

Ministère de l'Enseignement Supérieur, de la Recherche Scientifique
Université de Monastir

*_*_*_*_*_*_*_*

Institut Supérieur d'Informatique et de Mathématiques de Monastir



Projet de Fin d'Etudes

En vue de l'obtention du
Diplôme de Licence Fondamentale en Informatique

Optimisation Distribuée par Essaimes des Particules : Cas de MaxCSP

Réalisé par

Soufièn Jabeur & Manel Mili

Encadré par

Mme. Zemni Bechair

Année universitaire 2014-2015

Dédicaces

À mon cher père qui a toujours su me soutenir, me conseiller, m'assister et m'indiquer le bon chemin ... L'amour qu'il me voue est irremplaçable... ses sacrifices pour mon éducation et mes études sont énormes. Je lui dois beaucoup, et je lui suis plus que reconnaissante.

À mon chère mère, toujours serviable et dévouée, tout mon amour sans limites, pour sa douceur, sa tendresse et toute l'affection qu'elle m'a donnée tout au long de ma vie... Je lui dois beaucoup, et je lui suis plus que reconnaissante, elle demeurera pour moi, la source d'amour infini... C'est à mes parents qui n'ont cessé de me soutenir et m'encourager que je dédie ce travail... en leur exprimant ma gratitude, leur promettant de demeurer toujours à la hauteur des espoirs qu'ils placent en moi et de toujours honorer la famille...

À mes chers frères et mes chères sœurs.

Puisse Dieu, le tout Puissant, les garder en bonne santé.

À tous mes amis pour les bons souvenirs et le beau temps que nous avons passé ensemble, en leur souhaitant le succès et le bonheur dans leur vie.

Toutes les personnes qui j'aime et qui m'aiment.

Je vous remercie tous et je vous dédie ce travail, résultat de plusieurs années d'étude.

...  Soufièn

Dédicaces

Que ce travail témoigne de mes respects :

A mes parents : Grâce à leurs tendres encouragements et leurs grands sacrifices, ils ont pu créer le climat affectueux et propice à la poursuite de mes études. Aucune dédicace ne pourrait exprimer mon respect, ma considération et mes profonds sentiments envers eux. Je prie le bon Dieu de les bénir, de veiller sur eux, en espérant qu'ils seront toujours fiers de moi.

A mes frères : Ils vont trouver ici l'expression de mes sentiments de respect et de reconnaissance pour le soutien qu'ils n'ont cessé de me porter.

A tous mes professeurs : Leur générosité et leur soutien m'oblige de leurs témoigner mon profond respect et ma loyale considération.

A tous mes amis et mes collègues : Ils vont trouver ici le témoignage d'une fidélité et d'une amitié infinie.

...✍️ Manel

Remerciements

Nous remercierons avant tout ALLAH le tout puissant qui m'a donné les capacités physiques et intellectuelles nécessaires à la réalisation de ce projet de fin d'études.

Nous tenons à exprimer nos sincères remerciements aux membres du jury, qui nous ont honorés en acceptant de juger ce modeste travail. Veuillez accepter notre gratitude en guise de reconnaissance pour cette généreuse attention.

Nous remercions plus particulièrement Mme Zemni Bechair pour avoir accepté de nous encadrer pour ce projet de fin d'études et pour la confiance qu'elle a placée en nous. Nous avons une considération respectueuse que nous lui exprimons notre plus vive reconnaissance et notre plus grande estime.

A tous nous enseignants, il nous est à la fois un plaisir et un devoir de vous remercier. Nous vous sommes très reconnaissants pour tout le savoir. Veuillez trouver ici le témoignage de nos plus vifs remerciements.

Un grand merci à tous ceux qui ont, de près ou de loin, aidé à la réalisation de ce projet.

...✍ Soufièn et Manel

Résumé

Dans le cadre du raisonnement par contraintes, nous introduisons une approche par essaim de particules(**OEP**). Cette dernière est une nouvelle approche qui traite des problèmes complexes tels que les problèmes de satisfaction de contraintes (**CSPs**) et plus précisément **Max-CSPs**. Les algorithmes d'optimisation tels que l'algorithme **OEP** sont des méthodes efficaces pour résoudre ces problèmes.

Même si ces métaheuristiques permettent de réduire de manière significative le temps de calcul de l'exploration de l'espace de recherche d'une solution, cette exploration reste complexe lorsque de très grandes instances d'un problème sont à résoudre. Ainsi, l'utilisation du Système Multi-agents (**SMA**) est requise pour accélérer la recherche.

En effet, dans ce projet de fin d'études, nous nous concentrerons ainsi sur l'implémentation de l'algorithme distribué d'optimisation par essais particuliers pour les problèmes **Max-CSPs** sous une architecture multi-agents. Les résultats expérimentaux obtenus démontrent l'efficacité de l'approche proposée et sa capacité d'exploiter l'architecture du **SMA**.

Mots clés

Problème de Satisfaction de Contraintes, algorithme des essais particuliers, Max-CSPs,
Système Multi-Agents(SMA)

Abstract

As part of Constraint programming, we introduce a particle swarm optimization (**PSO**) approach which is a new computational method, it solves complex problems such as Constraint satisfaction problems (**CSPs**) and specifically **Max-CSPs**. The optimization algorithms such as the **PSO** algorithm are effective methods to solve these problems.

These metaheuristics allow to significantly reducing the calculation time of exploring the search space of a solution, however, it still excessively costing when very large instances of the problem are solved. Therefore, the use of Multi-Agent System (**MAS**) is required as an additional way to accelerate the research.

Indeed, in this graduation project, we will focus on the implementation of the distributed algorithm for **Max-CSP** problems multi-agent architectures The experimental results show the effectiveness of the proposed approach and its ability to exploit the architecture **MAS**.

Keywords

Constraint Satisfaction Problems, PSO algorithm, Max-CSPs, Multi-Agent System(MAS)

Table des matières

| | |
|--|----------|
| Introduction générale | 1 |
| 1 Etat de l'art | 3 |
| 1.1 Problèmes de satisfaction des contraintes | 4 |
| 1.1.1 Présentation | 4 |
| 1.1.2 Définitions | 4 |
| 1.1.3 Extension de CSPs | 7 |
| 1.1.4 Méthodes de résolutions de Max-CSPs | 8 |
| 1.1.4.1 Méthodes complètes | 8 |
| 1.1.4.2 Méthodes incomplètes | 10 |
| 1.2 Essaims particuliers | 17 |
| 1.2.1 Présentation | 17 |
| 1.2.2 Description générale | 18 |
| 1.2.3 Technique PSO | 19 |
| 1.2.3.1 Formalisme | 19 |
| 1.2.3.2 Coefficients du PSO | 20 |
| 1.2.3.3 Algorithme PSO | 20 |
| 1.2.3.4 Principales caractéristiques | 22 |
| 1.3 Les systèmes multi-agents | 22 |
| 1.3.1 Présentation | 22 |
| 1.3.2 La notion d'agent | 22 |
| 1.3.2.1 Définitions | 23 |
| 1.3.2.2 Caractéristiques d'un agent | 24 |
| 1.3.2.3 Classification des agents | 24 |
| 1.3.3 Système multi-agents | 27 |
| 1.3.3.1 Définition | 27 |
| 1.3.3.2 Composition du système multi-agents | 27 |
| 1.3.3.3 Caractéristiques d'un système multi-agents | 27 |
| 1.3.4 Interaction dans un système Multi-Agents | 28 |
| 1.3.4.1 La coopération | 28 |
| 1.3.4.2 La négociation | 29 |
| 1.3.4.3 La coordination | 29 |
| 1.3.5 Communication dans les systèmes Multi-Agents | 29 |

| | | |
|----------|---|-----------|
| 1.4 | Conclusion | 31 |
| 2 | Algorithme d'optimisation par Essaim Particulaires Distribué pour les problèmes MaxCSP | 32 |
| 2.1 | Présentation de l'approche | 33 |
| 2.1.1 | Générateur aléatoire de CSP | 33 |
| 2.1.2 | Correspondances entre CSP et PSO | 34 |
| 2.1.3 | Concept de Template | 35 |
| 2.1.4 | Dynamique globale | 36 |
| 2.1.5 | Structure des agents | 38 |
| 2.1.5.1 | L'agent interface | 38 |
| 2.1.5.2 | L'agent espèce | 39 |
| 2.2 | Conception | 39 |
| 2.2.1 | Architecture globale | 39 |
| 2.2.2 | Langage de Modélisation | 40 |
| 2.2.3 | Diagramme de cas d'utilisation | 40 |
| 2.2.4 | Diagrammes de séquences | 42 |
| 2.2.5 | Diagramme de classe | 51 |
| 2.3 | Conclusion | 54 |
| 3 | Réalisation et expérimentations | 55 |
| 3.1 | Environnement de travail | 56 |
| 3.1.1 | Environnement matériel | 56 |
| 3.1.2 | Environnement logiciel | 56 |
| 3.1.2.1 | Plate-forme multi-agents | 56 |
| 3.1.2.2 | Choix de la plate-forme multi-agents utilisée | 57 |
| 3.1.2.3 | Plateforme Madkit | 58 |
| 3.2 | Expérimentation et illustration | 60 |
| 3.2.1 | Description de processus d'expérimentation | 60 |
| 3.2.1.1 | Démarrage de l'application | 60 |
| 3.2.1.2 | Affichage du résultat | 61 |
| 3.2.2 | Mode expérimentale | 64 |
| 3.3 | Chronogramme | 70 |
| 3.4 | Conclusion | 71 |
| | Conclusion générale et perspectives | 72 |
| | Bibliographie | 73 |
| | Annexes | 75 |

Table des figures

| | | |
|------|---|----|
| 1.1 | Variables du problème "Coloriage de graphe" | 5 |
| 1.2 | Contraintes et relations du problème "Coloriage de graphe" | 5 |
| 1.3 | Graphe de contrainte du problème "Coloriage de graphe" | 6 |
| 1.4 | Algorithme de Metropolis | 11 |
| 1.5 | Algorithme de recuit simulé | 12 |
| 1.6 | Algorithme de recherche tabou | 13 |
| 1.7 | Comportement naturel des fourmis | 14 |
| 1.8 | Algorithme de colonie de fourmis pour un problème d'affectation | 15 |
| 1.9 | Dans l'ordre : reproduction, mutation d'addition, mutation de soustraction et mutation de modification | 16 |
| 1.10 | Diagramme de fonctionnement de l'algorithme génétique | 16 |
| 1.11 | Algorithme évalutionnaire générique | 17 |
| 1.12 | Choix des informatrices parmi les sept particules | 18 |
| 1.13 | Principe de déplacement d'une particule | 19 |
| 1.14 | Les étapes de l'algorithme de PSO | 21 |
| 1.15 | Algorithme d'optimisation par essaim particulaire(basique) | 21 |
| 1.16 | Architecture générale d'un agent réactif | 25 |
| 1.17 | Architecture générale d'un agent cognitif | 25 |
| 1.18 | Architecture horizontale d'agent hybride | 26 |
| 1.19 | Architecture verticale d'agent hybride | 26 |
| 1.20 | Architecture globale d'un système multi-agents | 28 |
| 1.21 | Communication via un Tableau Noir | 30 |
| 1.22 | Communication par envoi de messages | 30 |
| 2.1 | Algorithme de générateur aléatoire de CSP | 34 |
| 2.2 | La position x_i d'une particule | 35 |
| 2.3 | Le changement de la position d'une particule à travers la vitesse V_i | 35 |
| 2.4 | Exemple sur la notion de Template | 36 |
| 2.5 | Diagramme des étapes de dynamique globale | 37 |
| 2.6 | Algorithme de comportement d'un agent espèce | 38 |
| 2.7 | Architecture globale de l'application | 39 |
| 2.8 | Diagramme de cas d'utilisation générale de l'application | 41 |
| 2.9 | Diagramme de séquence de Communication | 43 |
| 2.10 | Diagramme de séquence de génération aléatoire de CSP | 44 |

| | | |
|------|--|----|
| 2.11 | Diagramme de séquence de création d'un essaim initial | 45 |
| 2.12 | Diagramme de séquence de calcul de la spécificité | 46 |
| 2.13 | Diagramme de séquence de partitionnement de l'essaim initial | 47 |
| 2.14 | Diagramme de séquence de création des agents espèces | 48 |
| 2.15 | Diagramme de séquence de l'application de PSO | 49 |
| 2.16 | Diagramme d'évaluation de fitness | 50 |
| 2.17 | Diagramme de séquence de consultation de fichier CSP | 51 |
| 2.18 | Diagramme de classes | 52 |
| | | |
| 3.1 | Evaluation comparative de plate-formes multi-agents | 58 |
| 3.2 | Modèle de Madkit | 59 |
| 3.3 | Architecture générale de Madkit | 60 |
| 3.4 | Interface de l'application | 61 |
| 3.5 | Exemple d'un CSP généré | 61 |
| 3.6 | Exemple de génération d'une population initiale | 62 |
| 3.7 | Exemple de calcul de fitness du population initiale | 63 |
| 3.8 | Exemple de GBest | 64 |
| 3.9 | Présentation du temps écoulé en fonction de la dureté | 66 |
| 3.10 | Temps écoulé en fonction de la fitness(nombre des contraintes violées) | 67 |
| 3.11 | Variation des contraintes satisfaites en fonction du dureté | 68 |
| 3.12 | Comparaison de DPSO et CPSO : Ratio-CPU | 69 |
| 3.13 | Compraison de DPSO et CPSO : Ratio-Satisfaction | 70 |
| 3.14 | Chronogramme du projet | 70 |

