   On trouve essentiellement 2 méthodes de remplacement différentes :

   — **Remplacement stationnaire** Un remplacement automatique des parents par les enfants sans prendre en considération leurs performances respectives.

— **Remplacement élitiste** : Le facteur performance est pris en compte dans cette stratégie qui consiste a garder au moins l'individu le plus performant lors du passage d'une génération à la suivante.

4.2.1.1.5 Critères d'arrêt

Plusieurs critères d'arrêt de l'algorithme peuvent être considérés :

— Selon le problème à résoudre , le nombre de générations que l'on souhaite exécuter peut être fixé à priori. C'est la démarche qu'on a suivie lorsque on a cherché à trouver une solution dans un temps déterminé.

— État de stagnation de la population.

En général les étapes de l'algorithme sont répétées jusqu'à ce qu'un critère d'arrêt soit vérifié [46] [68].

4.2.2 Avantages et inconvénients

Les GAs possèdent plusieurs avantages qui permettent de les utiliser dans des domaines multiples mais ils ont aussi des inconvénients qui imposent des restrictions dans leur utilisation, nous citons quelques-uns.

1. **Avantages [58]**

   — En général les GAs accélèrent le parcours d'un vaste ensemble de solutions. Et les mauvaises solutions ne sont pas prises en compte, elles sont éliminées afin de ne pas affecter la solution optimale.

   — Grande efficacité pour résoudre tous les problèmes d'optimisation qui peuvent être décrit avec le codage de chromosomes.

   — Les GAs traitent des problèmes très complexes d'une manière très efficace et permettant facilement de trouver les bonnes solutions, cette caractéristique a beaucoup aidé dans les domaines avec un grand nombre de paramètres et dont le temps de réponse doit être réduit (obtenir de bonnes solutions en quelques itérations seulement).

2. **Inconvénients [58]**

   — Trouver les bonnes solutions pour des problèmes avec un grand nombre de paramètres n'est pas toujours assuré par GA car il est très lent.

— GA peut converger vers les optimums locaux et ne pas trouver un optimum global.

— Il s'agit d'une technique d'intelligence artificielle, donc l'algorithme génétique ne peut pas assurer toujours les mêmes résultats en terme de temps. Cette propriété limite leur utilisation dans les applications temps réel.

4.2.3 Un état de l'art

Plusieurs travaux ont été proposés dans la littérature, pour traiter le problème de l'ordonnancement des tâches temps réel certains concernent l'utilisation d'une approche génétique classique.

La différence principale entre eux étant dans la manière de représenter les solutions (chromosomes).

Et certains concernent à hybrider les GAs à d'autres approches tels que les systèmes multi-agents et l'informatique quantique.

Dans cette partie, nous allons citer les travaux les plus importants sur l'application des GAs classiques et QGA aux problèmes d'ordonnancement multiprocesseurs temps réels :

Dans [45], les auteurs ont proposé un GA pour la résolution du problème d'ordonnancement non préemptif des tâches temps réels souples et indépendantes dans un système multiprocesseur. L'architecture des processeurs est centralisée et les priorités sont affectées de façon statique. Ou un processeur central possède une file d'attente pour toutes les tâches arrivées et par la suite ses tâches sont distribuées vers les autres processeurs pour s'exécuter.

Le chromosome dans cet algorithme est sous forme d'une liste de taille fixe égale au nombre de tâches. Ou le gène ici est un couple décimal de deux attributs <Tj,Pi>, Tj est une tâche et Pi est le processeur associé à cette tâche.

Une heuristique est utilisée pour générer la population initiale. Et pour les opérateurs génétiques ils ont utilisé la méthode de sélection par échantillonnage stochastique sans remplacement, le croisement en un point choisi de manière aléatoire. L'opération de mutation effectué une commutation entre deux tâches sélectionnées de façon aléatoire,et un changement aléatoire est appliqué à chacun des processeurs associés à ces tâches.

Dans [16], les auteurs ont proposé un GA basé sur une fonction d'énergie faisable

pour résoudre un problème d'ordonnancement multiprocesseur préemptif avec les contraintes de ressources(non préemptives) et temporelles dans un système temps réel dynamique.

Le terme d'énergie dans cet algorithme peut être défini comme la suite : avec, Vijk = 1 indique que la tâche i est exécutée sur le processeur j à l'instant k, Vijk = 0, autrement. où le variable "i" représente toutes les tâches à être ordonnancées, "j" représente tous les processeurs à être exploités. Enfin, le temps "k" représente un moment précis(deadline de la tâche).

Le chromosome est codé par (Pijk ; i = 1.... N ; j = 1, ... M ; k = 1, ..., T), où Pijk représente la probabilité de Vijk = 1. La dimension du chromosome est égale à N * M * T. La population initiale et générée aléatoirement en raison de la préemption, l'arrivée de nouvelles tâches, peut imposer que l'ordonnancement consume de temps donc une approche d'ordonnancement dynamique est développée pour le traitement réactif de nouvelles tâches implique que les nouvelles tâches arrivées seraient insérées dans les lignes en un rien de temps sans ordonnancer le reste de tâches pour réaliser ceci on va prévoyez le processeur qui sera inoccupé pour un moment plus grand que sa deadline et réserver des chambres pour les nouvelles tâches arrivées. La méthode de sélection par roulette, est utilisée.

Le point de croisement est aléatoirement choisi et deux individus sont appareillés aléatoirement pour échanger les segments correspondants pour former de nouvelles solutions avec une probabilité PC. L'opérateur de mutation est appliqué arbitrairement pour changer un gène d'un chromosome choisi avec un PM le critère d'arrêt dans cet algorithme est lorsque la valeur de la fonction d'énergie atteint zéro.

Dans [9], les auteurs ont proposé un QGA reposant sur une représentation quantique pour le codage de l'espace de recherche. Pour résoudre le problème d'ordonnancement non préemptif des tâches périodiques et apériodiques et des contraintes souples et dures et avec des contraintes de dépendance dans un système temps réel avec une architecture multiprocesseurs. QGA est utilisé pour optimiser (minimiser) le temps de réponse moyen de tâches et le nombre de tâches manquantes leurs échéances sous l'idée de l'équilibre entre le taux d'utilisation des processeurs.

Les tâches sont représentées par un graphe de tâches avec des relations de dépendance. L'ordonnancement des tâches utilise deux politiques : l'ordonnancement statique où les priorités des tâches sont pré-calculées, puis fixées pour toutes les périodes en

fonction de leurs échéances relatives (DM : Date limite monotone) ou leurs périodes (RM : Taux monotone) et l'ordonnancement dynamique où les priorités sont assignées au hasard à des tâches pour chaque période. Les messages entre les tâches sont également modélisées avec des graphes, Un message est activé uniquement lorsque la tâche qui produit ce message termine son exécution. Un chromosome est codé comme une matrice binaire. Deux algorithmes d'ordonnancement sont utilisés : RM et DM. Et Pour les tâches apériodiques, un algorithme de serveur des tâches apériodiques est appliqué.

Pour les opérateurs génétiques, ils ont utilisé deux techniques de croisement : en un point et deux points et la mutation est utilisée avec une certaine probabilité PMUT.

Dans le travail [7], un algorithme génétique hybride au sein d'un système multi-agents organisé selon une architecture hiérarchique a été proposé. Pour le problème d'ordonnancement des ateliers Flow-Shop hybride.

Dans le premier axe, ils ont implémenté au niveau de l'agent Superviseur une approche génétique hybride, l'ordonnancement initial est calculé par l'agent Superviseur dès le lancement des centres de travail la première fois. L'algorithme génétique est hybridé avec une méthode de recherche locale afin de pallier aux inconvénients pour l'aider à intensifier sa recherche dans les zones les plus prometteuses (autour d'une bonne solution). L'idée principale est d'ajouter une recherche locale en plus de la mutation. Cette recherche sera appliquée à tout nouvel individu obtenu au cours de la recherche.

Et dans Le second axe une nouvelle approche consiste à ne pas attribuer la fonction d'ordonnancement à un agent unique comme dans le premier axe (Agent superviseur), mais émerge d'un processus de négociation inter agents, ceci permet de diminuer la charge de l'agent superviseur. Cette approche suppose que des données sont connues au moment où l'ordonnancement initial est calculé par un premier algorithme, (approche génétique hybride) dit prédictif. Il devient alors nécessaire de déterminer une seconde méthode, dit dynamique (réactif). Cette méthode est un processus de collaboration et négociation des agents interagissant pour résoudre la situation troublée.

Alors, l'optimisation sera comme suit : A chaque itération, deux solutions P1 et P2 sont choisies aléatoirement dans la population, ces deux solutions-parents subissent un croisement. La solution (enfant) obtenue est évaluée d'une manière optimale. Par la suite, l'inclusion d'une procédure de recherche locale comme agent d'intensification, dans le but d'améliorer significativement les résultats, et on applique une mutation. La recherche locale est appliquée à tout nouvel enfant C avec une probabilité fixée Pm. L'itération

courante de l'algorithme se termine en remplaçant la solution courante, l'individu le plus mauvais (i.e., le plus coûteux) de la population, par l'enfant C à condition qu'il vérifie la condition de diversité.

4.3 Algorithmes Génétiques Quantiques

4.3.1 Introduction à L'informatique Quantique

4.3.1.1 Définition

L'informatique quantique est une nouvelle branche de l'informatique qui connaît un intérêt grandissant. Et fait appel à plusieurs spécialités : physique, génie, chimie, informatique et mathématique. En intégrant ces connaissances. On peut réaliser un ordinateur quantique et l'utilisation d'un tel ordinateur pour effectuer certains calculs beaucoup plus rapidement qu'avec un ordinateur fonctionnant de manière classique est possible. Cette accélération est rendue possible en tirant profit des phénomènes quantiques tels que les superpositions d'états, l'enchevêtrement et l'interférence.

4.3.1.2 Application à l'informatique

Par analogie l'informatique quantique est fondée sur les lois de la mécanique quantique. En informatique classique l'unité fondamentale est le bit mais en informatique quantique, les bits sont remplacés par une autre structure élémentaire de stockage de l'information : qubit, Un qubit correspond soit à une particule à deux états différents 0 et 1, soit la superposition de deux états différents. Une distribution de phase, qui prend la valeur 0 pour un angle de 90, un 1 pour un angle de 0 et pour un angle entre les deux la superposition d'états dans les proportions du sin et du cos de la phase. L'interrogation d'un qubit qui n'a pas une phase p comprise entre 0 et 90 conduit à une réponse par 0 avec une probabilité de sinp et à une réponse par 1 avec une probabilité de cosp. Les deux états sont représentés par convention |0> et |1>. Ce qui implique le parallélisme. Formellement, un qubit est souvent représenté par un vecteur dans l'espace de Hilbert de dimension 2, appeler la notation de Dirac : [43].

$$|\Psi >= \alpha|0 > +\beta|1 >$$

où

$$|\alpha|^2 + |\beta|^2 = 1$$

Et peut être aussi représenté sous une forme matricielle par :

$$|0> \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|1> \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|\phi> \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

En informatique classique, un bit peut prend toujours une valeur fixe dans 0,1, par contre un qubit n'a pas une valeur fixe tant qu'on ne l'a pas mesuré. Un qubit peut être dans une superposition des deux états en même temps.

Bit classique	**Bit quantique (qubit)**
Un bit a toujours une valeur définie	Pas de valeur définie sans observation
Un bit vaut seulement 0 ou 1	Un qubit peut être dans une superposition de 0 et 1 simultanément
Un bit peut être copié sans être affecté	Un qubit dans un état inconnu ne peut être copié
Un bit peut être lu sans affecter sa valeur	La lecture d'un qubit qui est initialement dans une superposition changera sa valeur
Lire un bit n'affecte pas un autre	Si un qubit est enchevêtré avec un autre, la lecture de l'un affectera le second.

Tableau 4.I – Comparatif Entre les Bits Classiques Et les Bits Quantiques [42]

4.3.1.3 Le registre quantique

Dans la pratique, ce n'est pas possible de travailler uniquement au niveau qubits, il faut déterminer des registres composés d'un ensemble de n qubits. Un tel registre de taille n est une superposition arbitraire de 2n valeurs. L'état de ce registre de taille n s'écrit comme suit :

$$\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{array}\right]$$

Figure 4.7 – Exemple d'un registre quantique

Dans un registre, on trouve la superposition de toutes les permutations possibles,

où les amplitudes de probabilité α_i et β_i sont des nombres complexes satisfaisant la condition :

$$|\alpha_i|^2| + |\beta_i|^2 = 1$$

Donc on a le vecteur $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ représente un qubit soit dans l'état |0> avec probabilité $\alpha_i|^2$, soit l'état |1> avec probabilité $|\beta_i|^2$ [44].