Liste des tableaux

- 3.1 Variation de temps de CPU en fonction du dureté 65
- 3.2 Variation de temps de CPU en fonction des contraintes violées 66
- 3.3 Variation des contraintes satisfaites en fonction du dureté 67
- 3.4 Ratio de Temps-CPU 69
- 3.5 Comparaison de DPSO et CPSO : Ratio-Satisfaction 69

Glossaire

AGR Agent Group Role

CSP Constraint Satisfaction Problem

CPSO Centralising Particle Swarm Optimization

CPU Central Processing Unit

DPSO Distributed Particle Swarm Optimization

FFR Fitness Function Range

FV Fitness Value

MAS Multi-Agent Systems

Max-CSP Maximal Constraint Satisfaction Problem

OEP Optimisation par Essaims Particulaires

\mathbf{P}_{best} Particule global best

\mathbf{P}_{lbest} Particule local best

PSO Particle Swarm Optimization

UML Unified Modeling language

Introduction générale

Les problèmes de satisfaction de contraintes ou CSP (Constraint Satisfaction Problem) sont des problèmes mathématiques où l'on cherche des états ou des objets satisfaisant un certain nombre de contraintes ou de critères. Les CSPs occupent une place très importante dans diverses disciplines de recherche opérationnelle (RO) et d'intelligence artificielle (IA).

De nombreux CSPs nécessitent la combinaison d'heuristiques et de méthodes d'optimisation combinatoire pour être résolus en un temps raisonnable. Ils sont notamment au cœur de la programmation par contraintes, un domaine fournissant des langages de modélisation de problèmes et des outils informatiques les résolvant. Leur importance se justifie, par le fait que plusieurs problèmes que nous rencontrons peuvent être facilement formalisés par le formalisme CSP (Constraint Satisfaction Problem). En titre d'exemples, on peut citer de nombreux problèmes qui sont définis en termes de contraintes (de temps, d'espace, ..., ou plus généralement de ressources) :

- les problèmes de planification et ordonnancement : planifier une production, gérer un trafic ferroviaire, ...
- les problèmes d'affectation de ressources : établir un emploi du temps, allouer de l'espace mémoire, du temps CPU par un système d'exploitation, affecté du personnel à des tâches, des entrepôts à des marchandises.
- les problèmes d'optimisation : optimiser des placements financiers, des découpes de bois, des routages de réseaux de télécommunication, ...
- etc, ...

Ces différents problèmes sont désignés par le terme générique CSP, et ont la particularité commune d'être fortement combinatoires : il faut envisager un grand nombre de combinaisons avant de trouver une solution. La programmation par contraintes est un ensemble de méthodes et algorithmes qui tentent de résoudre ces problèmes de la façon la plus efficace possible.

Toutefois, un grand nombre de méthodes de résolution ont été développées. Elles sont classées en deux catégories :

La première catégorie est les méthodes complètes, qui permettent de garantir la solution ou d'informer sur son inexistence par l'exploration totale de l'espace de recherche et de répondre au problème de décision et donc de prouver la satisfiabilité d'un problème, ou son insatisfiabilité dans le cas échéant. C'est pourquoi ces méthodes sont contrecarrées par l'explosion combinatoire en extrairont l'ensemble des solutions d'un problème proposé.

La deuxième catégorie est celles des méthodes incomplètes, qui sont des méthodes qui ne donnent aucune garantie sur l'existence d'une solution mais elles donnent des solutions approchées de l'optimum global. Parmi ces méthodes, on peut citer les algorithmes évolutionnaires [1]; ces méthodes sont basées sur une analogie avec le phénomène de l'évolution et de la sélection naturelle. Actuellement, les approches évolutives constituent des algorithmes robustes qui sont à la base d'un axe en plein essor de l'informatique. Cependant, les algorithmes évolutifs ont une principale limite qui réside dans le fait qu'ils sont gourmands en temps de calcul. En effet, ils ne permettent pas parfois d'atteindre une solution dans un temps raisonnable.

L'Optimisation par les Essaims Particulaires (OEP) constitue l'exemple le plus récent des algorithmes évolutifs. Cette approche est issue d'une analogie avec les comportements collectifs des animaux. A travers la communication, les différentes particules ajustent leur comportement pour atteindre le but commun. En effet, chez certains groupes d'animaux, comme les oiseaux, les abeilles, les poissons, nous observons souvent des mouvements complexes. Alors que les individus eux-mêmes n'ont accès qu'à des informations limitées, comme la vitesse et la position de leurs voisins.

Les problèmes de satisfaction de contraintes sont complexes et NP-difficiles [2], leur modélisation continue d'évaluer en termes de contraintes et d'objectifs et leur résolution est coûteuse en temps de calcul CPU. Bien que des algorithmes presque optimaux tels que les métaheuristiques permettent de réduire la complexité de leur résolution, elles restent insuffisantes pour résoudre des problèmes de grande taille. De nombreux travaux de recherche ont été conçus afin de réduire le temps de calcul et ce par l'utilisation de parallélisation des méthodes ainsi, ces méthodes deviennent plus adaptées et permettent d'obtenir une convergence plus rapide vers des solutions de bonne qualité. Aussi, l'approche multi-agents a eu de très grands apports dans la résolution des problèmes fortement combinatoires [3] [4]. C'est pour cela que nous avons jugé utile de distribuer l'algorithme d'optimisation par les essaims particuliers via les systèmes multi-agents. Le modèle aboutit ainsi est composé des agents créés dynamiquement. Ces derniers coopèrent pour résoudre le problème. Chaque agent exécute son propre OEP guidé par une nouvelle structure de données appelée template, sur un sous-essaim qui lui est propre. Chacune des sous-essaims est composée des particules qui ont les mêmes fitness.

Le présent projet consiste en premier lieu à implémenter la nouvelle extension, distribué par les systèmes multi-agents, de l'algorithme PSO pour le cas de Max-CSPs. Puis d'expérimenter la performance de l'algorithme sur des instances des CSPs générées aléatoirement.

Nous verrons dans le premier chapitre l'état de l'art relatif aux problèmes de satisfaction de contrainte, les extensions de CSPs, leurs méthodes de résolutions en particulier les algorithmes des essaims particuliers, puis nous introduisons les systèmes multi-agents. Le deuxième chapitre, décrit la nouvelle approche ainsi qu'une conception détaillée du modèle proposé. Dans le troisième chapitre, nous détaillons la plateforme de développement, ainsi que les résultats expérimentaux du projet. La conclusion récapitule les résultats de notre travaux et propose des perspectives sur les travaux effectués.

Chapitre 1

Etat de l'art

Introduction

Ce chapitre est consacré en première partie à une présentation générale des problèmes de satisfaction de contrainte, ainsi que leurs extensions et leurs méthodes de résolution connues dans la littérature. Ces méthodes sont réparties en méthodes exactes, permettant d'obtenir des solutions optimales et des métaheuristiques, permettant d'obtenir des solutions approchées. Puis, nous présentons une revue sur l'optimisation par essaims particuliers ainsi que quelques approches connues dans la littérature. Enfin, nous présentons les systèmes multi-agents.

1.1 Problèmes de satisfaction des contraintes

1.1.1 Présentation

Les problèmes de satisfaction des contraintes introduits par Montanari en 1974, occupent une place importante en intelligence artificielle. En effet, un problème de satisfaction de contraintes (CSP) est généralement présenté sous la forme d'un ensemble de variables, auxquelles sont associées des domaines, ainsi qu'un ensemble de contraintes. Chaque contrainte est définie sur un sous-ensemble de l'ensemble des variables et limite les combinaisons de valeurs que peuvent prendre ces variables.

La résolution d'un CSP consiste à trouver une affectation de valeur pour chaque variable de telle sorte que l'ensemble des contraintes soit satisfait. Pour certains problèmes, le but est de trouver toutes ces affectations. Plus formellement, nous définissons dans les sections suivantes les concepts utilisés par les CSP.

1.1.2 Définitions

Définition 1

Un problème de satisfaction de contrainte est la donnée d'un quadruplet (X, D, C, R) avec :

- $X = x_1, x_2, \dots, x_n$ est l'ensemble des n variables x_i .
- $D = D_1, D_2, \dots, D_n$ est l'ensemble des n domaines D_i ou $D(x_i)$ qui peuvent être soit :
 - discrets et finis où chaque D_i représente les q valeurs possibles de la variable x_i :
 $D = D_1, D_2, \dots, D_q$.
 - continus où chaque D_i représente l'intervalle des valeurs possibles de la variable x_i : $D_i = [D_1, D_q]$.
- $C = C_1, C_2, \dots, C_m$ est l'ensemble des m contraintes C_i où chaque C_i porte sur tous les éléments d'un sous-ensemble $Var(C_i)$ de p variables : $C_i = x_{i1}, x_{i2}, \dots, x_{ip}$.

- $R = R_1, R_2, \dots, R_m$ est l'ensemble des relations où chaque R_i ou $R(C_i)$ est associée à une C_i pour définir l'ensemble des combinaisons de valeurs satisfaisant C_i . R_i est définie par un sous-ensemble du produit cartésien $D_{i1} * D_{i2} * \dots * D_{in}$.

* **Exemple :**

Nous représentons dans cette section un exemple de problèmes de satisfaction de contraintes, ceci est présente par les deux figures 1.1 et 1.2, qui est le problème de coloriage de graphe (on traitera le cas d'un graphe avec 3 nœuds). Ce problème consiste à affecter une couleur à chaque sommet d'un graphe de sorte que deux sommets reliés par une arête doivent être de couleurs différentes. Le problème $P_{3-nœuds}$ peut être défini comme suit :

$P_{3-nœuds} = (X, D, C, R)$ avec :

- $X = \{x_1, x_2, x_3\}$: représente les nœuds du graphe.
- $D = \{D_{x1}, D_{x2}, D_{xn}\}$ avec $D_{x1} = D_{x2} = D_{x3} = \{R, V, B\}$ (avec R désigne la couleur Rouge, V la couleur Vert, B désigne la couleur Bleue).
- $C = \{C_1, C_2, C_3\}$ où toutes les variables sont contraintes les unes par rapport aux autres avec $C_i =$ deux nœuds adjacents ne devant pas avoir la même couleur.
- $R = \{R_1, R_2, R_3\}$.

| Variables | Domaines |
|-----------|------------------------|
| x_1 | $D(x_1) = \{R, V, B\}$ |
| x_2 | $D(x_2) = \{R, V, B\}$ |
| x_3 | $D(x_3) = \{R, V, B\}$ |

FIGURE 1.1 – Variables du problème "Coloriage de graphe"

| Contraintes | Relations |
|------------------|---|
| $C_1 = x_1, x_2$ | $R(C_1) = \{(R, V)(R, B)(B, R)(B, V)(V, B)(V, R)\}$ |
| $C_2 = x_1, x_3$ | $R(C_2) = \{(R, V)(R, B)(B, R)(B, V)(V, B)(V, R)\}$ |
| $C_3 = x_2, x_3$ | $R(C_3) = \{(R, V)(R, B)(B, R)(B, V)(V, B)(V, R)\}$ |

FIGURE 1.2 – Contraintes et relations du problème "Coloriage de graphe"

Définition 2

- L'arité d'une contrainte C_i est le nombre de variables impliquées dans cette contrainte : c'est le cardinal de $Var(C_i)$ $Var(|C_i|)$.
- Le degré d'une variable est le nombre de contraintes dans lesquelles est impliqué une variable.
- Un CSP est dit binaire si toutes les contraintes ont une arité ≤ 2 et au moins une contrainte est d'arité $= 2$.

- Un CSP est dit n -aire si toutes les contraintes ont une arité $\leq n$ et au moins une contrainte a une arité $= n$.

Définition 3

On appelle dureté d'une contrainte, le rapport entre le nombre de ses tuples interdits et la taille du produit cartésien des domaines de ses variables.

Définition 4

Un tel problème peut être représenté par différents types de représentations :

- Une représentation par graphe global.
- Une représentation par extension : où chaque relation est représentée par un ensemble de t -uples qui respectent la contrainte associée. Dans le cadre de notre projet, nous avons travaillé avec de contraintes en extension.

* Exemple :

Dans l'exemple de colorage de graphe on peut définir la solution de ce problème sous une représentation par extension comme suit :

Solution possible $S_p : \{(R, V, B), (R, B, V), (V, B, R), (V, R, B), (B, V, R), (B, R, V)\}$.

- Une représentation par intension : où les relations peuvent être décrites par une équation, une inéquation, prédicat ou une fonction booléenne.

* Exemple : $x_1 + x_2 = 0$.

- Une représentation par graphe des contraintes : quand un CSP est constitué uniquement des contraintes binaires, on peut le représenter par un graphe non orienté, appelé graphe des contraintes où les nœuds désignent les variables et les arcs représentent les contraintes.

* Exemple :

On peut représenter le problème de colorage de graphe sous forme d'un graphe de contraintes comme illustre la figure 1.3 :

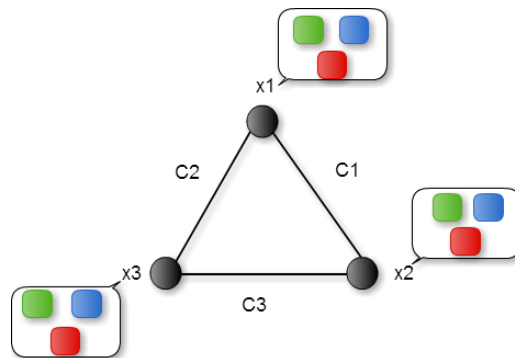


FIGURE 1.3 – Graphe de contrainte du problème "Coloriage de graphe"

Définition 5

Une solution d'une CSP : $P = (X, D, C, R)$ est une instanciation consistante complète. En d'autres termes, c'est une instanciation de toutes les variables de X sur D qui doit satisfaire toutes les contraintes de C .

Définition 6

Un CSP : $P = (X, D, C, R)$ est consistant si et seulement si $S_p \neq \phi$.

Définition 7

Soit $P = (X, D, C, R)$ et $P' = (X', D', C', R')$. On dit que P et P' sont équivalents ($P \equiv P'$) si et seulement si $S_p = S_{p'}$.

1.1.3 Extension de CSPs

Diverses extensions de CSP ont été mise en œuvre pour répondre à des besoins de formalisation des problèmes réels. Parmi les extensions, nous trouvons :

❖ CSOP (Constraint Satisfaction and Optimization Problem)

Le CSOP [5] est une CSP avec une fonction objective F à optimiser relative à une évaluation numérique de chaque solution. Le but dans ce cas n'est pas seulement une instanciation qui ne viole aucune des contraintes, mais aussi qui optimise la fonction F (maximiser ou minimiser). Plus formellement, un CSOP est défini par $\prec X, D, C, R, F \succ$:

- $\prec X, D, C, R \succ$ est un CSP ordinaire.
- F est la fonction objective à optimiser.

Pour la résolution des CSOP il existe plusieurs méthodes de recherche de solutions, nous citons par exemple la méthode Branch and Bound [6].

❖ PCSP (Partial Constraint Satisfaction Problem)

Le formalisme PCSP est une extension de CSP, qui a pour objectif de trouver une assignation qui satisfait un nombre maximal de contraintes. Un PCSP implique la recherche d'un sous-ensemble de variables qui satisfait un sous-ensemble des contraintes. Contrairement aux CSOPs qui exigent que toutes les contraintes soient satisfaites. Un PCSP consiste à trouver une solution qui minimise le nombre de contraintes violées. Les PCSP peuvent être vue comme un cas particulier des CSOPs où la fonction à optimiser est le nombre de contraintes violées [5].

❖ VCSP (Valued Constraint Satisfaction Problem)

Le VCSP est une extension des CSPs où une valuation exprime une probabilité d'existence ou d'interdiction est associée à chaque contrainte. Ces valuations expriment l'impact de la violation d'une contrainte ou du choix de l'affectation des variables sur la qualité de la solution [5].

❖ Max-CSP (Maximal Constraint Satisfaction Problem)

Ce sont des PCSPs ou chaque contrainte possède un coût [3]. Le but est de minimiser le nombre de coûts des contraintes violées (maximisation de la somme des coûts des contraintes satisfaites).

D'autres extensions existent comme les CSPs temporels qui s'intéressent à la résolution des problèmes de satisfaction de contrainte temporelle où les variables représentent le temps et les contraintes les relations temporelles entre elles. Le choix du formalisme de codification d'un problème d'affectation sous contrainte dépend de la spécificité de la problématique et du point de vue de concepteur.

1.1.4 Méthodes de résolutions de Max-CSPs

Il est facile de se rendre compte que malgré le nombre fini des solutions des problèmes d'optimisation combinatoire, la stratégie de résolution naïve, qui consiste à les énumérer toutes, en évaluant à chaque fois et en retenant la meilleure, est non concevable en pratique puisque le nombre de solutions croît généralement de manière exponentielle avec la taille du problème. Cette approche ne peut fonctionner que pour des problèmes de petite taille. A la frontière de la recherche opérationnelle et de l'intelligence artificielle, la recherche des méthodes de résolution a constituée l'objectif majeur de divers travaux. Ainsi, diverses méthodes de résolution des CSPs ont été proposées, elles peuvent être classées en deux grandes familles [7] :

- ❖ Méthodes complètes
- ❖ Méthodes incomplètes

1.1.4.1 Méthodes complètes

Une méthode de résolution est dite exacte, complète ou optimale si elle arrive à garantir l'obtention des solutions admissibles (qui satisfont l'ensemble des contraintes) et optimales au sens de la fonction objectif. Si les solutions n'existent pas, elle doit être capable de détecter l'échec. On dit que ces méthodes assurent la consistance et la complétude. Ainsi, ces méthodes ne produisent pas des résultats faux (inconsistants) et on peut dire qu'un algorithme qui cherche toutes les solutions d'un problème CSP est complet s'il produit effectivement toutes ses solutions. Les méthodes complètes reposent sur une recherche arborescente. Elles partent d'une instanciation vide où aucune variable n'est instanciée et construisent la solution si elle existe en parcourant l'ensemble des variables et en les instanciant une à une. Ce sont des méthodes qui opèrent selon une approche dite par construction.

a) La méthode séparation et évaluation (Branch and Bound)

L'algorithme de séparation et évaluation, plus connu sous son appellation anglaise Branch and Bound (B&B), repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des nœuds, et des feuilles. Le Branch-and-Bound est basé sur trois axes principaux [8] :

- **L'évaluation**

L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous-ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le Branch-and-Bound utilise une élimination de branches dans l'arborescence de recherche de la manière suivante :

la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque nœud parcouru à celui de la meilleure solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

- **La separation**

La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble de nœuds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

- **La stratégie de parcours**

- **La largeur d'abord**

Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées

- **La profondeur d'abord**

Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire

- **Le meilleur d'abord**

Cette stratégie consiste à explorer des sous problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.

b) **Generate and Test**

Cette méthode de résolution est parmi les premiers algorithmes conçus pour résoudre les problèmes de satisfaction de contrainte. Cette approche consiste à tester toutes les combinaisons possibles des valeurs admissibles (qui respectent toutes les contraintes). Donc, chaque combinaison sera générée et testée pour vérifier si elle satisfait toutes les

contraintes. La vérification de satisfaction des contraintes est donc testée après l'instanciation complète de toutes les variables du problème. Le nombre de combinaisons considérées par cette méthode est de taille du produit cartésien de tous les domaines des variables.

c) **Test and Generate (Backtrack)**

Le Test and Generate se base essentiellement sur une recherche en profondeur d'abord tout en réalisant des retours en arrière chronologiques [9]. Cet algorithme instancie les variables dans un ordre prédéterminé et procède par essais et erreurs : avant la phase du Backtrack, si une instanciation I_K est consistante, une variable supplémentaire X_{K+1} est alors instanciée. Dans la phase de Backtrack. Si aucune valeur du domaine de X_{K+1} ne peut prolonger I_K consistant, un retour est effectué sur la dernière variable X_K instancier et à laquelle sera affectée une nouvelle valeur de son domaine. L'algorithme s'arrête lorsqu'une solution est trouvée ou tout l'espace de recherche a été exploré. Contrairement à l'algorithme précédant Generate and Test, la vérification de satisfaction des contraintes se fait après l'instanciation partielle des variables. Quand une instanciation partielle viole une contrainte, l'algorithme élimine un sous-espace de produit cartésien des domaines de toutes les variables.

Il existe aussi le Backtrack intelligent qui est une stratégie basée sur l'identification de la vraie cause des échecs afin que l'algorithme puisse revenir sur la variable qui a causé l'échec et de prendre ainsi des décisions pertinentes. Cependant, cette stratégie nécessite un temps de calcul élevé. Des versions de backtrack intelligent développées pour la résolution des CSPs sont proposées dans [10].

d) **Backjumping**

Le principe de l'algorithme Backjumping est de sauter plusieurs niveaux en arrière jusqu'à la dernière variable liée par une contrainte avec la variable courante (retour à la dernière mauvaise décision). Si cette variable de retour ne possède pas d'autres valeurs, l'algorithme retourne plus loin à la variable liée par une contrainte avec la nouvelle variable de retour et ainsi de suite. Une version plus puissante de cet algorithme est appelée conflict-directed backjumping [11] ou graph based backjumping. D'autres méthodes existent comme Le Backchecking et le backmarking [12].

1.1.4.2 Méthodes incomplètes

Ces méthodes sont appelées aussi les méthodes approchées, elles sont utilisées généralement dans les problèmes de grande taille. Ces méthodes sont basées sur l'exploration progressive de l'espace afin de trouver une solution de qualité dans un temps raisonnable. Contrairement aux méthodes complètes, les méthodes incomplètes permettent de contrôler le temps puisque l'exécution peut être arrêtée à tout moment tout en obtenant une solution complète au problème. La qualité des résultats de ces algorithmes dépend généralement du temps d'exécution et la probabilité d'avoir une solution meilleure augmente au cours du temps (sans toutefois garantir une solution meilleure).

Des progrès ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, appelées méta-heuristique qui constituent des mécanismes très génériques qui peuvent être adaptés pour traiter plusieurs problèmes différents.

a) **Recuit simulé(simulated annealing)**

• **Présentation**

Le recuit simulé [13] est parmi les méthodes les plus anciennes et populaires. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus.

La méthode de recuit simulé repose sur l'algorithme Metropolis décrit par la figure 1.4 et s'inspire du processus de recuit physique utilisé en métallurgie pour améliorer la qualité d'un solide (la structure d'un solide refroidi dépend de la vitesse de refroidissement). Nous débutons avec une haute température. La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermo dynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

```
Initialiser un point de départ  $x_0$  et une température  $T$  ;  
pour  $i=1$  à  $n$  faire faire  
    tant que  $x_i$  n'est pas accepté faire  
        si  $f(x_i) \leq f(x_{i-1})$  alors  
            Accepter  $x_i$  ;  
        fin  
        si  $f(x_i) > f(x_{i-1})$  alors  
            Accepter  $x_i$  avec la probabilité  $e^{-\frac{f(x_i)-f(x_{i-1})}{T}}$  ;  
        fin  
    fin  
fin
```

FIGURE 1.4 – Algorithme de Metropolis

Le processus consiste à partir d'une température élevée T et à l'abaisser petit à petit, suivant un schéma d'abaissement de température. A chaque itération, une nouvelle configuration S' appartenant à $V(S)$ (le voisinage de S la configuration courante) est générée d'une manière aléatoire.

Si $\Delta = f(s') - f(s) \leq 0$ alors la nouvelle configuration est retenue. Sinon, elle est acceptée avec la probabilité de transition $e^{-\Delta/T}$. Par cette règle d'acceptation, le recuit simulé échappe aux optima locaux.