Pour la manipulation du contenu de ces registres, des portes quantiques sont utilisées.

4.3.1.4 Les portes quantique

Les portes quantiques dans les ordinateurs quantiques sont équivalentes des portes logiques en informatique classique. Une porte quantique est une opération unitaire de l'espace de Hilbert de dimension 2^N, qui peut s'écrire comme des combinaisons de transformations locales agissant sur un ou plusieurs qubits [44].

En réalité toute opération élémentaire dans un algorithme quantique doit être réalisée par des circuits quantiques. Il existe plusieurs portes quantiques, nous citerons quelques exemples :

— **Porte de Hadamard (lame demi-onde à 22,5)** : Permet d'effectuer une rotation de l'état d'un qubit, pour créer des superpositions d'états. Elle permet de créer 2^N valeurs à partir de n qubits [38].

Figure 4.8 – La porte quantique de Hadamard

— **Porte Non Contrôlée (CNOT)** : Elle s'applique à deux qubits : consiste à inverser le deuxième qubit si et seulement si le premier est dans l'état |1>.

Entrée	**Sortie**
00	00
01	01
10	11
11	10

Tableau 4.II – Table de vérité de la porte CNOT

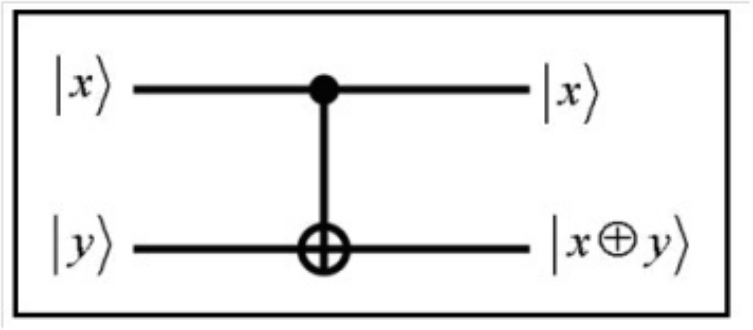


Figure 4.9 – La porte quantique CNOT

— **Porte de Toffoli** : Elle est appliquée sur trois qubits. Consiste à inverser le troisième qubit si et seulement si les deux premiers qubits sont à|1>.

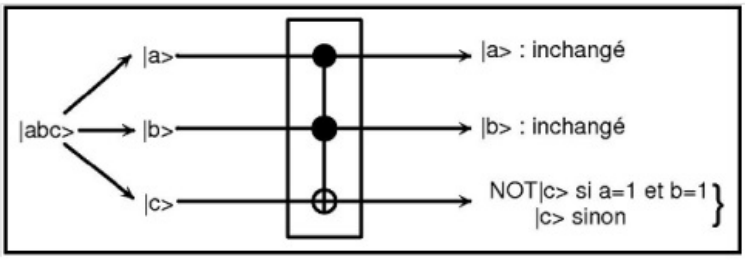


Figure 4.10 – La porte quantique de Toffoli

4.3.1.5 Circuits quantiques

Un circuit quantique est une combinaison de deux ou plusieurs portes quantiques dont le but est de remplir un traitement plus compliqué sur un système quantique. [38].

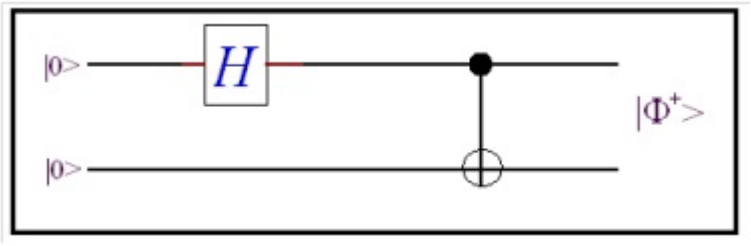


Figure 4.11 – Exemple d'un circuit quantique

4.3.2 Algorithmes génétiques quantiques

Un QGA est un GA dont les individus manipulés sont des chromosomes quantiques, c'est-à-dire une représentation basée sur le concept du qubit, et avec d'autres opérations quantiques. La figure 4.12 illustre le fonctionnement d'un QGA :

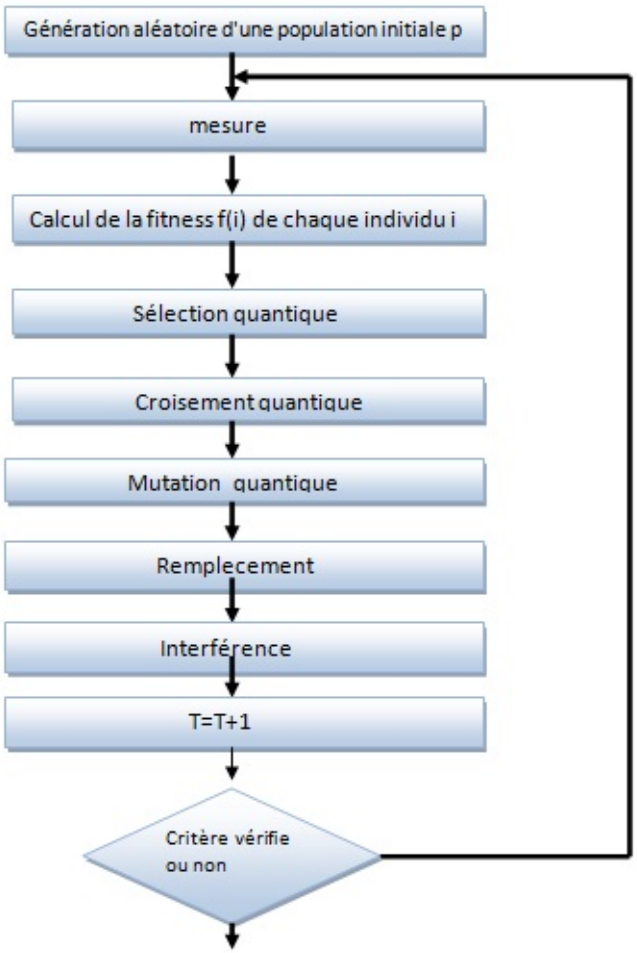


Figure 4.12 – Principe de fonctionnement d'un QGA

4.3.2.1 Codage des chromosomes quantiques

Dans l'informatique quantique, les individus sont basés sur une codification quantique (qubit), donc un chromosome est tout simplement représenté sous forme d'une chaîne de m qubits formant un registre quantique.

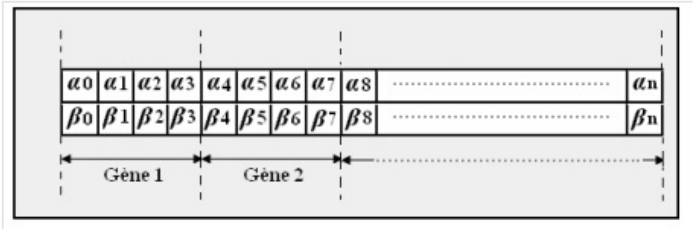


Figure 4.13 – Représentation du chromosome quantique

4.3.2.2 Initialisation de la population

Une façon simple pour l'initialisation de la population consiste à initialiser toutes les amplitudes des qubits par la valeur $2^{-1/2}$, c'est-à-dire un chromosome quantique donne les mêmes probabilités pour tous les états de superposition. Notant que : Le qubit se trouve dans l'état 0 avec une probabilité α^2 et la probabilité que le qubit se trouve dans l'état 1 est β^2.

4.3.2.3 Évaluation des individus

La phase d'évaluation des individus consiste à attribuer pour chaque individu, après l'application de l'observation, une valeur d'adaptation, et par la suite et grâce à une fonction d'adaptation La quantification de la qualité de chaque chromosome quantique se fait pour effectuer une reproduction.

4.3.2.4 Opérations génétiques quantiques

1. **fonction mesure :**

   L'objectif de cette fonction est d'obtenir un chromosome classique à partir d'un autre quantique afin de l'évaluer. Donc elle consiste à transformer chaque qubit à un bit classique avec une seule valeur : 0 ou 1.

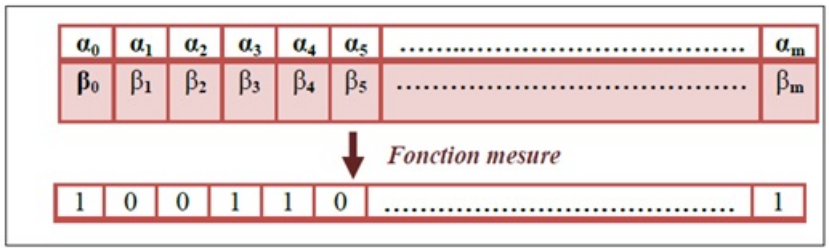


Figure 4.14 – Mesure d'un chromosome quantique

L'algorithme suivant illustre bien l'implémentation de cette fonction :

Algorithm 2: Fonction mesure ()

Debut ;
r = tirer r dans $[0, 1]$;
Si $r > \alpha^2$ **alors**
  retourner 1
sinon
  retourner 0
Fin

2. **La sélection quantique :**

   Après la réalisation de l'évaluation de la génération, la sélection est réalisée selon les valeurs d'adaptation. Les individus sont sélectionnés pour se reproduire selon les mêmes techniques de sélection que ceux d'un GA ordinaire.

3. **Le croisement quantique :**

   L'opérateur du croisement quantique peut être appliqué d'une manière similaire à celle d'un GA classique à l'exception qu'elle opère sur les qubits d'un chromosome quantique. Deux nouveaux chromosomes sont créés par l'échange des fragments situés après les points de coupures. Dans la Pratique, on peut appliquer le croisement en un point, deux points ou encore en plusieurs points

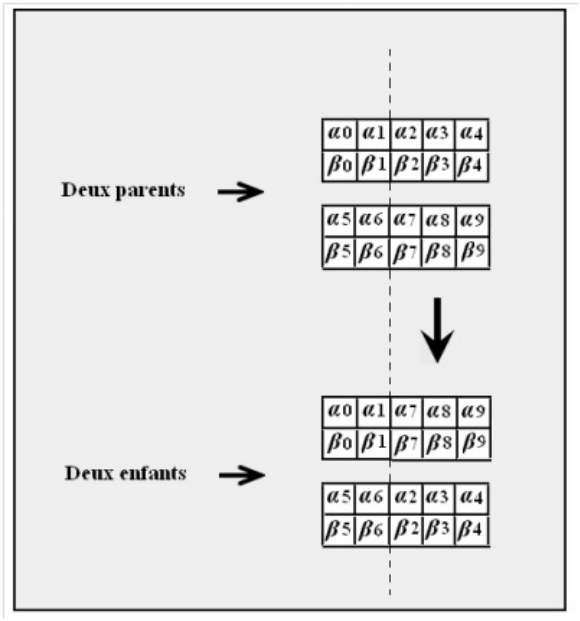


Figure 4.15 – Croisement quantique en un point

4. **La mutation quantique :**

   Cet opérateur est appliqué selon un taux de mutation, consiste à choisir un qubits de façon aléatoire et changer la valeur de sa positions. La réalisation de Ce changement se fait par la permutation des amplitudes (α_i, β_i) du qubit choisi. C'est-à-dire inverser les probabilités d'être a l'état 1 ou 0.

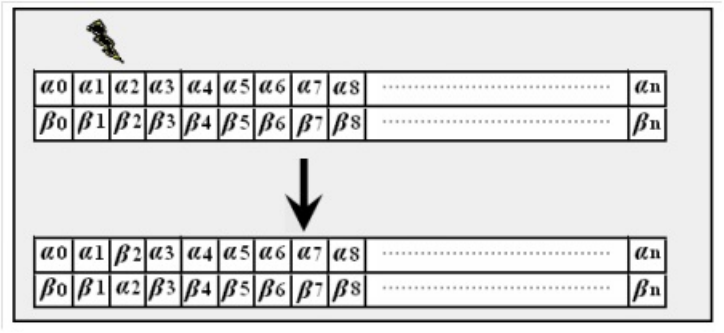


Figure 4.16 – Mutation quantique

5. **L'interférence :**

   L'opération d'interférence en informatique quantique consiste à modifier les amplitudes des individus pour le but d'améliorer l'exploitation de l'espace de recherche. Son rôle principal est de déplacer l'état de chaque qubit d'une manière à être proche de la valeur de la meilleure solution trouvée ,afin d'intensifier la recherche autour de la meilleure solution. Elle peut être réalisée à l'aide des portes quantiques choisies selon le problème donné, qui permet une rotation avec un angle θ. La valeur de θ est déterminé en fonction des amplitudes (a_i, b_i) et aussi la valeur du bit correspondant à la solution référence enregistré. Il faut bien noter que le choix de La valeur de l'angle de rotation peut guider l'algorithme vers la convergence prématurée. Elle est souvent fixée empiriquement [44].

4.4 Conclusion

Dans ce chapitre, nous avons présenté premièrement les notions de base de GA classique et ensuite nous avons évoqué son hybridation avec l'informatique quantique ce qui a donné naissance aux algorithmes génétiques inspirés quantiques QGA. Un QGA est un algorithme inspiré de la physique et de la génétique. Grâce au codage quantique dans cet algorithme et la superposition des états dans le traitement des chromosomes, la réduction de la taille de la population tout en assurant la diversité et aussi permet un parallélisme qui devrait réduire considérablement le temps d'exécution. Ainsi cette hybridation a donné des meilleurs résultats dans de nombreux problèmes d'optimisation que les GAs classiques.

CHAPITRE 5

L'APPROCHE PROPOSÉE MAS-QGA POUR L'ORDONNANCEMENT DE TÂCHES DANS LES SYSTÈMES EMBARQUÉS TEMPS RÉEL

Ce chapitre présente l'approche MAS-QGA, pour l'ordonnancement de tâches dans les systèmes embarqués temps réel, proposée suivie par la partie résultats numériques.

5.1 Introduction

Dans ces dernières années, les méta-heuristiques sont considérés comme étant une stratégie efficace pour la recherche des solutions approchées des problèmes d'optimisation.

L'hybridation entre ces méta-heuristiques avec d'autres approches a beaucoup aider dans l'accélération de la recherche des solutions approchées.

L'objectif principal de cette hybridation est d'intensifier la recherche des solutions et de s'échapper aux optimums locaux puisqu'elle permet toujours de rester autour des zones de solutions les plus prometteuses.

Dans ce chapitre, nous présentons l'approche proposée ainsi que les techniques de résolution adoptées.

Premièrement, pour résoudre les problèmes rencontrés lors de l'application des GAs à savoir la grande taille de population et le grand nombre d'itération nécessaire pour la convergence, on va faire une hybridation entre ces GAS et le quantique afin de réduire la taille de la population. Nous appelons cette approche algorithme génétique quantique (en anglais Quantum Genetic Algorithm) (QGA). Un autre inconvénient des GAs est le temps de calcul. En effet, bien que les GAs sont efficace pour trouver l'optimum global, leurs convergences nécessitent un nombre considérable d'itération, ce qui les rendent très coûteux en matière de temps de calcul.

Les GAs sont naturellement parellèlisables et sont facilement implémentés sur des architectures distribuées.

Dans le but de proposer une approche distribuée pour notre problème, nous exploitons les avantages des systèmes multi-agents, afin d'une part d'accélérer nos QGAs et d'autre part de tirer profit de l'avantage de la coopération des MASs.

Dans ce travail, nous proposons un système multi-agent quantique (MAS-QGA) en introduisant au sein des agents un QGA. Ainsi, un QGA parallèle est lancé et chaque Q-agent va être capable d'évoluer pour trouver les meilleures solutions d'ordonnancement.

Dans ce travail, notre objectif est d'optimiser le temps de réponse moyen au niveau d'un système temps réel embarqué distribués par une méthode hybride entre les systèmes multi-agents et QGA, pour ce but on va suivre la démarche suivante :

— implémenter un GA classique pour l'ordonnancement des tâches de type sporadique et périodique dans un système temps réel embarqué distribué,

— implémenter un QGA pour l'ordonnancement des tâches de type sporadique et

périodique dans un système embarqué distribué temps réel.

— concevoir et implémenter un MAS-QGA.

Première partie

La partie conception

5.2 La méthode GA classique pour l'ordonnancement dans un système embarqué distribué temps réel

Les GAs tentent de reproduire le processus de l'évolution naturelle dans un environnement donné et de la sélection proposée par Darwin [19].

Dans cette méthode, on va représenter notre système par deux graphe :

— Le premier graphe est un graphe orienté acyclique (DAG : Directed Acyclic Graph) pour décrire les dépendances entre les tâches. Chaque noeud représente une tâche temps réel et chaque arête représente une relation de précédente et comme nous avons deux types de tâches (sporadique et périodique) donc il faut séparer entre les deux graphes de ses deux type de tâche.

— Le second graphe est proposé pour décrire les connexions entre les processeurs. Les processeurs peuvent être connectés soit par un bus ou lien ou un pont. Les processeurs de notre approche sont hétérogènes et Les communications inter-processeurs peuvent s'effectuer par passage de messages.

On a le graphe d'application (TG : Task Graph) est un graphe orienté G(T,E) où chaque sommet $\tau \in$T représente une tâche de l'application et l'arc orienté (τ_i,τ_j) noté $e_{ij} \in$E représente les relations de dépendances entre les tâches τ_i, τ_j. Et le graphe d'architecture NT (NoC Topology graph) est un graphe orienté P(S,F) où chaque sommet $s_i \in S$ représente un nœud de la topologie et l'arc (s_i, s_j) noté par $_i \in F$ où F représente un lien physique direct entre les éléments de $s_i et s_j$ de l'architecture.

5.2.1 Codage des chromosomes

Un individu i est représenté sous forme d'une matrice binaire M[i,j] ou l'indice du colonne j représente le numéro de la tâche et l'indice du ligne représente le numéro du processeur de tels sorte que :

Si M[i,j]=1 alors la tâche j est affectée au processeur i
Sinon M[i,j]=0 alors la tâche n'est pas affectée.

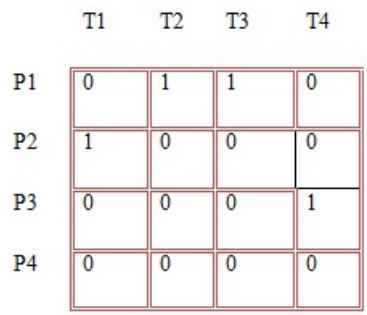


Figure 5.1 – Représentation du chromosome

5.2.2 Initialisation de la population

La population initiale est considérée comme un point de départ pour l'GA. Les générations futures sont au fur et à mesure améliorées afin d'accèder aux meilleures solutions, en appliquant les opérateurs génétiques.

Une bonne initialisation aide d'une manière considérable l'évolution du processus.

On trouve plusieurs mécanismes pour générer une population initiale. Parmi ces mécanismes : le tirage aléatoire, les heuristiques ou une combinaisons de solution heuristique et l'aléatoire. Dans notre approche, on va générer la population initiale de façon aléatoire.

Algorithm 3: L'algorithme de génération de la population initiale

```
Debut ;
Choisir Psize la taille de la population initiale;
K=1;
Tant que K ≠ Psize faire
    Pour j de 1 to m faire
        Pour i de 1 to n faire
            Générer un nombre aléatoire r pour chaque tâche pour tous les
            processeurs;
            Si r∈[0,5] alors
                M[i,j]=1 ;
            sinon
                M[i,j]=0 ;
    K=K+1 ;
Fin ;
```

5.2.3 Réparation du chromosome

Pour un chromosome, chaque tâche doit être allouée à un seul processeur et chaque processeur doit traiter au minimum une tâche.

5.2.4 Évaluation des individus

Cette partie consiste à calculer finesse pour chaque individu et pour cette raison on doit donner un ordre d'exécution de l'ensemble des tâches temps réel de notre système afin de calculer à la fin le temps de réponse moyen. Dans notre approche on utilise l'ordonnancement multiprocesseur par partitionnement qui consiste à affecter chaque tâche à un processeur avant de débuter leur exécution de sorte que les tâches d'une même partition sont affectées au même processeur et par la suite l'algorithme d'ordonnancement mono-processeur est utilisé pour ordonnancer les tâches sur chaque processeur avec la politique d'ordonnancement temps réel rate Monotonic (RM).

On a deux types de tâche périodique et sporadiques sauvegardées dans deux listes différentes selon leur échéance pour chaque processeur, donc chaque processeur possède deux listes une pour les tâches sporadiques attribuées à ce processeur et l'autre pour les tâches périodiques.

Pour l'ordonnancement des tâches sporadiques on utilise la méthode de differ server qui consiste à définir une tâche périodique comme une tâche serveur avec la plus haute priorité et avec une période définie. Et considérer le temps inter-arriver est la période des tâches sporadiques.

Lorsqu'une nouvelle tâche arrive, on vérifie si cette tâche est le serveur ou non.

Dans le cas ou cette tâche est le serveur

donc le processeur commence à exécuter les tâches sporadiques prêtes. Chaque tâche vérifie d'abord si elle est dépendante ou indépendante si elle est dépendante donc elle doit attendre que la tâche prédécesseur de cette tâche termine son exécution. Et si elle est indépendante donc on vérifie si le processeur est libre ou non. Si le processeur est libre la tâche sporadique s'exécute directement. Et dans le cas ou le processeur est occupé on vérifie la priorité de cette tâche avec la priorité de la tâche en cours d'exécution. Si la nouvelle tâche possède la plus haute priorité donc interrompre la tâche en cours d'exécution et l'insérer dans la liste des tâches sporadiques pour ce processeur et exécuter la nouvelle tâche. Si la nouvelle tâche ne possède pas la grande priorité alors la nouvelle tâche est insérée dans la liste des tâches sporadiques prêtes pour ce processeur.

Lorsqu'une tâche *Ti* termine son temps d'exécution, elle doit envoyer tous les mes-

sages sortant d'elle vers tous ses successeurs,et ensuite cette tâche est insérée dans la liste avec les nouvelles paramètres : prochaine date d'arrivée égale au temps inter-arrive + la date d'arrivée. Et calculer le temps de fin d'exécution.

Dans le cas où toutes les tâches sporadiques terminent leur temps d'exécution et il n'y a pas d'autre tâche dans la liste des tâches sporadiques pour ce processeur et la période de la tâche serveur n'est pas encore terminé donc le processeur exécute les tâches périodiques mais une fois une tâche sporadique arrive pendant cette période donc l'exécution de la tâche périodique est interrompue et on exécute la tâche sporadique, une fois la période du serveur est terminée donc la tâche serveur est suspendue et l'insérer dans la liste des tâches périodiques avec sa nouvelle date d'arriver égale au temps inter - arrivé + la date d'arrivée.

L'ordonnancement des tâches périodiques se fait de la même façon que les tâches sporadiques , l'ordonnanceur vérifie la liste des tâches périodiques prêtes de tous les processeurs et exécute les tâches périodiques qui sont ordonnancées selon leur échéance avant l'exécution de la tâche on vérifie d'abord si elle est dépendante ou indépendante. Dans le cas où elle est indépendante elle doit vérifier est-ce que le processeur est occupée ou libre dans le cas ou le processeur est occupé par les tâches sporadiques. Donc la tâche périodique doit attendre jusqu'à ce que ce processeur devienne libre. Et dans le cas ou le processeur est occupé par une autre tâche périodique donc on vérifie les priorités de ses tâches et exécuter celle avec la grande priorité.

Si une tâche périodique termine son temps d'exécution, alors elle doit envoyer tous les messages sortant d'elle, et ensuite elle est insérée dans la liste avec les nouvelles paramètres(la prochaine date d'arrivée égale à la période + la date d'arrivée). Et calculer le temps de fin d'exécution de cette tâche.

Les messages sont aussi ordonnancés en parallèle avec les tâches, une foi la tâche termine son temps d'exécution un message est envoyé par cette tâche(source du message) vers tous les processeurs associés à ses tâches successeurs pour les activer et être prête à s'exécuter.

L'ordonnanceur doit transférer le message sur le bus ou le processeur associé à la tâche de destination est connecté.

S'il existe un lien entre le processeur de la tâche source et le processeur de la tâche destination donc le message est transféré sur ce lien au lieu de Bus. Pour calculer le temps de réponse de toutes les tâches on calcule premièrement le temps de réponse pour

chaque tâche par la formule suivante :

$S = fin_{exe} - date_{arive}$

$TR_i = TR_i + S$

Ainsi le temps de réponse moyen pour chaque tâche est :

TRM*i*=$\frac{\text{La somme de tous les temps de réponse de la tâche i}}{\text{Le nombre d'activation de cette tâche}}$

Finalement, on obtient le temps de réponse du système égal à :

TRM*s*=: $\frac{\text{La somme de temps de réponse moyen de tous les tâches i}}{\text{Le nombre de tâches}}$

5.2.5 Sélection dans les GAs

Il existe plusieurs méthodes de sélection, telles que la sélection par roulette biaisée ou la sélection par la méthode de l'élitisme etc.