L'algorithme de recuit simulé est résumé par la figure 1.5 .

```
Données : mécanismes de perturbation d'une configuration
Résultat : une Configuration  $S$ 
Initialiser la température  $T$  ;
tant que La condition d'arrêt n'est pas atteinte faire
    tant que L'équilibre n'est pas atteint faire
        Tirer une nouvelle configuration  $S'$ 
        Appliquer l'algorithme de Metropolis ;
        si  $f(S') < f(S)$  alors
             $S_{min} = S'$  ;
             $f_{min} = f(S')$  ;
        fin
    fin
    Décroître la température
fin
```

FIGURE 1.5 – Algorithme de recuit simulé

- **Avantages et Inconvénients**

- **Avantages**

- ✓ La méthode du recuit simulé a l'avantage d'être souple vis-à-vis des évolutions du problème et facile à implémenter.
 - ✓ Contrairement aux méthodes de descente, SA évite le piège des optima locaux.
 - ✓ Excellents résultats pour un nombre de problèmes complexes.

- **Inconvénients**

Le principal inconvénient du recuit simulé est qu'une fois l'algorithme piégé à basse température dans un minimum local, il lui est impossible de s'en sortir. Plusieurs solutions ont été proposées pour tenter de résoudre ce problème, par exemple en acceptant une brusque remontée de la température de temps en temps, pour relancer la recherche sur d'autres régions plus éloignées [14].

Appart cela on peut citer quelques autres inconvénients comme :

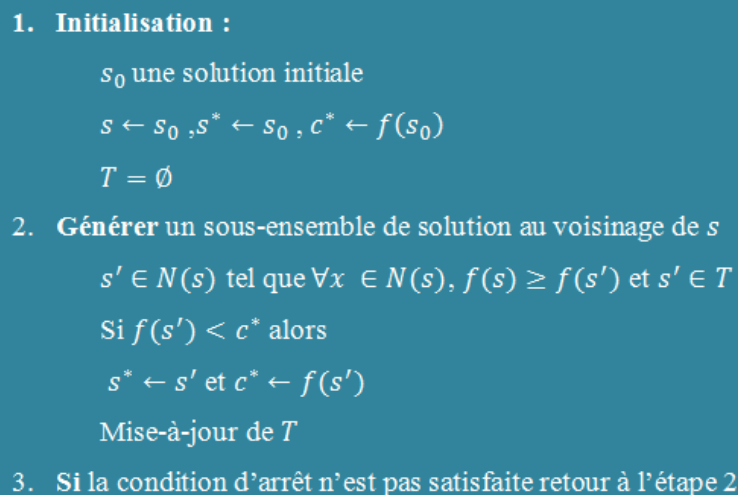
- ✗ La difficulté de déterminer la température initiale .
 - ✗ L'impossibilité de savoir si la solution trouvée est optimale .
 - ✗ Dégradation des performances pour les problèmes où il y a peu de minima locaux (comparé avec les heuristiques classiques comme la descente du gradient par exemple).

b) La recherche Tabou (Tabu Search)**• Présentation**

La recherche tabou (TS) est une méthode de recherche locale combine avec un ensemble de techniques permettant d'éviter d'être piégé dans un minimum local ou la répétition d'un cycle. La recherche tabou est introduite principalement par Glover [15], Hansen, Glover et Laguna dans [16]. Cette méthode a montré une grande efficacité pour la résolution des problèmes d'optimisation difficiles. En effet, à partir d'une solution initiale s dans un ensemble de solutions local S , des sous-ensembles de solution $N(S)$ appartenant au voisinage S sont générés. Par l'intermédiaire de la fonction d'évaluation nous retenons la solution qui améliore la valeur de f , choisie parmi l'ensemble de solutions voisines $N(S)$.

L'algorithme accepte parfois des solutions qui n'améliorent pas toujours la solution courante. Nous mettons en œuvre une liste tabou (tabu list) T de longueur K contenant les k dernières solutions visitées, ce qui ne donne pas la possibilité à une solution déjà trouvée d'être acceptée et stockée dans la liste tabou. Alors le choix de la prochaine solution est effectué sur un ensemble des solutions voisines en dehors des éléments de cette liste tabou. Quand le nombre K est atteint, chaque nouvelle solution sélectionnée remplace la plus ancienne dans la liste. La construction de la liste tabou est basée sur le principe FIFO, c'est-à-dire le premier entré est le premier sorti. Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de x^* , ou bien fixer un temps limite après lequel la recherche doit s'arrêter.

L'algorithme de recherche Tabou peut être résumé par la figure 1.6 :



```
graph TD
    1[1. Initialisation :  
s0 une solution initiale  
s ← s0, s* ← s0, c* ← f(s0)  
T = ∅] --> 2[2. Générer un sous-ensemble de solution au voisinage de s  
s' ∈ N(s) tel que ∀x ∈ N(s), f(s) ≥ f(s') et s' ∈ T  
Si f(s') < c* alors  
s* ← s' et c* ← f(s')  
Mise-à-jour de T]
    2 --> 3[3. Si la condition d'arrêt n'est pas satisfaite retour à l'étape 2]
```

1. Initialisation :
 s_0 une solution initiale
 $s \leftarrow s_0, s^* \leftarrow s_0, c^* \leftarrow f(s_0)$
 $T = \emptyset$

2. Générer un sous-ensemble de solution au voisinage de s
 $s' \in N(s)$ tel que $\forall x \in N(s), f(s) \geq f(s')$ et $s' \in T$
Si $f(s') < c^*$ alors
 $s^* \leftarrow s'$ et $c^* \leftarrow f(s')$
Mise-à-jour de T

3. Si la condition d'arrêt n'est pas satisfaite retour à l'étape 2

FIGURE 1.6 – Algorithme de recherche tabou

- **Avantages et Inconvénients**

- **Avantages**

- ✓ Offre des économies de temps de résolution pour des programmes de grosse taille.
 - ✓ Très bons résultats sur certains types de problèmes.
 - ✓ Algorithmes faciles à mettre en œuvre.

- **Inconvénients**

- ✗ Paramètres peu intuitifs.
 - ✗ Demande en ressources importantes si la liste des tabous est trop imposante.
 - ✗ Aucune démonstration de la convergence.

- c) **Colonies de fourmis (Ant Colony Optimization)**

- **Présentation**

L'algorithme d'optimisation par colonies de fourmis (OCF) a été proposé pour la première fois en 1992 par et appliqué au problème de planification de trajectoires en 1994 . Il est basé sur le comportement naturel des fourmis qui tracent un trajet entre leur nid et une source de nourriture afin d'accumuler des réserves. Tel qu'illustré à la figure 1.7, les fourmis voyagent de la fourmilière et la nourriture tout en sécrétant de la phéromone (schéma 1). Initialement, elles choisissent un chemin de façon aléatoire et retournent sur leurs propres pas suivant leur phéromone (schéma 2). La phéromone s'évapore, mais garde une concentration plus élevée sur le chemin le plus court puisqu'il est parcouru plus rapidement et donc plus souvent. Au lieu de retourner sur leurs pas, les fourmis qui avaient initialement choisi un chemin plus long optent maintenant pour le chemin plus court dû au plus haut niveau de phéromone. À la longue, la colonie de fourmis s'organise et emprunte le chemin optimal (schéma 3).

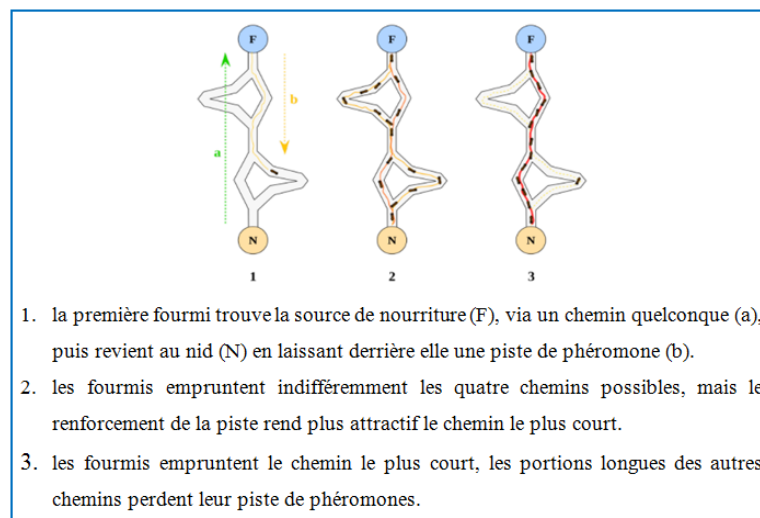


FIGURE 1.7 – Comportement naturel des fourmis

Ainsi la figure 1.8 présente un résumé sur l'algorithme de colonie de fourmis appliqué pour un problème d'affectation général.

```
Initialisation  $f(s^*)$  ; ncycles=0 ; meilleur_cycle=0
Tant que le critère d'arrêt n'est pas atteint faire
    ncycles = ncycles+1
    pour Chaque fourmis faire
        pour  $k = 1$  à  $n$  faire
            Choisir un objet  $i$  avec une probabilité  $obj$  ;
            Choisir une ressource admissible  $j$  pour l'objet  $i$  avec probabilité  $res$  ;
            Affecter la source  $j$  à l'objet  $i$  ;
        Fin
        Calculer le coût  $f(s_f)$  de la solution  $s_f = (x_1, \dots, x_n)$  ;
    Fin
     $s^* = \min\{f(s) | s \in \sigma = \{s_1, s_2, \dots, s_{n_{fourmis}}\}\}$ 
    Si  $f(s) < f(s^*)$  alors
         $s^* = s$  ; et meilleur_cycle= ncycles ;
        Evaporer les traces  $\tau$  ;
    Fin
Fin
```

FIGURE 1.8 – Algorithme de colonie de fourmis pour un problème d'affectation

- **Avantages et Inconvénients**

- **Avantages**

- ✓ Très grande adaptabilité .
 - ✓ Parfait pour les problèmes basés sur des graphes.

- **Inconvénients**

- ✗ Un état bloquant peut arriver.
 - ✗ Temps d'exécution parfois long.
 - ✗ Ne s'applique pas à tous type de problèmes.

d) **Algorithms génétiques (Genetic Algorithm)**

- **Présentation**

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils ont été adaptés à l'optimisation par John Holland, également les travaux de David Goldberg ont largement contribué à les enrichir [17].

Le vocabulaire utilisé est le même que celui de la théorie de l'évolution et de la génétique, on emploie le terme individu (solution potentielle), population (ensemble de solutions), génotype (une représentation de la solution), gène (une partie du génotype), parent, enfant, reproduction, croisement, mutation, génération, etc...

Leur fonctionnement est extrêmement simple, on part d'une population de solutions potentielles (chromosomes) initiales, arbitrairement choisies. On évalue leur performance (Fitness) relative. Sur la base de ces performances on crée une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : la sélection, le croisement et la mutation. Quelques individus se reproduisent, d'autres disparaissent et seuls les individus les mieux adaptés sont supposés survivre. On recommence ce cycle jusqu'à ce qu'on trouve une solution satisfaisante. En effet, l'héritage génétique à travers les générations permet à la population d'être adaptée et donc répondre au critère d'optimisation, quelques opérateurs génétiques se trouve à la figure 1.9 et le diagramme de l'algorithme génétique original à la figure 1.10 :

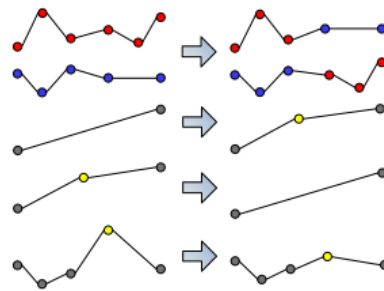


FIGURE 1.9 – Dans l'ordre : reproduction, mutation d'addition, mutation de soustraction et mutation de modification

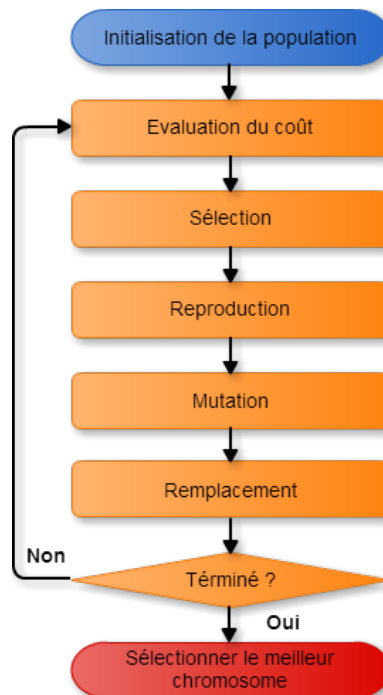


FIGURE 1.10 – Diagramme de fonctionnement de l'algorithme génétique

Ainsi l'algorithme génétique peut être résumé par la figure 1.11 :

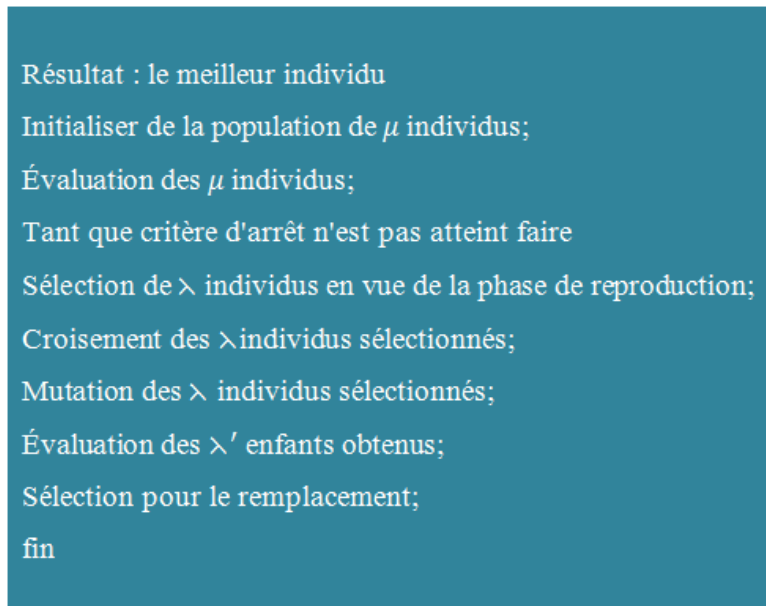


FIGURE 1.11 – Algorithme évolutionnaire générique

- **Avantages et Inconvénients**

- **Avantages**

- ✓ élimination de solutions non valides.
 - ✓ Permet de traiter des espaces de recherche importants (beaucoup de solutions, pas de parcours exhaustif envisagé).
 - ✓ Nombre de solutions important.
 - ✓ Relativité de la qualité de la solution selon le degré de précision demandé.