Dans notre approche on va utiliser une stratégie de sélection par Rang (Ranking) de classement : un rang Ri est associé pour chaque individu qui dépend essentiellement de son fitness, et par la suite on va classer tous les individus de la population dans une liste suivant un ordre croissant de leur fitness, une probabilité Pi d'être sélectionnée est affectée à chaque individu i, cette stratégie de sélection sera effectuée de façon proportionnelle au rang des individus dans la liste de la population :

$$Pi = \frac{R_i}{\sum_{i=1}^{n} R_i}$$

Algorithm 4: L'algorithme de sélection

Début ;
Pour *i = 1* **to** *N* **faire**
  trier les individus dans une liste ;
  Calculer la probabilité de sélection P_i;
  Générer un nombre aléatoire r∈[0,1];
  Si $r < P_i$ **alors**
    choisir l'individu i ;
Fin

5.2.6 Le croisement (Crossover)

L'opérateur génétique du croisement est un opérateur essentiel dans les GAs, il permet de mieux explorer l'espace de recherche. Après le choix de deux individus parent1 et parent2 parmi les individus de la population,on applique l'opérateur du croisement. Le but de cet opérateur est de générer deux nouvelles individus enfant1 et enfant2, en effectuant des échanges globaux entre les deux matérielles génétiques de ces deux parents.

Algorithm 5: L'algorithme du croisement

Début ;
Trier les individus sélectionner dans une liste selon la valeur du fitness ;
Choisir le premier individu $indiv1$ et le dernier individu $indiv2$ de cette liste ;
Choisir de façon aléatoire un processeur k ;
Croisement ;
Pour $i = 1$ **to** m **faire**
    Si $i \neq k$ **alors**
        gardé le même séquencement de tâche des individus $indiv1$,$indiv2$;
    sinon
        attribué au $enf1$ le même séquencement de tâche que $indiv2$;
        attribué au $enf2$ le même séquencement de tâche que $indiv1$;
Fin ;

Donc on va choisir de façon aléatoire un processeur k et on applique un croisement entre la séquence des tâches associées à ce processeur seulement, le séquencement des tâches sur les autres processeurs reste le même sur les fils.

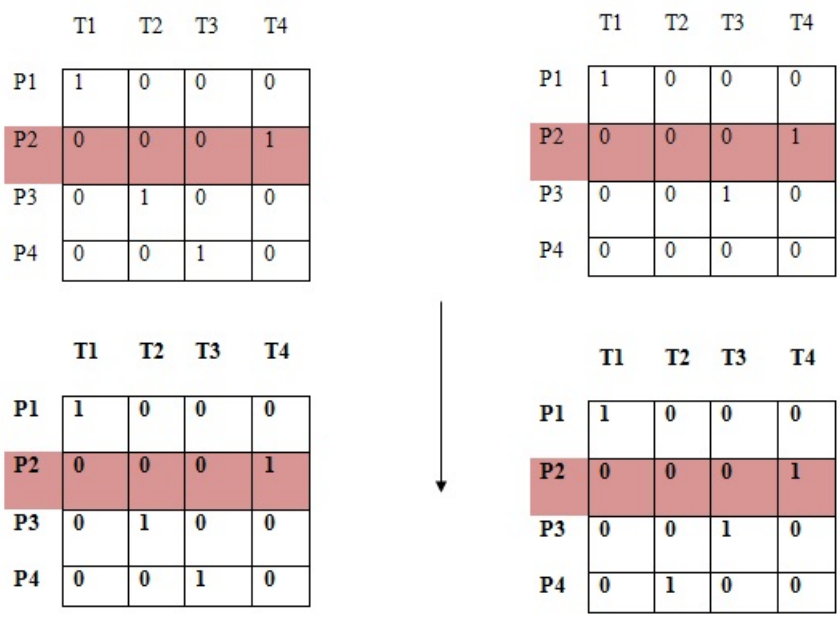


Figure 5.2 – Exemple de croisement

5.2.7 Mutation

Une mutation est une perturbation introduite sur la composante de l'individu afin de garantir la diversité et élargir le champ d'exploration. Permet d'échapper aux minimums locaux.

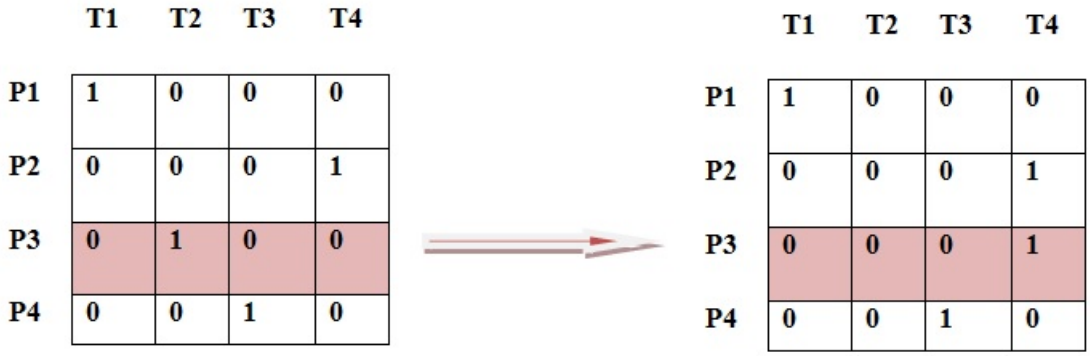


Figure 5.3 – Exemple de mutation

Algorithm 6: L'algorithme de mutation

Début ;
Choisir de façon aléatoire un individu i;
Choisir de façon aléatoire un processeur k ;
Calculer la Probabilité de mutation P*mut*;
Générer un nombre aléatoire r∈[0,1] ;
Si *r ∈ [0,Pmut]* **alors**
  appliqué l'opérateur de mutation sur cet individu ;
Pour *h=1* **to** *m* **faire**
  Si $h \neq k$ **alors**
    attribué au *Indiv* muté le même séquencement de tâches que *indiv* i ;
  sinon
    Choisir de façon aléatoire deux affectations M[i,j] et M[i',j];
    Permuter l'ordre de ces deux affectations ;
Fin ;

5.2.8 Remplacement

Dans cette étape, les nouveaux individus obtenus à partir des parents et des enfants de la génération courante après l'application des opérateurs de croisements et mutations, constitue la nouvelle population de la génération suivante à partir des parents et des enfants de la génération courante.

Algorithm 7: L'algorithme de remplacement

Début ;
trier les individus de la population parents dans une liste selon la valeur de fitness associe à ces individus ;
trier les individus de la population enfants dans une liste selon la valeur de fitness associe à ces individus ;
choisir les K meilleurs individus parmi les enfants et les Parents ;
Fin

5.2.9 Critère d'arrêt

Dans notre approche, nous avons considéré le critère de convergence un nombre maximum d'itération. A chaque itération, la meilleure solution est sauvegardée dans une liste nommée *bestsolution*, donc la taille de cette liste égale au nombre de générations et les meilleures solutions de cette liste sont alors considérées comme solutions au problème.

5.3 Un QGA pour l'ordonnancement dans un système embarqué distribué temps réel

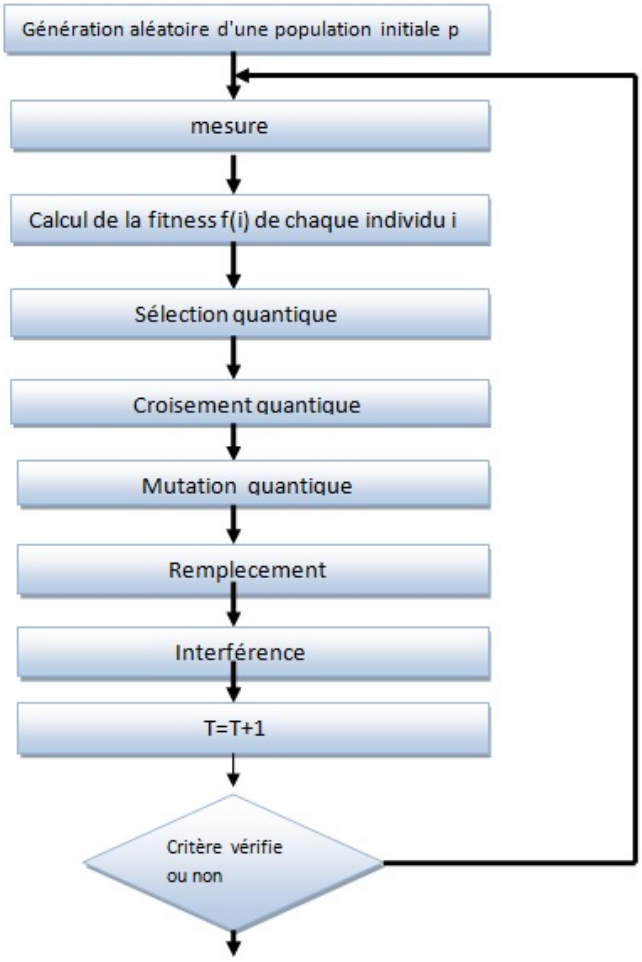


Figure 5.4 – Principe du fonctionnement d'un QGA

Un GA classique demande un nombre considérable d'itérations pour converger et une très grande taille de la population exigée pour trouver une bonne solution.

Afin de pallier ces inconvénients premièrement une hybridation est faite entre les GAs et l'informatique quantique. Dans cette approche appelée QGA, cette hybridation permet d'intensifier la recherche de solutions et s'assurer de rester autour des zones les plus prometteuses ainsi de réduire le nombre d'individus grâce au codage quantique (quatum chromosome).

Après les expérimentations effectuées nous avons constater qu'en appliquant l'opérateur de l'interférence tous seul le temps de calcul est réduit par a port dans le cas ou on applique les autres opérateurs génétiques telle que la sélection, le croisement et la mutation.

5.3.1 Codage des chromosomes

Comme l'élément de base dans QGA est le qubit, un chromosome est tout simplement représenté sous forme d'une matrice de qubits.

5.3.2 Génération de la population initiale

La façon la plus simple consiste à donner la même probabilité à tous les états de superpositions. Donc, toutes les amplitudes des qubits ont la même valeur.

5.3.3 Évaluation des individus

5.3.3.1 La fonction mesure

Cette fonction consiste à extraire un individu classique à partir d'un autre quantique. L'objectif principal de cette transformation est d'évaluer les individus de la population en fonction des individus classiques extraits.

Algorithm 8: Algorithme de mesure du qubit

Début ;
Générer un nombre aléatoirement r $\in$ **[0,1]** ;
Si $r > \alpha^2$ **alors**
  retourner 1 ;
sinon
  retourner 0 ;
Fin ;

5.3.3.2 Réparation

On peut observer dans les chromosomes binaires après l'application de la fonction mesure certains cas inacceptables comme l'affectation d'une tâche a plusieurs processeurs ou une tâche n'est affectée à aucun processeur et aussi aucune tâche n'est affectée à un processeur donc pour éviter ces cas on a utilisé une fonction de réparation.

5.3.3.3 Calcul de la fitness

Notre objectif est d'estimer le temps de réponse pour chaque individu. On va utiliser la même stratégie que dans GA classique car nous avons des chromosomes binaires.

5.3.3.4 Interférence

Cette opération permet de modifier les amplitudes d'un individus en fonction du meilleur individu trouvé. Cette opération est utile pour intensifier la recherche autour de la meilleure solution. Peut être définit comme étant une transformation unitaire qui permet une rotation spatiale dont l'angle est en fonction des amplitudes (a_i, b_i) et de la valeur du bit correspondant à la solution référence. La population est donc mise à jour avec une rotation. Notre politique de rotation est donnée par la formule :

$$\begin{bmatrix} \alpha\prime_{ij} \\ \beta\prime_{ij} \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta ijXs(\alpha ij, \beta ij)) - \sin(\Delta\theta ijXs(\alpha ij, \beta ij)) \\ \sin(\Delta\theta ijXs(\alpha ij, \beta ij))\cos(\Delta\theta ijXs(\alpha ij, \beta ij)) \end{bmatrix} \begin{bmatrix} \alpha ij \\ \beta ij \end{bmatrix}$$

La valeur $\Delta\theta i$ est déterminée par la table de recherche.

xij	bij	$f(x) > f(b)$	$\Delta\theta$ij	S (aij, bij)			
				aij* bij > 0	aij* bij < 0	aij=0	bij = 0
0	0	false	θ_1	±	±	±	±
0	0	true	θ_2	±	±	±	±
0	1	false	θ_3	±	±	±	±
0	1	true	θ_4	±	±	±	±
1	0	false	θ_5	±	±	±	±
1	0	true	θ_6	±	±	±	±
1	1	false	θ_7	±	±	±	±
1	1	true	θ_8	±	±	±	±

Tableau 5.2 : Table de recherche générale pour la rotation des portes quantiques.

Dans notre approche les meilleurs résultats sont obtenu avec les valeurs de θ suivant : 0.02 π ,0 ,0.01 π.

5.4 Un MAS-QGA pour l'ordonnancement dans un système embarqué distribué temps réel

Dans ce travail, nous avons adopté l'approche distribuée pour développer une approche efficace d'ordonnancement dans les systèmes embarqués temps réel. Il s'agit de tirer profit des avantages des :GAs , QGAs et des MASs.

En effet, l'objectif de notre MAS-QGA est d'utiliser un ensemble d'agents quantiques basés sur un QGA coopérant d'une manière efficace pour trouver la bonne solution de notre problème.

En effet, dans notre MAS-QGA, on considère un ensemble de Q-agents qui interagissent en vue de la réalisation d'un objectif local commun, et dans lequel des messages sont échangés dans un ordre coopératif pour l'achèvement d'un objectif global.

Un des avantages évidents qu'offrent les MASs, c'est qu'on peut ajouter d'une manière simple un nouveau paradigme au niveau des agents ; par exemple l'intégration de d'autres méthodes d'optimisation (PSO comme un savoir-faire par exemple), ce qui donne une robustesse et une extensibilité à notre méthode.

Par ailleurs, les MASs sont massivement parallèlisables, on pensera alors à un gain de performance se traduisant par un temps de calcul nettement moins long.

En effet, notre approche consiste à lancer plusieurs Q-agents dans une activité compétition/coopération utilisant chacun un savoir-faire pour chercher la meilleure configuration possible du problème.

Dans ce travail, notre architecture choisie pour MAS-QGA est le modèle en îles (island model) (voir la Figure 5.6). Cette architecture est composée d'un ensemble d'agents d'optimisation quantique (Q-agents).

Dans ce modèle d'ıles, l'organisation des agents est régie par une topologie qui peut affecter la tolérance à la convergence locale. On trouve dans ce modèle un ensemble de règles dont leur rôle est de définir les interactions entre les agents : périodicité dans le nombre de générations, périodicité temporelle, ainsi que la qualité d'une solution acceptable, etc.

Techniquement, pour intégrer les nouvelles meilleures solutions arrivées il faut attendre la fin d'une génération (à cause du fait que le contrôle de ce qui se passe entre le début et la fin d'une génération est difficile). Pour réaliser cette approche nous avons besoin de plusieurs étapes :

5.4.1 Politique d'intégration

Il faut définir une manière dont laquelle l'intégration des nouveaux individus est faite au sein De chaque agent. Donc on peut utiliser le même mécanisme de remplacement de GA lors d'une génération.

5.4.2 Politique de migration

Les deux principales questions dans la politique de migration est quand la migration est faite (exemple : Toutes les 10 générations) et quels sont les individus à migrer (exemples : Les 10 meilleurs individus).

5.4.3 Critère d'arrêt

Le modèle doit s'arrêter lorsque plus aucun agent n'est en activité. Pour savoir si un agent est arrêté il suffi de regarder le statut de QGA qui tourne on lui, i.e. convergence.

Dans un exemple, on lance un système constitué de trois QGA. Ces trois algorithmes d'optimisation sont en coopération, la nature de cette coopération c'est une migration des meilleures solutions.

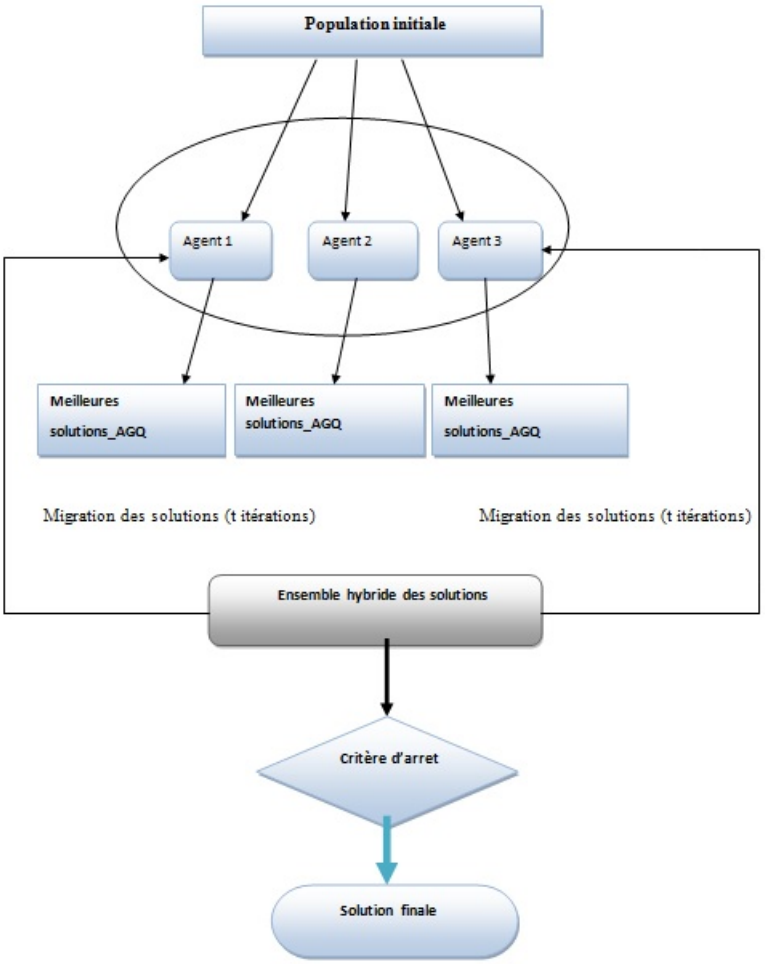


Figure 5.5 – Schéma descriptif de l'approche proposée et la migration des meilleures solutions entre les agents

L'objectif principal de cette étape consiste à trouver la bonne solution relative au bon ordonnancement dans un temps réduit (i.e. en réduisant le nombre d'évaluation et le temps d'exécution), puisqu'on traite des systèmes temps réels et dont le temps de

réponse est critique. Dans notre contexte on va lancer un système multi–agents dont le but est de lancer plusieurs QGA en parallèle.

Donc, chaque agent quantique (Q-agent) exécute un algorithme génétique quantique (QGA), la sortie de ce Q-agent est une liste des meilleures solutions. Ce modèle distribué peut être facilement mis en œuvre dans les ordinateurs parallèles. Ces algorithmes sont en coopération.

Cette activité de compétition/coopération assure que ce processus distribué accède aux meilleurs résultats récupérés donc le concept et les caractéristiques de l'agent et le codage quantique sont introduits pour la recherche intelligente élevée.

Chaque agent quantique caractérisé par un bit quantique, est défini comme étant une solution candidate pour le problème d'optimisation d'ordonnancement de tâches.

Dans ce travail, différents agents utilisent QGA qui sont codées dans une population Q(t) de n individus, chaque chromosome q est constitué de m qubits, chaque individu codé par un qubit il ne représente pas seulement une solution, mais toutes les solutions possibles au problème.

La taille de la population initiale pour chaque agent est $n * L'$ ou n est le nombre d'agents dans les systèmes c.a.d le nombre de QGA lancé, et l' sont Les meilleures solutions présentées par chaque agent.

Nous avons utilisé les QGAs dans un système multi-agents pour exploiter les avantages de parallélisme afin d'obtenir la solution la plus performante dans un laps de temps réduit que QGA tout seul et aussi minimisait les itérations par chaque algorithme. Et grâce à la stratégie de migration, les agents échangeaient les bonnes solutions jugées sélectionnées parmi les descendants (voir Figure5.5). Chaque agent contiendra un ensemble de bonnes solutions actuelles. Dans cette stratégie présenté avec un modèle entièrement connecté (voir Figure 5.6), au sein de chaque agent, un QGA standard est exécuté sur un ensemble de chromosomes initiaux afin de choisir une nouvelle meilleure ensemble de chromosomes dans le but de produire de bonnes initialisations(voir figure 5.7).

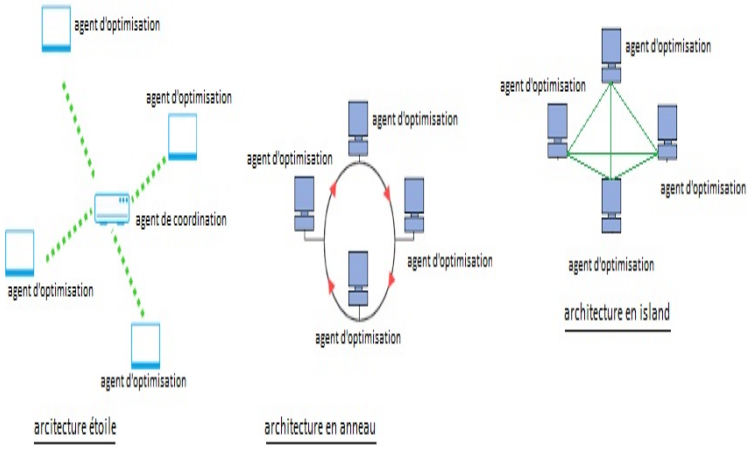


Figure 5.6 – Architecture des MASs

Pour chaque cycle, chaque agent reçoit les meilleurs individus jugés par les autres agents, effectue QGA sur L'ensemble d'individus actuels, mis à jour les meilleures solutions puis retransmet cet nouvel ensemble de solutions à d'autres agents.

Dans la première itération, la population initiale va être générée de façon aléatoire par chaque agent mais à partir de la deuxième évolution la population initiale sera constituée des solutions migrées. Les solutions migrées sont les l meilleures solutions ,et l'algorithme accepte seulement l solutions si la taille de la population en entrée $l = n * l'$ ou n :est le nombre d'agents dans notre système.

Dans notre approche on ajoute une condition qui consiste à définir le nombre de générations de QGA égale au nombre de meilleures solutions à migrer.

Entrées
l : la taille de la population.
Pmut : la probabilité de mutation,
Pc : la probabilité de croisement,
Num.evol : le nombre maximal des évolutions (le nombre des migrations),
itr : le nombre des générations de l'algorithmes
l' : le nombre des meilleures solutions à migrer(l'<l)
Sortie :
Les l meilleures solutions à migrer,
best : meilleure solution,

Au début, il faut d'abord modifier QGA et retirer l'étape de génération de la population initiale. Donc, l'algorithme général sera comme suit :

Algorithm 9: L'algorithme de l'approche proposée

Début ;
Générer la population initiale de la même façon que QGA ;
Pour *j de 1* **to***Num* **faire**
  Lancer QGA. ;
  sélectionner les l meilleures solutions et les migrées ;
  Récupérer les meilleures solutions produites par les autres agents (migrées) ;
  Hybrider l'ensemble des solutions récupérer avec les l meilleures solutions de cette agent ;
Fin ;

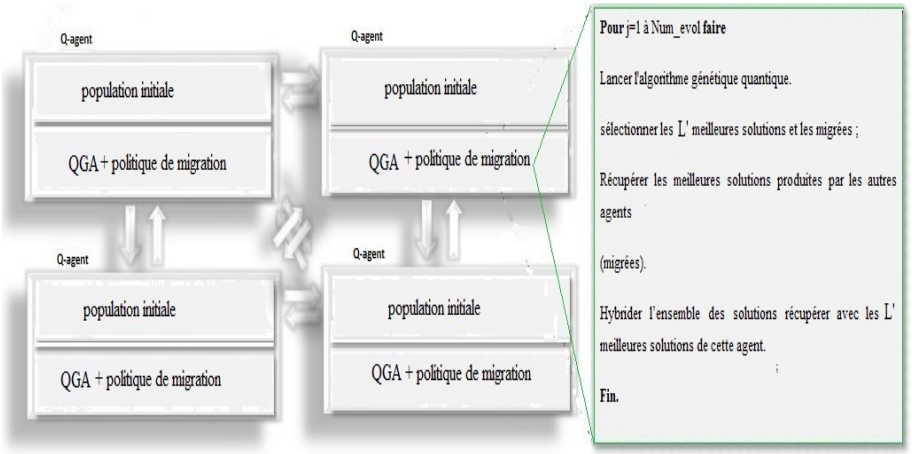


Figure 5.7 – L'architecture du MAS-QGA

Deuxième partie

La partie expérimentale

5.5 Matériels et Logiciels utilisés

Notre algorithme a été implémenté en utilisant NetBeans IDE 7.4 sur un ordinateur de 4 Go de RAM sur un système d'exploitation 64-bits et un processeur AMD E2-1800 APU de 1.70 GHz.