- **Inconvénients**

- ✗ Nécessitent plus de calculs que les autres algorithmes méta heuristiques (notamment la fonction évaluation).
 - ✗ Paramètres difficiles à fixer (taille population, mutation)
 - ✗ Choix de la fonction d'évaluation délicat.
 - ✗ Pas assuré que la solution trouvée est la meilleure, mais juste une approximation de la solution optimale.
 - ✗ Problèmes des optimaux locaux si paramètres mal évalués.

1.2 Essais particuliers

1.2.1 Présentation

L'optimisation par essais particuliers (OEP) ou (PSO (Particule Swarm Optimization) en anglais) [17] est une métaheuristique d'optimisation, inventée par Russel Eberhart (ingénieur en électricité) et James Kennedy (socio psychologue).

Cet algorithme s'inspire à l'origine du monde des vivants. Il s'appuie notamment sur un modèle développé par le biologiste Craig Reynolds à la fin des années 1980, permettant de simuler le déplacement d'un groupe d'oiseaux. Cette méthode d'optimisation se base sur la collaboration des individus entre eux. Elle a d'ailleurs des similarités avec les algorithmes des colonies de fourmis, qui s'appuient eux aussi sur le concept d'auto-organisation. Cette idée veut qu'un groupe d'individus peu intelligents puisse posséder une organisation globale complexe. Ainsi, grâce à des règles de déplacement très simples (dans l'espace des solutions), les particules peuvent converger progressivement vers un minimum global. Cette méta-heuristique semble cependant mieux fonctionner pour des espaces en variables continues. Au départ de l'algorithme, chaque particule est donc positionnée (aléatoirement ou non) dans l'espace de recherche du problème. Chaque itération fait bouger les particules en fonction de 3 composantes :

- ❖ L'ancienne vitesse de la particule.
- ❖ La meilleure position de la particule.
- ❖ La meilleure position des informatrices.

1.2.2 Description générale

Au départ de l'algorithme, les essaims sont repartis au hasard dans l'espace de recherche, chaque particule possède une vitesse et une position aléatoire. Ensuite à chaque pas du temps :

1. Chaque particule évalue la qualité de sa position ainsi elle peut mémoriser la meilleure position et la qualité de cette position qu'elle a trouvée jusqu'à cet instant.
2. Chaque particule a la capacité de communiquer avec ces congénères et d'échanger les meilleures solutions trouvées. Ces particules sont appelées les informatrices.
3. Chaque particule, à chaque instant, doit aussi choisir la meilleure parmi les meilleures performances qu'elle connaisse (soit ces propres performances soit les performances des informatrices), elle modifie sa vitesse en fonction de ce choix et se déplace par conséquent.

Pour le choix des informatrices, il y a plusieurs approches. Mais l'approche la plus utilisée c'est que les informatrices sont fixées dès le début comme il illustre la figure 1.12 :

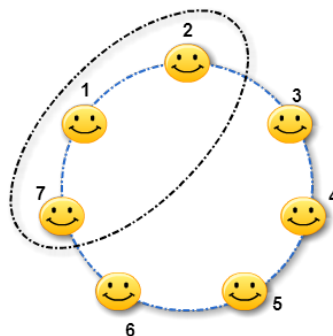


FIGURE 1.12 – Choix des informatrices parmi les sept particules

Les particules sont organisées dans la structure d'un anneau virtuelle et comme le nombre des informatrices est un paramètre à fixer au moment du lancement de l'algorithme les informatrices d'une particule seront les particules adjacentes (Exemple dans la figure 1.12 pour un nombre des informatrices de taille trois pour la particule numéro 1, les informatrices sont les particules numéros 2 et 7).

A chaque pas du temps la particule doit modifier sa vitesse ce qui est en fait une combinaison linéaire de trois tendances et à l'aide du coefficient de confiance.

- La tendance aventureuse : La particule continue à se déplacer avec la vitesse actuelle.
- La tendance conservatrice : La particule revienne sur ces pas (vers les meilleures positions déjà trouvées) pour cette tendance il peut y avoir un terme de hasard qui favorise l'exploration de l'espace de recherche.
- La tendance sociale : La particule suit l'informatrice qui a la meilleure position.

Pour réaliser son prochain mouvement, chaque particule combine trois tendances : suivre sa propre vitesse, revenir vers sa meilleure performance et aller vers la meilleure informatrice.

La figure 1.13 illustre la stratégie de déplacement d'une particule :

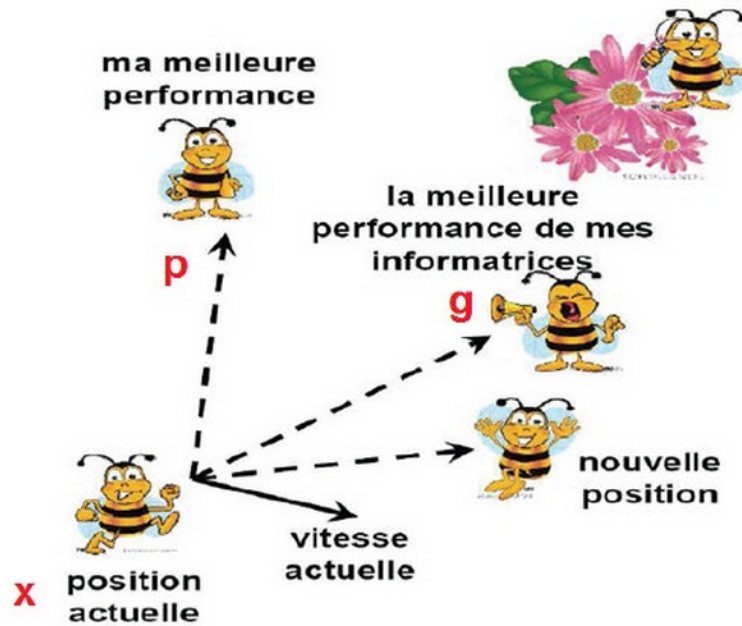


FIGURE 1.13 – Principe de déplacement d'une particule

1.2.3 Technique PSO

1.2.3.1 Formalisme

L'OEP peut être formalisée par, la dimension de l'espace de recherche D . La position courante d'une particule dans cet espace à l'instant (t) est donnée par le vecteur $X(t)$. La vitesse courante est $V(t)$. La meilleure position trouvée localement est notée par $P_{lbest}(t)$. La meilleure

position donnée par les informatrices est donnée par $P_{gbest}(t)$. Le n_{ieme} élément de vecteur $X(t)$ est noté $x_n(t)$.

Les équations de mouvement d'une particule pour chaque dimension d sont :

$$V(t) = w \times V(t-1) + r_1 c_1 \times (P_{lbest} - X(t-1)) + r_2 c_2 \times (P_{gbest} - X(t-1)) \quad (1)$$

$$X(t) = V(t) + X(t-1) \quad (2)$$

Lors de l'évolution de l'essaim, il peut arriver qu'une particule sorte de l'espace de recherche initialement défini. Plus généralement, on souhaite souvent rester dans un espace de recherche fini donné. Par conséquent, on ajoute un mécanisme pour éviter qu'une particule ne sorte de cet espace. Le plus fréquent est le confinement d'intervalle.

Supposons, par simplicité, que l'espace de recherche soit $[x_{min}, x_{max}]$. Alors, ce mécanisme stipule que si une coordonnée x_d , calculée selon les équations de mouvement, sort de l'intervalle $[x_{min}, x_{max}]$, on lui attribue en fait la valeur du point frontière le plus proche. De plus, on complète souvent le mécanisme de confinement par une modification de la vitesse.

1.2.3.2 Coefficients du PSO

Le coefficient w a été introduit pour limiter la vitesse des particules dans l'espace de recherche et il est appelé dans la bibliographie inertie massique ("*inertia weight*" en anglais). Ce coefficient a pour effet d'accélérer la convergence de la particule en contrôlant alternativement la capacité des particules à explorer et à exploiter l'espace de recherche.

Les coefficients c_1 et c_2 sont des constantes et représentent une accélération positive.

Les coefficients r_1 et r_2 sont des valeurs aléatoires positives compris entre $[0..1]$.

Les valeurs pour c_1 , c_2 et w correspondent à la mise en equation suivante :

$$\begin{cases} w &= 0.5 \\ c_1 &= 1.4 \\ c_2 &= 1.4 \end{cases}$$

1.2.3.3 Algorithme PSO

Tout d'abord l'algorithme commence par la génération d'une population initiale aléatoirement, puis il consiste à évaluer les fitness de chaque individu de la population. Ensuite chaque particule met à jour l'information de la meilleure valeur de la fitness et ses nouvelles valeurs de vitesse et position. Enfin l'algorithme affiche la meilleur solution si le critère d'arrête est atteint.

Le digramme sur la figure 1.14 représente les étapes de l'algorithme d'optimisation par essaim de particules :

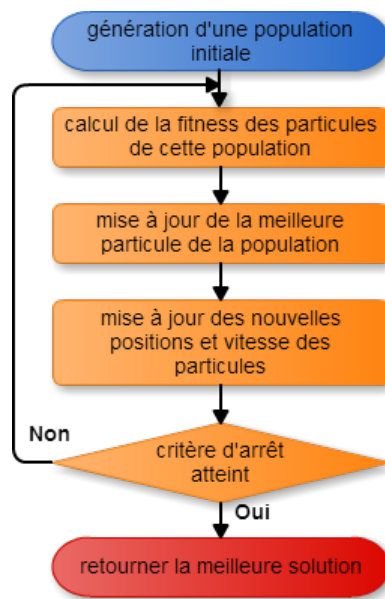


FIGURE 1.14 – Les étapes de l'algorithme de PSO

L'algorithme standard de PSO, dont les étapes sont décrites précédemment, est présenté par la figure 1.15 le suivant :

```

Initialisation aléatoire des positions de chaque particule;
Tant que le critère d'arrêt n'est pas atteint faire
  Pour chaque particule  $i$  faire
    Pour  $i = 1$  à  $N$  faire
      Déplacement de la particule selon les formules (1) et (2) ;
      Évaluation des positions;
      Si  $f(\vec{x}_i) < f(\vec{p}_i)$  alors
         $\vec{p}_i = \vec{x}_i$ ;
      Fin
      Si  $f(\vec{p}_i) < f(\vec{g})$  alors
         $\vec{g} = \vec{p}_i$ ;
      Fin
    Fin
  Fin
Fin
  
```

FIGURE 1.15 – Algorithme d'optimisation par essaim particulaire(basique)

1.2.3.4 Principales caractéristiques

L'algorithme PSO présente quelques propriétés intéressantes, qui le font un bon outil pour de nombreux problèmes d'optimisation, particulièrement les problèmes fortement non linéaires, continus ou mixtes (certaines variables étant réelles et d'autres entières) :

- Il est facile à programmer, quelques lignes de code suffisent dans n'importe quel langage évolué.
- Il est robuste (de mauvais choix de paramètres dégradent les performances, mais n'empêchent pas d'obtenir une solution).