Plusieurs raisons qui nous ont conduit à utiliser NetBeans, on peut citer :

— La capacité de supporter un grand nombre de langages de programmation (Java, C, C++ ,PHP, HTML . . .).

— Il est constitué d'un éditeur en couleur, projets multi-langage, éditeur graphique d'interfaces et de pages Web.

— Il permet de faire appel à des bibliothèques externes.

Dans nos expériences, l'unité de temps de simulation désigne la durée moyenne qu'un individu relatif à un ordonnancement peut réaliser sur l'architecture considérée.

5.6 Tests et comparaisons

Dans cette, partie, l'objectif est de montrer la validité et l'efficacité de notre MAS-QGA en la comparant avec les méthodes GA classique et l'approche QGA.

5.6.1 La méthode GA classique

Dans un premier temps, on a effectué plusieurs tests en appliquant GA classique. Nous présentons dans ce mémoire trois exemples. Chaque exemple représentent un graphe de tâche et un diagramme d'architecture.

5.6.1.1 Premier exemple :

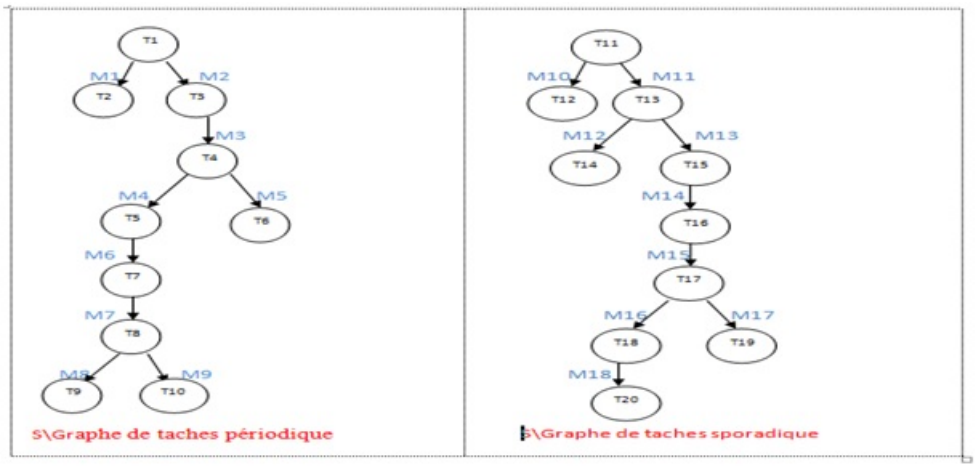


Figure 5.8 – Graphe de tâches du premier exemple

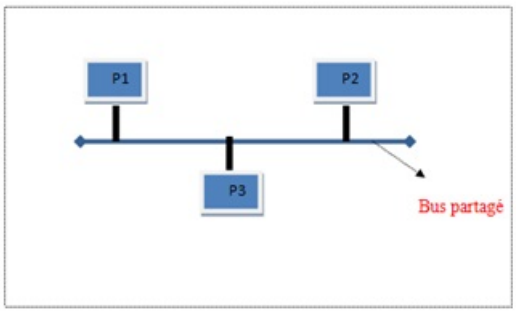


Figure 5.9 – Graphe d'architecture du premier exemple

5.6.1.2 Deuxième exemple :

Dans le deuxième exemple nous gardons le même graphe de tâches du premier exemple, et on augmente le nombre de processeurs.

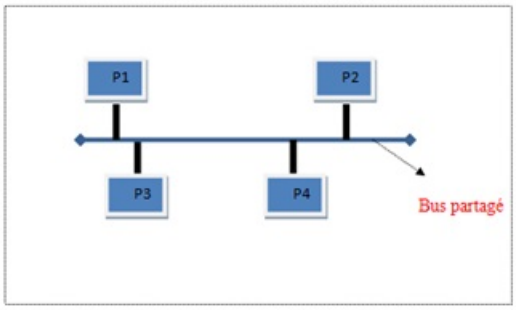


Figure 5.10 – Graphe d'architecture du Deuxième exemple

5.6.1.3 troisième exemple :

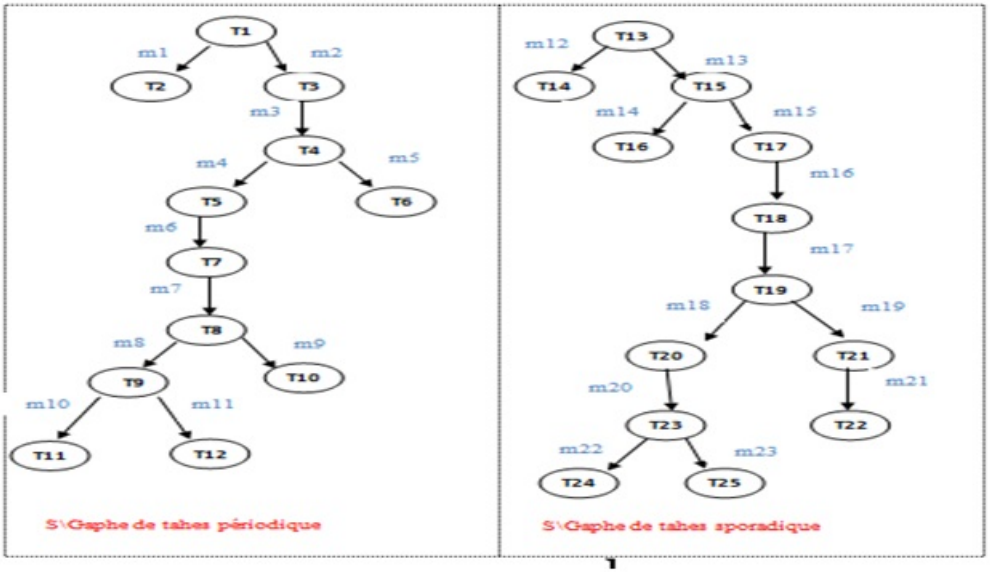


Figure 5.11 – Graphe de tâches du troisième exemple

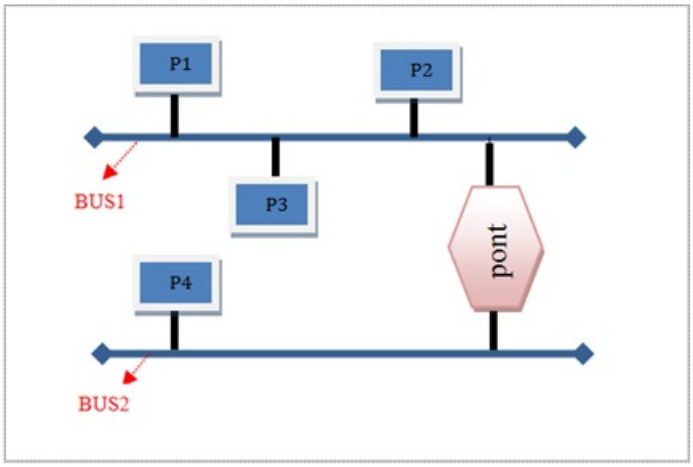


Figure 5.12 – Graphe d'architecture du troisième exemple

Nous présentons dans le tableau ci-dessous (voir Table 5.I) les différents jeux de paramètres appliqués aux trois exemples, sur une architecture simple. Nous présentons aussi quelques résultats que nous avons obtenus en appliquant l'GA classique (voir Figure 5.13).

Nb Tach	Nb Cpu	Nb génération	Taille-population	Tps simulation	Nbr-Bus
20	03	100	30	80	1
20	4	100	30	80	1
25	4	100	30	80	2

Tableau 5.I – Les différents jeux de paramètres

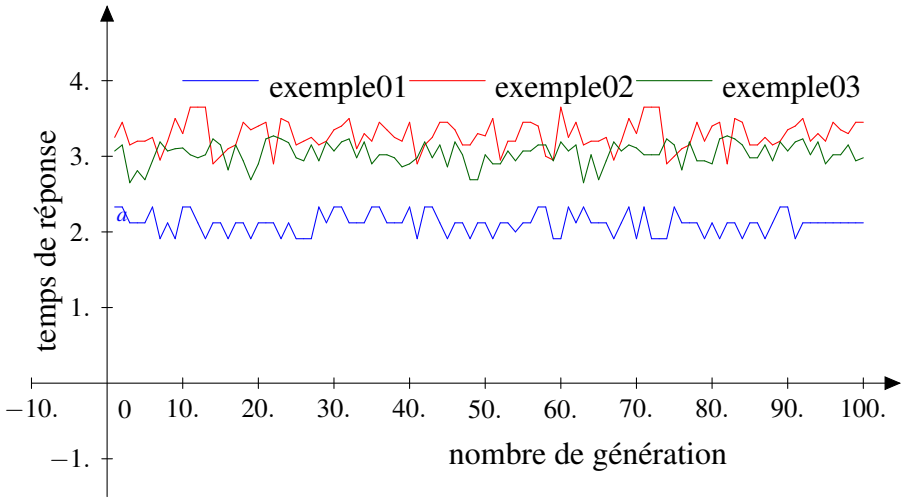


Figure 5.13 – Résultats des trois exemples en appliquant GA

5.6.2 Discussion

Dans les courbes de la Figure 5.13 on a représenté le temps de réponse moyenne (par unité de temps simulé) du meilleur individu (ordonnancement) trouvé pour chaque génération. On remarque que la plus grande valeur de temps de réponse de ce graphe est atteinte par le deuxième exemple c.a.d dans le cas ou le nombre du processeur est grand et le nombre de tâches est petit et dans le premier exemple malgré que le nombre du processeur est petit et avec le même nombre de tâches que le deuxième exemple ; le temps de réponse atteint les meilleures valeurs.Et on remarque aussi que le nombre de processeurs dans le troisième exemple est égale au nombre du processeur dans le deuxième exemple et le nombre de tâches est plus grand dans le troisième exemple mais ce dernier fournit de meilleures valeurs de temps de réponse que le deuxième exemple on peut expliquer cette remarque par l'augmentation du nombre de bus dans le troisième exemple ; qui sert à minimiser le conflit des messages par rapport au cas d'un seul bus ce qui accélère leurs transferts et donc minimiser le temps de réponse.

5.6.3 Expérimentation sur la méthode QGA

Dans un second temps, nous avons appliqué un QGA pour montrer l'amélioration par rapport à GA classique. En effet, nous constatons que les résultats obtenus par QGA sont meilleurs que les résultats obtenus par GA classique, ce qui valide l'approche théorique. Dans ces tests, le temps de réponse moyen atteint une valeur minimale égale à 0,95. Les mêmes remarques sont signalées à propos du nombre de processeurs et de bus ; si le nombre de processeurs est grand et le nombre de bus est petit donc le temps de réponse va être grand.

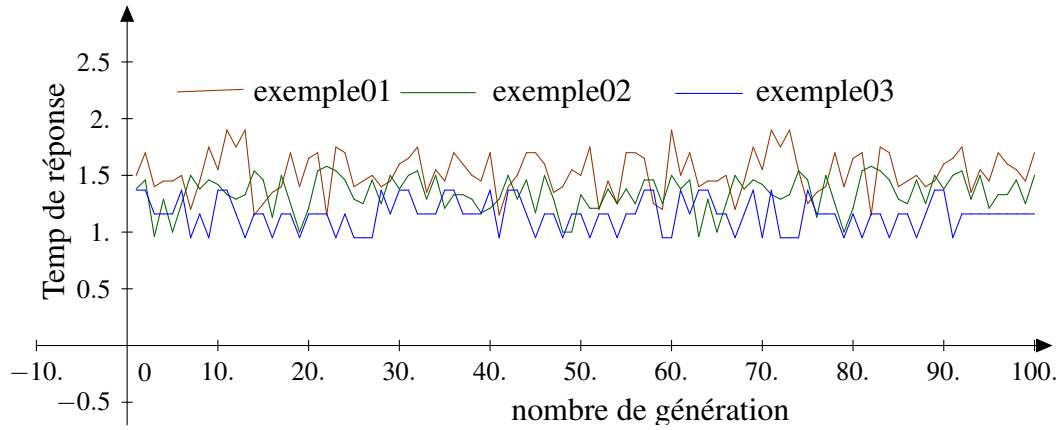


Figure 5.14 – Résultats des trois exemples en appliquant QGA

5.6.4 Expérimentation sur l'approche MAS-QGA

Afin de valider notre méthode MAS-QGA et démontrer sa robustesse, nous avons refait les mêmes tests sur les méthodes GA classique et QGA en comparant avec MAS-QGA, en fixant les paramètres de la première stratégie.