Signalons de plus, qu'il existe des versions adaptatives qui évitent même à l'utilisateur la peine de définir les paramètres (taille de l'essaim, taille des groupes d'informatrices, coefficients de confiance). Par ailleurs, on notera que cette heuristique se démarque des autres méthodes évolutionnaires (typiquement les algorithmes génétiques) sur deux points essentiels : elle met l'accent sur la coopération plutôt que sur la compétition et il n'y a pas de sélection (au moins dans les versions de base), l'idée étant qu'une particule même actuellement médiocre mérite d'être conservée, car c'est peut-être justement elle qui permettra le succès futur, précisément du fait qu'elle sort des sentiers battus.

1.3 Les systèmes multi-agents

1.3.1 Présentation

La simulation est devenue un outil indispensable à la recherche pour explorer les systèmes sans avoir recours à l'expérience. En effet nous cherchons souvent à repousser les limites, c'est-à-dire à analyser des modèles plus grands et plus précis pour se rapprocher de la réalité d'un problème. De ce fait, la taille croissante des modèles a un impact direct sur la quantité de calcul et les ressources des systèmes centralisés ne sont pas ou plus suffisantes pour simuler ces modèles. L'utilisation de ressources parallèles permet de s'abstraire des limites de ressources des systèmes centralisés et ainsi d'augmenter la taille des modèles simulés.

Dans le monde de la simulation, la méthode de modélisation utilisée pour représenter « le système complexe », est un système composé d'un grand nombre d'entités hétérogènes, où les interactions entre les entités créent de multiples niveaux de structure et d'organisation qui empêchent un observateur de prévoir un comportement ou sa rétroaction, varie en fonction des caractéristiques de ce dernier. Les systèmes multi-agents sont ainsi souvent utilisés pour modéliser et simuler les systèmes complexes car ils reposent sur une description algorithmique d'agents qui interagissent et représentent ainsi bien le comportement attendu et proposent un cadre de réponse aux enjeux déjà décrites.

1.3.2 La notion d'agent

Durant ces dernières années, nous avons assisté à un fort et rapide développement des recherches sur les agents et les systèmes multi-agents. Le terme agent est un terme générique

qui se rapporte à différentes entités [18] :

- ❖ Des entités biologiques : les agents associés sont appelés agents biologiques
- ❖ Des robots autonomes
- ❖ Des logiciels informatiques et leurs composants qu'ils soient intégrés dans des systèmes d'exploitation ou des systèmes informatiques complexes

1.3.2.1 Définitions

Dans la littérature spécialisée, nous trouvons plusieurs définitions du concept d'Agent. Elles présentent certaines similitudes et dépendent du type d'application pour laquelle est conçu l'agent. Ce qui implique qu'il n'existe pas, actuellement, une définition de l'agent qui fasse foi dans le monde de l'intelligence artificielle distribuée. Il est donc nécessaire, pour avoir une bonne vision de ce concept, de confronter plusieurs de ces définitions. Nous allons présenter quatre d'entre elles.

- **Première définition due à Jacques Ferber [3] :**

Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement et la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents.

- **Deuxième définition due à John Bigham [19] :**

Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.

Les définitions exposées ci-dessus abordent une notion essentielle : l'autonomie. En effet, ce concept est au centre de la problématique des agents. L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains. Des autres notions importantes abordées par ces définitions concernent la capacité d'un agent à communiquer avec d'autres et d'être flexible. Maes définit les agents comme suit :

- **Troisième définition due à Maes [20] :**

Les agents autonomes sont des systèmes de calcul qui se trouvent dans un environnement complexe et dynamique, perçoivent et agissent d'une manière autonome dans cet environnement, et ce faisant réalisent les objectifs et les tâches pour lesquelles ils ont été conçus.

La définition exposée ci-dessus présente une propriété clés qu'est l'action. En effet l'action est un concept fondamental pour les SMA reposant sur le fait que les agents accomplissent des actions qui vont modifier leur environnement et donc leurs prises de décisions futures. Les interactions entre agents dépendent du mode de communication. Une communication entre deux agents est considérée comme une action pour celui qui la transmet et comme une perception pour celui qui la reçoit.

- **Quatrième définition due à Costa[21] :**

Un agent est une entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir, et d'interagir avec les autres agents.

Cette définition introduit l'interaction qui est le moteur des systèmes multi-agents. En effet, l'interaction suppose la présence d'agents capables de se rencontrer, de communiquer, de collaborer et d'agir.

1.3.2.2 Caractéristiques d'un agent

- **Autonomie** : L'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne ; en d'autre terme, les agents sont dits autonomes dans le sens où le créateur du système ne pilote pas leur comportement. C'est l'agent qui décide de ses actions par rapport à un éventail de possibilités données.
- **Réactivité** : un agent perçoit son environnement et répond aux changements qui s'y produisent en un temps raisonnable. L'environnement peut être le monde physique, un utilisateur via une interface graphique, d'autres agents, un système d'information (internet).
- **Communication** : un agent peut communiquer avec d'autres agents ainsi qu'avec des utilisateurs humains.
- **Aptitude sociale** : un agent peut interagir avec d'autres agents de façon coopérative ou compétitive pour atteindre ses objectifs.
- **Pro-activité** : l'agent est capable, sur sa propre initiative, de se fixer des buts pour atteindre ses objectifs (opportuniste).
- **Agent Intelligent** : un agent est « intelligent » s'il est capable de réaliser des actions flexibles et autonomes pour atteindre les objectifs qui lui ont été fixés. La flexibilité correspond aux propriétés de Réactivité, Pro-activité, Aptitudes sociales. Les systèmes multi-agents permettent d'observer des phénomènes ou des comportements au cours du temps par l'intermédiaire de simulations.

1.3.2.3 Classification des agents

Selon leurs modes de fonctionnement et leurs représentations de leurs environnements, les agents peuvent être classés en trois catégories essentielles qui sont : les agents réactifs, les agents cognitifs et les agents hybrides.

a) Les agents réactifs

Ce sont des agents qualifiés de non intelligents, ils répondent d'une façon opportune aux modifications de leurs environnements résultants des stimuli externes, les agents réactifs

agissent en fonction de ces dernières sans nécessiter de compréhension de leurs univers ni de leurs buts.

La figure 1.16 présente la structure générale d'un agent réactif dans système multi-agents :

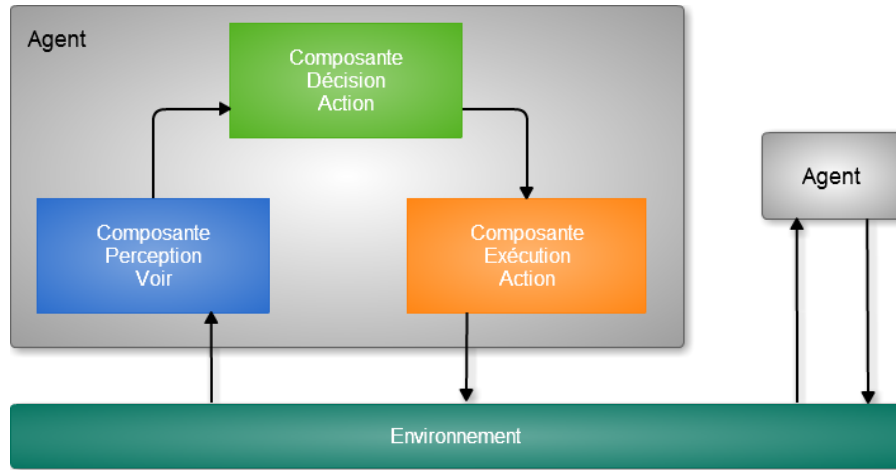


FIGURE 1.16 – Architecture générale d'un agent réactif

b) Les agents cognitifs

Ils sont parfois dits "intentionnels", leur caractéristique fondamentale est la volonté de communiquer et de coopérer, ils possèdent des buts à atteindre à l'aide d'un plan explicite. Les sociétés d'agents cognitifs contiennent communément un petit groupe d'individus de forte granularité, régit par des règles sociales prédéfinies (c'est-à-dire lors de situations conflictuelles les agents seront amenés à négocier). Ces agents sont capables à eux seuls d'exécuter des opérations complexes, ils peuvent raisonner en s'appuyant sur des bases de connaissances.

La figure 1.17 présente la structure générale d'un agent cognitif dans système multi-agents :

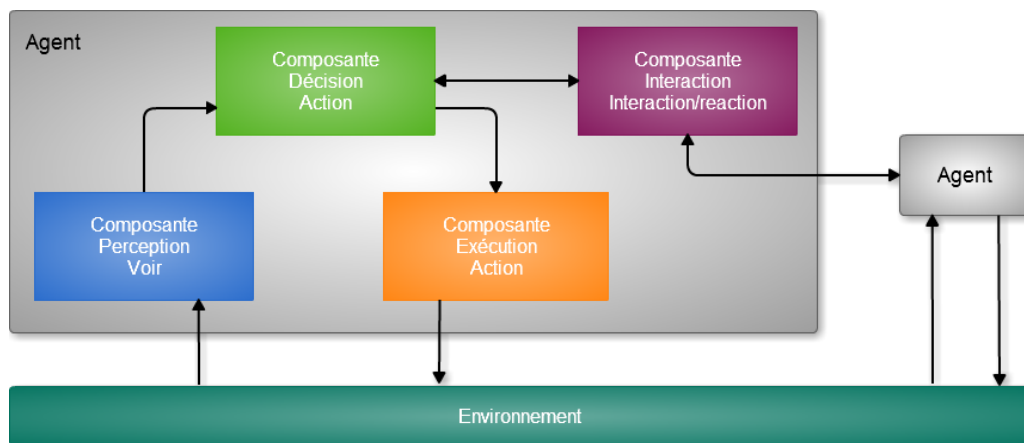


FIGURE 1.17 – Architecture générale d'un agent cognitif

c) Les agents hybrides

Dès le début des années 90, on savait que les systèmes réactifs pouvaient bien convenir pour certains types de problèmes et moins bien pour d'autres. De même, pour les agents cognitifs (notamment en l'intelligence artificielle).

On commença dès lors à investiguer la possibilité de combiner les deux approches (c'est à dire réaliser une graduation entre l'agent réactif pur, qui ne réagit qu'aux stimuli, et l'agent cognitif total, qui possède un modèle symbolique du monde qu'il met continuellement à jour et à partir duquel il planifie toutes ses actions) afin d'obtenir une architecture hybride en exploitant les avantages des deux architectures tout en éliminant leurs limitations.

Dans ce cas, un agent est composé de plusieurs couches, arrangées selon une hiérarchie, la plupart des architectures considèrent que trois couches suffisent amplement. Ainsi, au plus bas niveau de l'architecture, on retrouve habituellement une couche purement réactive, qui prend ses décisions en se basant sur des données brutes en provenance des senseurs. La couche intermédiaire fait abstraction des données brutes et travaille plutôt avec une vision qui se situe au niveau des connaissances de l'environnement. Finalement, la couche supérieure se charge des aspects sociaux de l'environnement, c'est-à-dire du raisonnement tenant compte des autres agents [22] La figure 1.18 et la figure 1.19 illustrent l'architecture en couche d'un agent hybride :

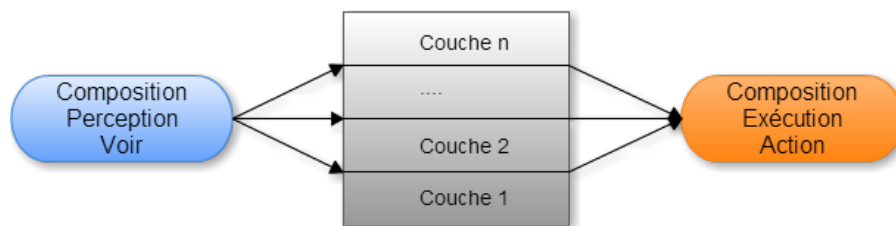


FIGURE 1.18 – Architecture horizontale d'agent hybride

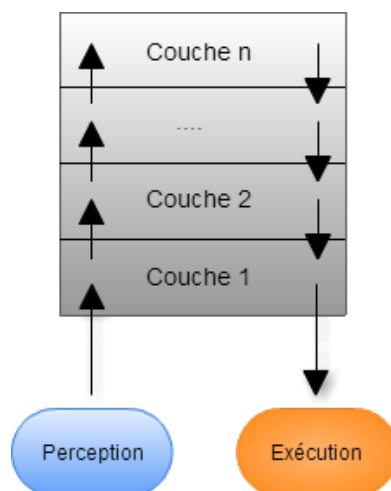


FIGURE 1.19 – Architecture verticale d'agent hybride

1.3.3 Système multi-agents

1.3.3.1 Définition

Usuellement, un système est un ensemble organisé d'éléments concourant à la réalisation d'une tâche donnée. En suivant cette définition on peut définir immédiatement le système Multi Agents comme étant un ensemble organisé d'agents se chargeant de réaliser un but commun. Les systèmes Multi Agents sont des systèmes distribués conçus et implantés idéalement comme un ensemble d'agents interagissant, le plus souvent, selon des modes de coopération, de concurrence et de coexistence.

1.3.3.2 Composition du système multi-agents

Un système multi-agent est un système composé des éléments suivants :

- **Un environnement E** : c'est-à-dire un espace disposant généralement d'une métrique.
- **Un ensemble d'agent A** : ce sont des objets particuliers, ils représentent les entités actives du système.
- **Un ensemble des objets O** : ces objets sont situés, c'est à dire que, pour tout objet, il est possible à un moment donné, d'associer une position dans E . Ces objets sont passifs, c'est à dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- **Un ensemble de relation R** : ce sont les différents types de manipulation qu'appliquent les agents sur les objets du système et qui sont en générale : perception production, consommation, transformation...etc.
- **Un ensemble d'opérateurs OR** : permettant aux agents A de percevoir produire, consommer, transformer et manipuler les objets de O .

1.3.3.3 Caractéristiques d'un système multi-agents

Un SMA possède généralement les caractéristiques suivantes :

- Chaque agent dispose d'informations ou de capacités de résolution de problèmes limités (ainsi, chaque agent a un point de vue partiel)
- Il n'y a aucun contrôle global du système multi-agent.
- Les données sont décentralisées
- Le calcul est asynchrone

La figure 1.20 donne une image globale d'un système Multi-Agents :

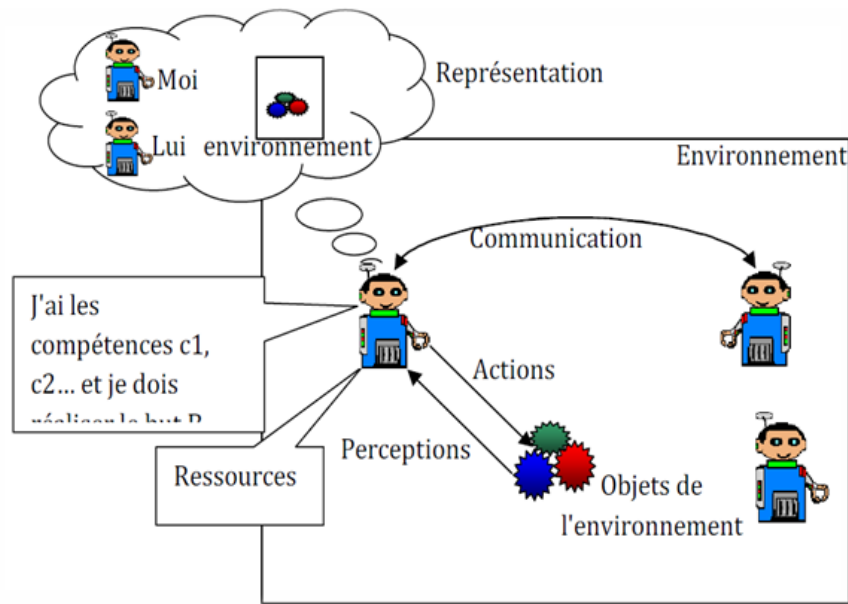


FIGURE 1.20 – Architecture globale d'un système multi-agents

1.3.4 Interaction dans un système Multi-Agents

Définition de l'interaction

Jacques Ferber [3] définit l'interaction comme : *"une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble de relations réciproques"*. Les interactions sont non seulement la conséquence d'actions effectuées par plusieurs agents en même temps, mais aussi l'élément nécessaire à la constitution d'organisations sociales.

En général, les interactions sont mises en œuvre par un transfert d'informations entre agents ou entre l'environnement et les agents ; soit par perception, soit par communication.

L'interaction entre les agents apparaît sous plusieurs modes, qui sont la coopération, la coordination et la négociation.

1.3.4.1 La coopération

Parmi les caractéristiques fondamentales d'un système Multi Agents, on trouve la distribution du travail entre les différents agents qui le constituent, ou chacun d'eux se charge de réaliser ses propres buts (qui sont un sous-problème du problème global).

Chaque agent possède un ensemble de compétences qui lui permettent de résoudre les différents problèmes, mais il existe des situations où ses capacités et ses compétences ne suffisent pas à accomplir certaines tâches (ou bien il ne dispose pas des moyens nécessaires) donc il aura besoin de l'intervention d'un autre agent du système qui va l'aider à résoudre le problème, c'est-à-dire qu'il y'a une coopération pour faire évoluer le système vers ses objectifs. La coopération

consiste, donc, à faire participer plusieurs agents pour satisfaire un but individuel ou commun.

Jacques Ferber définit la coopération entre plusieurs agents comme *"une situation dans laquelle, soit l'ajout d'un nouvel agent permet d'accroître les performances du groupe, soit l'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels "*.

1.3.4.2 La négociation

On définit la négociation comme le processus d'améliorer les accords (en réduisant les inconsistances et l'incertitude) sur des points de vue communs ou des plans d'action grâce à l'échange structuré d'informations pertinentes.

Contrairement à la coopération qui suppose la sociabilité des agents, la négociation correspond à la collaboration entre agents en univers compétitif. La négociation est un mécanisme puissant pour gérer les dépendances inter agents ou le processus par lequel un groupe d'agents arrive à une décision mutuelle acceptable sur un sujet donné, elle représente un axe fondamental qui distingue un agent d'un objet.

Pour mener à bien le processus de négociation, il est nécessaire de suivre un protocole qui facilite la convergence vers la solution, la négociation est caractérisée en général par un protocole minimal d'actions qui est : proposer, évaluer, accepter ou refuser une solution.

1.3.4.3 La coordination

Quand les agents utilisent des ressources communes ou résolvent des problèmes qui ne sont pas complètement indépendants mais liés et complémentaires, les agents du système doivent accomplir en plus de leurs tâches de résolution des problèmes individuels, des tâches supplémentaires (appelées tâches de coordination) qui améliorent le fonctionnement du système.

Jacques Ferber donne la définition suivante : *"la coordination d'actions dans un système Multi-Agents est définie comme l'ensemble des tâches effectuées par les agents pour réaliser les autres actions (actions effectives) dans les meilleures conditions"*.

La coordination entre les agents d'un système apparaît sous deux formes distinctes, elle sert d'une part à éviter les problèmes, et à améliorer le fonctionnement du système d'autre part.

Sa première forme consiste à bien coordonner les plans de fonctionnement des agents pour assurer une meilleure gestion des ressources (notamment celles qui sont rares) et éviter les conflits d'accès, c'est la coordination due à la gêne.

Sa deuxième forme est la coordination due à l'aide, comme le signifie son nom, les agents doivent synchroniser leurs actions, et échanger les résultats qui sont nécessaires au fonctionnement des autres.

1.3.5 Communication dans les systèmes Multi-Agents

La communication représente la base de réalisation de tous les modes d'interaction qu'on a vu précédemment, soit la négociation ou bien la coordination...

Elle est définie comme une forme d'action locale d'un agent vers d'autres agents. Les questions abordées par un modèle de communication peuvent être résumées par l'interrogation suivante : qui communique, quoi, à qui, quand, pourquoi, et comment ?

- **Pourquoi les agents communiquent-ils ?** La communication doit permettre la mise en œuvre de l'interaction et par conséquent la coopération et la coordination d'actions.
- **Quand est ce que les agents communiquent-ils ?** Les agents sont souvent confrontés à des situations où ils ont besoin d'interagir avec d'autres agents pour atteindre leurs buts locaux ou globaux. La difficulté réside dans l'identification de ces situations.
- **Avec qui les agents communiquent-ils ?** Les communications peuvent être sélectives sur un nombre restreint d'agents ou diffusées à l'ensemble d'agents.
- **Comment les agents communiquent-ils ?** La mise en œuvre de la communication nécessite un langage de communication compréhensible et commun à tous les agents. Il faut identifier les différents types de communication et définir les moyens permettant non seulement l'envoi et la réception de données mais aussi le transfert de connaissances avec une sémantique appropriée à chaque type de message. Il existe principalement deux modes de communication :
 - Communication par tableau noir : dans les systèmes fonctionnant par partage de ressources, les différents composants ne sont pas directement liés entre eux. Ils communiquent au travers d'une zone de données commune appelée tableau noir (Blackboard), dans laquelle sont stockées les connaissances du système.

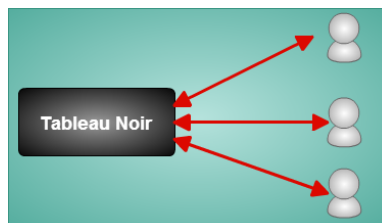


FIGURE 1.21 – Communication via un Tableau Noir

- Communication par envoi de messages : dans les systèmes où la communication se fait par envoi de messages, les connaissances sont distribuées entre les différents agents. Chacun d'eux communique directement avec les autres par envoi de messages.

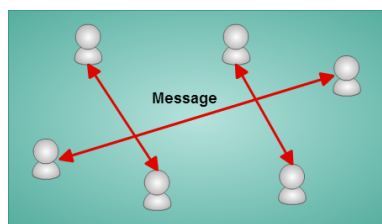


FIGURE 1.22 – Communication par envoi de messages

1.4 Conclusion

Dans ce chapitre, nous avons dressé l'état de l'art du formalisme CSP par quelque définition, exemples des extensions et méthodes de résolution. Puis nous avons donné une revue sur l'algorithme des essais particuliers. Enfin, nous avons également donné un aperçu sur les systèmes multi-agents. Le chapitre suivant sera dédié à la présentation de la méthode de résolution d'un CSP sous une architecture multi-agents.

Chapitre 2

Algorithme d'optimisation par Essaim Particulaires Distribué pour les problèmes MaxCSP

Introduction

Dans ce chapitre, nous présentons notre approche en exposant les différentes étapes et fonctions développées pour l'implémentation de l'algorithme PSO distribué pour le problème Max-CSP. Puis nous présentons une conception du modèle à implémenter.

2.1 Présentation de l'approche

2.1.1 Générateur aléatoire de CSP

Etant donné un CSP formé de (X, D, C, R) comme décrit dans le chapitre 1. Nous avons choisi comme nombre de variable appartenant à X et comme taille de chaque domaine D_i de D . La génération du CSP se fait en deux étapes :

- ❖ La première consiste à générer les variables et leurs domaines
- ❖ la deuxième est la génération des contraintes.

Dans le cadre de notre projet, nous utiliserons la génération de contraintes binaires définies en extension (exprimées sous la forme d'un ensemble de tuples autorisés c'est-à-dire l'ensemble des affectations des variables satisfaisant les contraintes).

Après la génération du CSP, nous déterminons une instance aléatoire à partir du CSP généré : c'est la population initiale c'est pour cette raison qu'il faut mettre le CSP généré dans une forme exploitable par PSO.

la figure 2.1 illustre le générateur aléatoire de CSP [23] :

```
Connectivite =0.6 / durete =0.5
/* génération des variables et leurs domaines */
for 1 to nbvariables et leurs domaines
    aleatoirei = rand (-10,10) ;
    randomIndex = trunc(aleatoirei) + 1
    tailedomainei = rand(2,5)
    for 1 to tailedomainei do
        listevARIABLES += randomIndex
        randomIndex = randomIndex + 1
    end for
end for
```

```
/*génération des contraintes binaires définie en extension*/
for 1 to nbvariables do
    for i+1 to nbvariables do
        x = rand(0,1)
        if x < connectivite then
            nbrContraintes = nbContraintes + 1
            for 1 to tailedomainei do
                for 1 to tailedomainej do
                    y = rand(0,1)
                    if y < durete then
                        contrainte0 = listevariablesi
                        contrainte1 = listevariablesj
                    end if
                end for
            end for
        end if
    end for
end for
```

FIGURE 2.1 – Algorithme de générateur aléatoire de CSP

2.1.2 Correspondances entre CSP et PSO

Pour pouvoir utiliser l'algorithme PSO dans la résolution des CSP, il faut mettre les CSP dans une forme exploitable par l'algorithme. Nous supposons que le CSP comporte n variables et chaque variable peut être assignée par une seule valeur de son domaine. Chaque contrainte peut inclure de 1 à n variables.