La Figure 5.15 montre le code source d'un Q-agent utilisé.

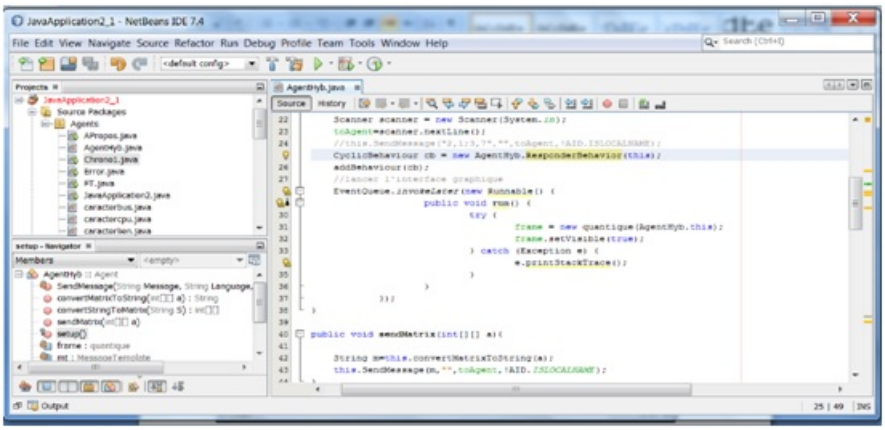


Figure 5.15 – Le code source de l'agent

La Figure 5.16 présente l'exécution de l'agentA et la réception d'un message après la fin de la première itération. Ce message contient les meilleurs solutions envoyées par l'agentB.

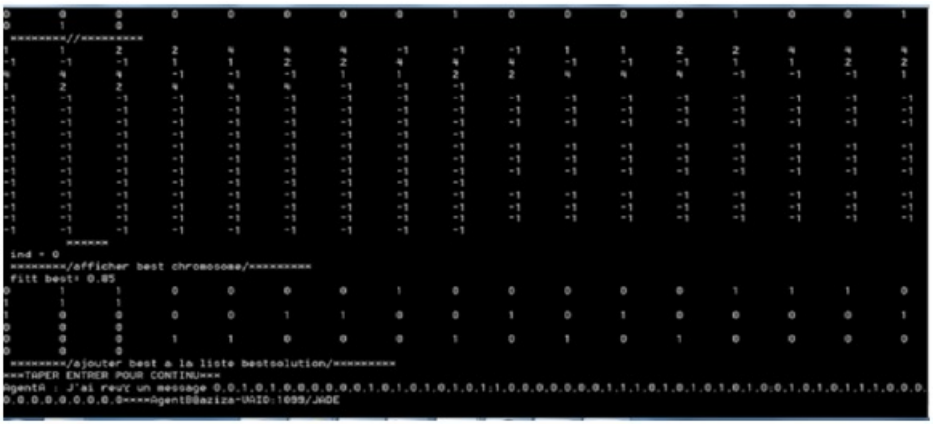


Figure 5.16 – L'exécution de l'agentA et la réception des meilleures solutions

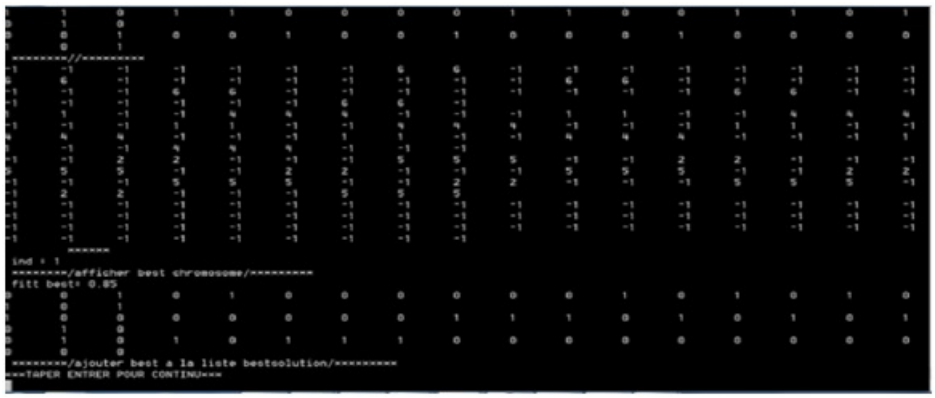


Figure 5.17 – L'exécution de l'agent B et l'affichage des meilleures solutions

La Figure 5.18 présente le graphe d'évolution de deux Q-agents A et B. A chaque cycle, les Q-agents cherchent les meilleurs configurations relatives aux ordonnancement réalisant le meilleurs temps de réponse. On remarque du graphe, même aux itérations fin de cycle, les Q-agents restent toujours à la recherche des bonnes solutions. Ceci signifie que notre MAS-QGA tente d'éviter les minima locaux.

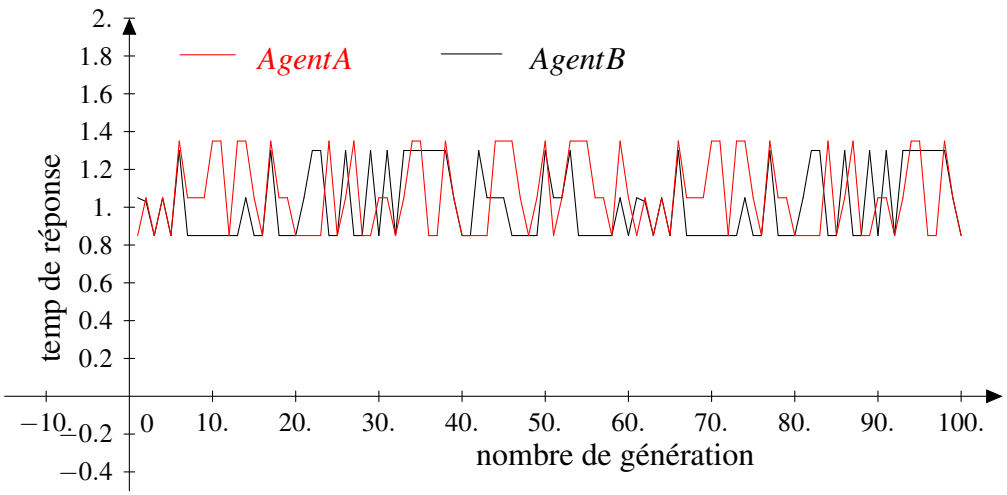


Figure 5.18 – Graphe d'évolution de Q-agents

Les graphes de la figure 5.19 résument les résultats des trois approches.

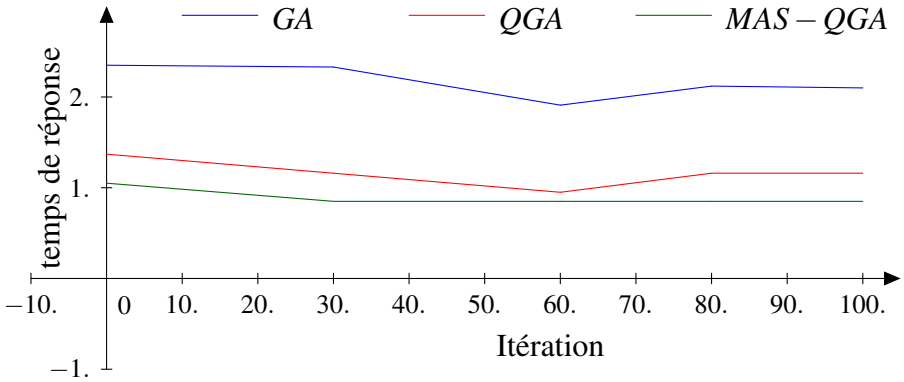


Figure 5.19 – La comparaison des méthodes GA, QGA et notre méthode MAS-QGA. Ces courbes montrent l'évolution de la recherche des bons ordonnancement appliquée sur des jeux de tests.

5.6.5 Discussion

Dans les courbes reportées dans la figure 5.18, on remarque bien que le temps de réponse moyen atteint une valeur minimale, ce qui signifient que notre approche MAS-QGA offre des meilleurs résultats comparant avec GA et QGA.

Selon les courbes reportées dans la Figure 5.19, on remarque que les résultats obtenus par QGA sont nettement meilleurs que ceux obtenus par GA. Mais, quand on applique notre MAS-QGA les résultats obtenus sont remarquablement excellent et stable en les

comparant ces résultats avec ceux du QGA.

En effet, l'algorithme QGA n'obtient la meilleure solution que dans l'itération 60, mais après cette itération, le QGA n'arrive pas à trouver mieux. Bien au contraire il obtient des solutions de mauvaises qualités.

Cependant notre MAS-QGA et grâce à la coopération basée sur la migration des bon solutions entre les Q-agents notre MAS-QGA accède rapidement aux bons solutions , mais également ce système arrive à maintenir une certaine stabilité en matière de la qualité des solutions obtenues.

En conclusion,dans l'approche MAS-QGA le temps de réponse moyen atteint une meilleure valeur égale à 0,85, et l'algorithme converge vers la solution optimale plus rapidement que l'algorithme QGA. En plus, MAS-QGA utilise un très nombre réduit de générations et une petite taille de population, plus petit que les autres stratégies (QGA,GA), et ceci grâce au principe de la distribution (MAS ou "island model").

Après l'exécution des trois codes des méthodes GA, QGA et MAS-QGA sur la même configuration matériel et logiciel citée, on trouve que le temps de convergence nécessaire pour que la méthode obtient les résultats finaux sont :

— le temps de convergence de la méthode GA égale à 14 minutes.

— le temps de convergence de la méthode QGA égale à 10 minutes.

— le temps de convergence de notre approche MAS-QGA égale à 9 minutes.

Bien que le temps d'exécution de la méthode QGA est nettement meilleur que celui de la méthode GA, notre MAS-QGA et sans un parallélisme réel notre approche MAS-QGA obtient les bon résultats mieux que les autres méthodes.

Le temps d'exécution de la méthode QGA est mieux la méthode GA, cela est dû au fait que dans QGA, on n'a pas utilisé les opérateurs génétiques parce que la diversité est garantie par le qubit.

L'approche MAS-QGA nécessite un temps plus réduit par rapport aux approches QGA et GA, car dans MAS-QGA le nombre d'itérations et la taille de la population sont réduits grâce au principe de divide and conquer (MAS ou island modèle).

Par ailleurs, le temps nécessaire dans QGA est très proche du temps nécessaire dans MAS-QGA, et cela est causé par les interactions entre les Q-agents et que notre algorithme n'est pas exécuté dans une architecture parallèle.

5.7 Conclusion

Dans cette section, nous avons proposé une nouvelle approche MAS-QGA pour l'ordonnancement de taches dans un système embarqué distribué temps réel. Dans un premier temps, nous avons présenté les approches basées sur algorithmes génétiques pour l'ordonnancement dans un système embarqué distribué temps réel.

Dans un second temps, nous avons présenté une amélioration de GA classique en introduisant le quantique dans le but de réduire la taille de la population de GA.

En fin, afin d'améliorer les performances de ces deux approches, nous avons proposé un MAS-QGA pour améliorer l'efficacité des systèmes embarqués distribués temps réel.

Cette résolution approchée est fondée principalement sur une combinaison efficace entre les systèmes multi-agents et les algorithmes génétiques quantiques pour l'exploration efficace de l'espace de recherche. En effet, les Q-agents ayant un savoir-faire comme un QGA approprié au problème en question coopèrent pour accéder aux meilleures solutions. Ce processus de compétition/coopération offre une nette supériorité à l'approche proposée par rapport aux approches similaire.

CONCLUSION ET PERSPECTIVES

Les problèmes d'optimisation sont de plus en plus complexe et le développement rapide de la technologie a fait l'utilisation du calcul parallèle de plus en plus nécessaire.

En outre, le rapport coût/performance dans les systèmes informatiques parallèles ne cesse de diminuer.

Le calcul parallèle et distribué est utilisé dans la conception et la mise en œuvre de méta-heuristiques pour accélérer la recherche, et pour améliorer la qualité des solutions obtenues, afin d'améliorer la robustesse et à résoudre les problèmes à grande échelle.

Dans ce mémoire, notre recherche est faite sur le problème d'ordonnancement multiprocesseur dans les systèmes embarqués distribués temps-réel. Ce type de problèmes appartient à la classe des problèmes NP-difficile.

Nous avons proposé une nouvelle approche efficace MAS-QGA pour la résolution de problème d'ordonnancement de tâche dans un système embarqué temps réel, basée sur la combinaison des algorithmes génétiques quantiques et les systèmes multiagent.

Le système MAS-QGA est composé de N Q-agents inter-connectés. A partir d'une configuration quantique initiale, chaque Q-agent calcule sa propre configuration adéquate, pour l'optimisation d'ordonnancement de tâches en temps réel.

Ensuite, il envoi les meilleurs résultats aux différents Q-agents. Ces Q-agents reçoivent les informations, appliquent une autre fois le processus d'optimisation avec la nouvelle population. Le MAS-QGA répète ce processus un nombre d'itérations.

En fait, ce processus de compétition/coopération offre aux différent Q-agents une efficacité d'obtenir les bonnes configurations comme le montrent les résultats expérimentaux.

Grâce à la modularité qu'offre le concept des MASs, on peut intégrer d'autre méthodes aux Q-agents pour améliorer la performance de notre approche MAS-QGA. En effet, comme perspectives, nous envisageons de proposer d'autres approches d'optimisation inspirée de la nature comme par exemple les PSOs, la colonie de fourmis, les systèmes abeilles, et les techniques inspirées du DNA computing.

Nous envisageons également à utiliser l'apprentissage par renforcement dans l'optimisation, afin d'accélérer plus le processus.

BIBLIOGRAPHIE

[1] Selma Azaiez. *Approche Dirigée par les modèles pour le développement de systèmes multi-agents*. Thèse de doctorat, Université de Savoie, 2007.

[2] Thomas Back, David B Fogel et *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[3] Thomas Bäck et Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[4] Gregory John Barlow. Improving memory for optimizatioin and learning in dynamic environments. 2011.

[5] Sanjoy Baruah et Joël Goossens. Scheduling real-time tasks : Algorithms and complexity. *Handbook of scheduling : Algorithms, models, and performance analysis*, 3, 2004.

[6] Lalatendu Behera et Durga Prasad Mohapatra. Schedulability analysis of task scheduling in multiprocessor real-time systems using edf algorithm. Dans *Computer Communication and Informatics (ICCCI), 2012 International Conference on*, pages 1–9. IEEE, 2012.

[7] Djamila Bouhalouan, Nassima Aissani et Bouziane Beldjilali. Proposition d'un modèle pour ordonnancement d'un système automatisé de production applications des algorithmes génétiques hybrides. Dans *CIIA*, 2009.

[8] Mohamed-Lamine Boukhanoufa. *Adaptabilité et reconfiguration des systèmes temps-réel embarqués*. Thèse de doctorat, Université Paris Sud-Paris XI, 2012.

[9] Fateh Boutekkouk et Soumia Oubadi. Periodic/aperiodic tasks scheduling optimization for real time embedded systems with hard/soft constraints. *IT4OD*, page 135, 2014.

[10] Samia Bouzefrane, J Etienne et Claude Kaiser. Gestion de la surcharge dans les systèmes de gestion de base de données temps réel. *Technique et Science Informatiques*, 27(7):879–910, 2008.

[11] Jason Brownlee. *Clever algorithms : nature-inspired programming recipes*. Jason Brownlee, 2011.

[12] Giorgio C Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[13] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson et Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook on scheduling algorithms, methods, and models*, pages 30–1, 2004.

[14] Christophe Caux, Henri Pierreval et M-C Portmann. Les algorithmes génétiques et leur application aux problèmes d'ordonnancement. *Automatique-productique informatique industrielle*, 29(4-5):409–443, 1995.

[15] Albert MK Cheng. *Real-time systems : scheduling, analysis, and verification*. John Wiley & Sons, 2003.

[16] Shu-Chen Cheng, Der-Fang Shiau, Yueh-Min Huang et Yen-Ting Lin. Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints. *Expert Systems with applications*, 36(1):852–860, 2009.

[17] Maxime Chéramy, Anne-Marie Déplanche et Pierre-Emmanuel Hladik. Ordonnancement temps réel : des politiques monoprocesseurs aux politiques multiprocesseurs. 2012.

[18] Milton Corrêa et Helder Coelho. Agent's programming from a mental states framework. Dans *Advances in Artificial Intelligence*, pages 31–39. Springer, 1998.

[19] Charles Darwin. *On the origin of species*. New York :D. Appleton and Co.,. URL `http://www.biodiversitylibrary.org/item/71804`. http ://www.biodiversitylibrary.org/bibliography/28875.

[20] Noël De Palma, Sara Bouchenak, Fabienne Boyer, Daniel Hagimont, Sylvain Sicard et Christophe Taton. Jade, un environnement d'administration autonome. *Technique et Science Informatiques*, 27(9-10):1225–1252, 2008.

[21] Suman Debnath et R Narayan. The optimization in 13 level photovoltaic inverter using genetic algorithm. *International Journal of Engineering Research and Applications*, 2(3):385–389, 2012.

[22] Michael L. Dertouzos et Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Eng.*, 15(12):1497–1506, 1989. URL `http://doi.ieeecomputersociety.org/10.1109/32.58762`.

[23] Marco Dorigo et Thomas Stützle. The ant colony optimization metaheuristic : Algorithms, applications, and advances. Dans *Handbook of metaheuristics*, pages 250–285. Springer, 2003.

[24] Lee D Erman, Frederick Hayes-Roth, Victor R Lesser et D Raj Reddy. The hearsay-ii speech-understanding system : Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, 12(2):213–253, 1980.

[25] Nicolas Esposito. Les agents mobiles–une introduction. *DASSAULT SYSTEMES–Recherche et nouvelles technologies, France*, 2000.

[26] Jacques Ferber et Jean-François Perrot. *Les systèmes multi-agents : vers une intelligence collective*. InterEditions, 1995.

[27] Tim Finin, Rich Fritzson, Don McKay et Robin McEntire. KQML—a language and protocol for knowledge and information exchange. Dans *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, 1994.

[28] Dominique Francisci. Algorithmes évolutionnaires et optimisation multi-objectifs en data mining. *Laboratoire informatique, signaux et systèmes de Sophia Antipolis UMR*, 60702002, 2002.

[29] Christian Gagne. Algorithmes évolutionnaires appliqués à la reconnaissance des formes et à la conception optique. *Université Laval*, 2005.

[30] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st édition, 1989. ISBN 0201157675.

[31] Joël Goossens. Techniques avancées de systèmes d'exploitation. *Faculté des Sciences, Département Informatique, Université Libre de Bruxelles*, 2004.

[32] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1):329–365, 1995.

[33] John H Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[34] John H Holland. Genetic algorithms. *Scholarpedia*, 7(12):1482, 2012.

[35] John Henry Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[36] Nicholas R Jennings et Michael Wooldridge. Applications of intelligent agents. Dans *Agent technology*, pages 3–28. Springer, 1998.

[37] Nick R Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *International Journal of Intelligent and Cooperative Information Systems*, 2(03):289–318, 1993.

[38] Jonathan A Jones et Dieter Jaksch. *Quantum Information, Computation and Communication*. Cambridge University Press, 2012.

[39] J. Kennedy et R. Eberhart. Particle swarm optimization. Dans *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[40] Fatima-Zohra Kettaf et Jean-Pierre Asselin de Beauville. Des algorithmes évolutionnaires pour la classification automatique. *La Revue de Modulad*, (25):23–45, 2000.

[41] Eisuke Kita. *Evolutionary algorithms*. InTech, 2011.

[42] Zakaria LABOUDI et Salim CHIKHI. Evolution d'automates cellulaires par algorithmes génétiques quantiques sur un environnement parallèle. URL `http://umc.edu.dz/buc/theses/informatique/LAB5344.pdf`.

[43] Michel Lannoo. Allocution du président entrant. *Reflets de la physique*, (28):2–2, 2012.

[44] Abdesslem Layeb et Djamel-Eddine Saidouni. Quantum genetic algorithm for binary decision diagram ordering problem. *International Journal of Computer Science and Network Security*, 7(9):130–135, 2007.

[45] Yajun Li, Yuhang Yang, Maode Ma et Rongbo Zhu. A problem-specific genetic algorithm for multiprocessor real-time task scheduling. Dans *Innovative Computing Information and Control, 2008. ICICIC'08. 3rd International Conference on*, pages 186–186. IEEE, 2008.

[46] Evelyne Lutton. Darwinisme artificiel : une vue d'ensemble. *Traitement du Signal, numéro spécial" Gestion intelligente de senseurs*, 2004.

[47] Rajib Mall. *Real-time systems : theory and practice*. Pearson Education India, 2009.

[48] Thomas W Malone. Modeling coordination in organizations and markets. *Management science*, 33(10):1317–1332, 1987.

[49] Samir Mekid. Further structural intelligence for sensors cluster technology in manufacturing. *Sensors*, 6(6):557–577, 2006.

[50] Mitchell Melanie. An introduction to genetic algorithms. *Cambridge, Massachusetts London, England, Fifth printing*, 3, 1999.

[51] Kamal E Melkemi, Mohamed Batouche et Sebti Foufou. A multiagent system approach for image segmentation using genetic algorithms and extremal optimization heuristics. *Pattern Recognition Letters*, 27(11):1230–1238, 2006.

[52] Kamal Eddine Melkemi et Sebti Foufou. Fuzzy distributed genetic approaches for image segmentation. *Journal of Computing and Information Technology - CIT*, pages 221—-231, 2010.

[53] K. M. Mezghiche, K. E. Melkemi et S. Foufou. Matching with quantum genetic algorithm and shape contexts. Dans *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on*, pages 536–542, Nov 2014.

[54] Mohamed Khalil MEZGHICHE. *Approche quantique pour l'appariement de formes*. Thèse de doctorat, Université Mohamed Khider-Biskra, 2015.

[55] Arezou Mohammadi et Selim G Akl. Scheduling algorithms for real-time systems. *School of Computing, Queens University*, 2005.

[56] Fadia Nemer. *Optimisation de l'estimation du WCET par analyse inter-tâche du cache d'intructions*. Thèse de doctorat, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2008.

[57] Kevin M Passino. Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems, IEEE*, 22(3):52–67, 2002.

[58] B Hanumantha Rao et S Sivanagaraju. Optimum allocation and sizing of distributed generations based on clonal selection algorithm for loss reduction and technical benefit of energy savings. Dans *Advances in Power Conversion and Energy Technologies (APCET), 2012 International Conference on*, pages 1–5. IEEE, 2012.

[59] KV Shibu. *Introduction to Embedded Systems*. Tata McGraw-Hill Education, 2009.

[60] JA Stancovic. Misconceptions about real-time computing. *IEEE computer*, 21: 10–19, 1988.

[61] John A Stankovic. *Real-time computing systems : The next generation*. Department of Computer and Information Science, University of Massachusetts Amherst, 1988.

[62] John A Stankovic, Marco Spuri, Krithi Ramamritham et Giorgio C Buttazzo. *Deadline scheduling for real-time systems : EDF and related algorithms*, volume 460. Springer Science & Business Media, 2012.

[63] Noria Taghezout. *Conception et développement d'un système multi-agent d'aide à la décision pour la gestion de production dynamique*. Thèse de doctorat, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2011.

[64] El-Ghazali Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009.

[65] WJ Tang, QH Wu et JR Saunders. Bacterial foraging algorithm for dynamic environments. Dans *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1324–1330. IEEE, 2006.

[66] Samir Touaf. *DIAGNOSTIC LOGIQUE DES SYSTEMES COMPLEXES ET DYNAMIQUES DANS UN CONTEXTE MULTI-AGENT*. Thèse de doctorat, Université Joseph-Fourier-Grenoble I, 2005.

[67] E Uma et R Shanmughasundaram. Simulation of scheduling test for rhythmic task model with a case study. Dans *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*, pages 1–4. IEEE, 2014.

[68] Yanfei Zhong, Liangpei Zhang, Jianya Gong et Pingxiang Li. A supervised artificial immune classifier for remote-sensing imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 45(12):3957–3966, 2007.