D'abord, le PSO demande un espace de recherche où les particules peuvent se déplacer. Donc, nous pouvons utiliser le nombre des variables de CSP comme dimension de l'espace de recherche, une variable pour chaque dimension. Dans chaque dimension i la particule ne prend qu'une valeur qui est dans le domaine de la variable i . Donc une position d'une particule est une instantiation complète des variables $(var_1, var_2, \dots, var_n)$.

la figure 2.2 représente une position x_i d'une particule :

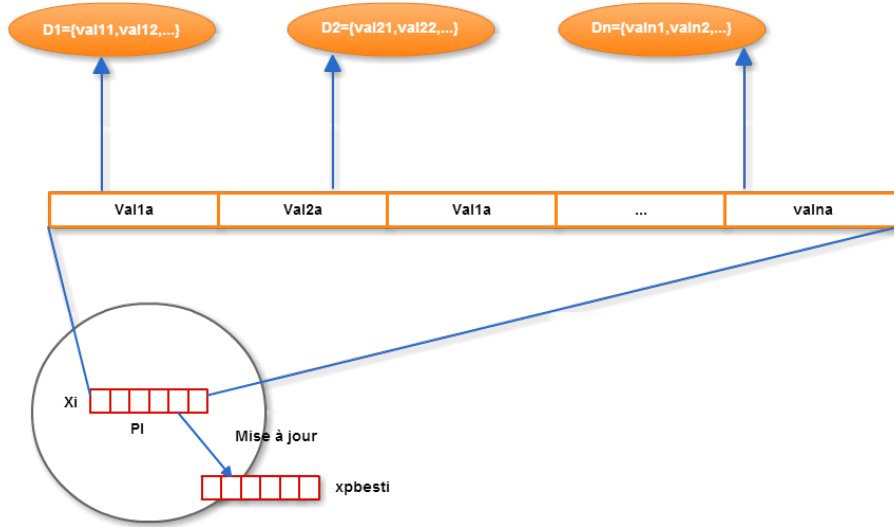


FIGURE 2.2 – La position x_i d'une particule

La vitesse $V_i(t)$ d'une particule P_i est un vecteur de dimension n qui permet le changement de la position de la particule $X_i(t-1)$ vers la position $X_i(t)$. Ainsi, le changement de la position d'une particule à travers la vitesse V_i est donné par la figure 2.3 :

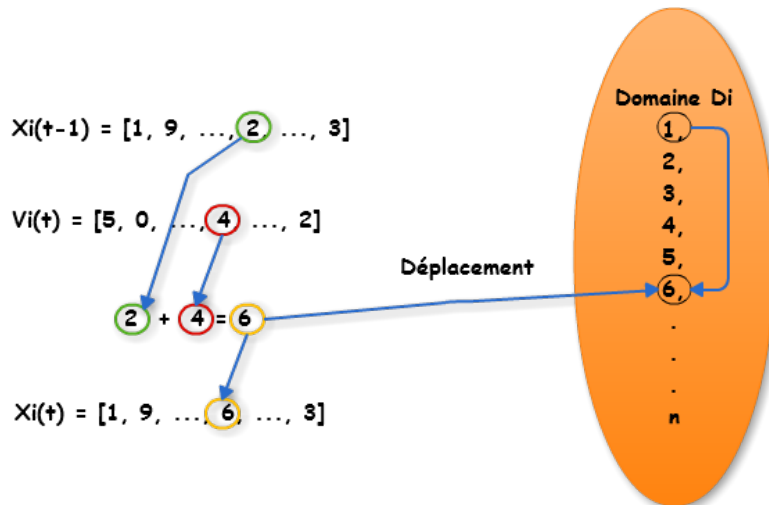


FIGURE 2.3 – Le changement de la position d'une particule à travers la vitesse V_i

2.1.3 Concept de Template

L'approche proposée s'inspire de l'algorithme D^3G^2A de S. BOUAMAMA et K. GHEDIRA [7]. A chaque variable de la vectrice position, nous allons attacher une nouvelle structure de donnée que nous appelons Template. Pour une $variable_{i,j}$ on associe un $Template_{i,j}$ où les indices i et j désigne la variable j de la particule i . La $Template_{i,j}$ de la $variable_{i,j}$ contient un poids qui représente le nombre des contraintes violées par cette variable. Un poids est un entier positif qui ne dépasse pas le nombre total des contraintes du problème. Pour chaque vecteur de position, l'opérateur de pénalisation commence par l'initialisation de tous les poids de Template

correspondant. Puis il vérifie la violation des contraintes. Pour une contrainte violée C_j , tous les poids des variables impliquées dans cette contrainte sont incrémentés et le traitement sera répété pour toutes les contraintes. Pour chaque nouvelle position, l'opérateur de pénalisation vérifie la violation des contraintes et met à jour les poids de la Template associée.

La figure 2.4 présente un exemple sur la notion de Template :

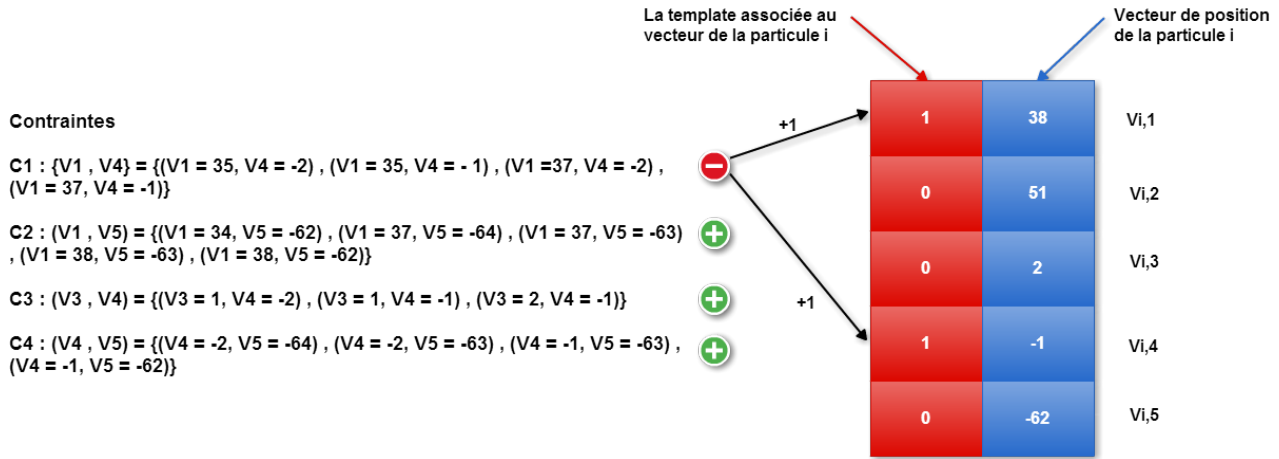


FIGURE 2.4 – Exemple sur la notion de Template

2.1.4 Dynamique globale

L'idée de la dynamique globale est de partitionner la population initiale en sous-populations et d'affecter chacune à un agent qui s'appelle " Agent Espèce ". Une sous-population est composée des particules qui ont la fonction de fitness dans le même FFR (Fitness Function Range), qui est la spécificité de cet agent. Aussi un autre type d'agent intermédiaire qui existe qui s'appelle " Agent Interface " qui est nécessaire entre la société des agents Espèces et l'utilisateur. Cet agent a pour objectif de détecter les optimums, récupérer les paramètres depuis l'utilisateur et il se charge de la création des nouveaux "Agents Espèces " si nécessaire.

L'agent interface génère un CSP, ensuite à partir du CPS généré, il détermine une instance aléatoire qui présente la population initiale des particules. Par la suite, l'agent interface calcule la fitness de toutes les particules initiales. Les différentes valeurs de fitness trouvées représentent le nombre de spécificités.

Avant que l'agent espèce commence son processus d'optimisation, chaque agent espèces initialise les Templates puis il exécute l'algorithme PSO sur la sous-population attribuée par l'agent Interface au début.

A chaque génération, l'agent vérifie la valeur de fitness de chaque particule de sa sous population. Si ce nombre est égal à la spécificité de l'agent à laquelle la particule appartient alors cet agent conserve cette particule dans sa souspopulation. Dans le cas contraire, deux types de traitements sont possibles :

- S'il existe un agent dont la spécificité peut contenir la valeur de fitness de la particule courante donc cette particule sera envoyée à cet agent.

- Sinon la particule sera envoyée à l'agent interface qui va créer un autre agent espèce avec la spécificité convenable et lui envoie la particule correspondante.

Après avoir créé un nouvel agent espèce, l'agent interface informe tous les autres agents espèces sur son existence.

Les agents poursuivant l'exécution de processus tant que le critère d'arrêt n'ait pas été atteint qui est le nombre des générations fixé par l'utilisateur au début.

La figure 2.5 présenter le diagramme qui résume le déroulement de processus de résolution :

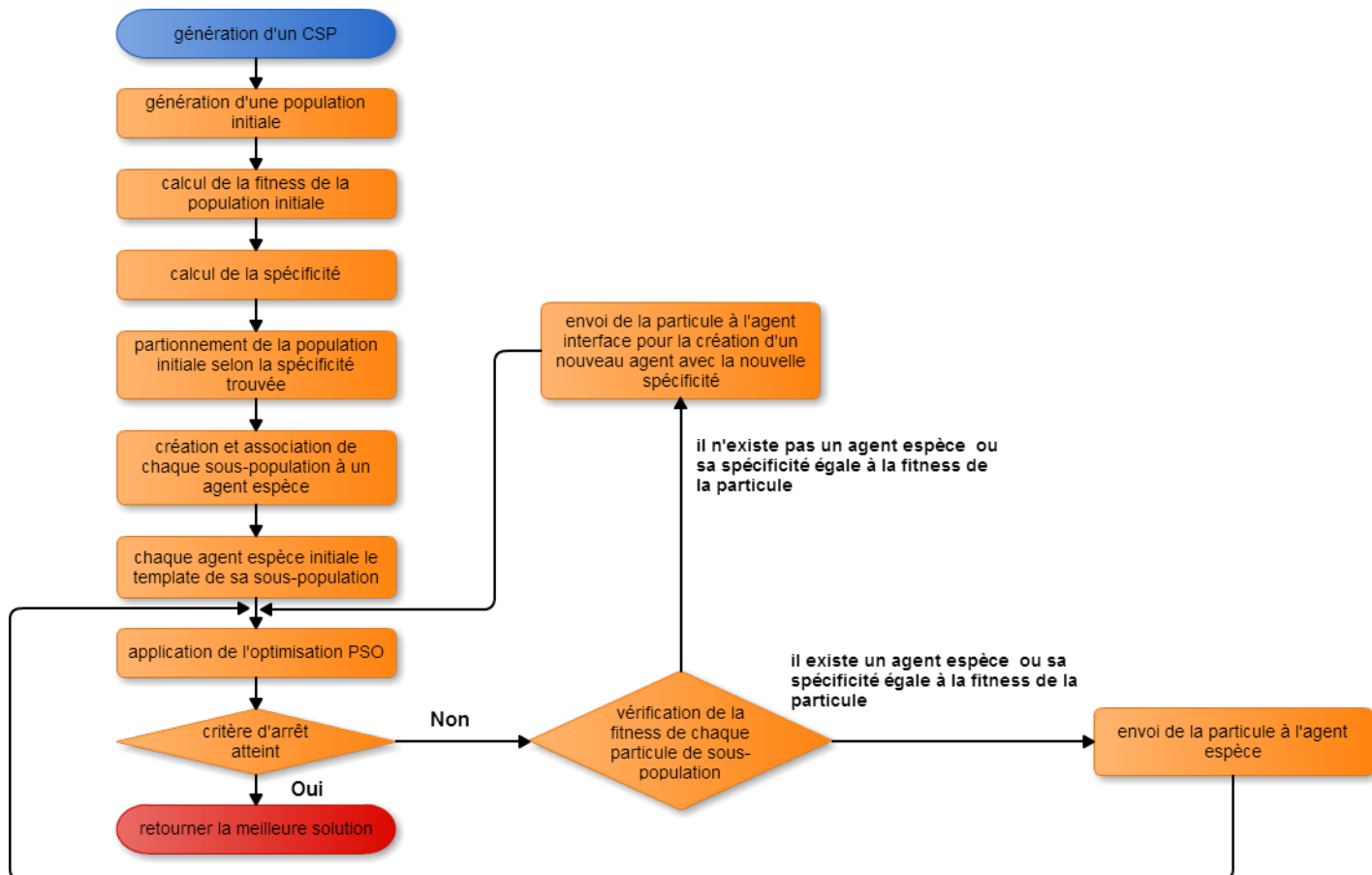


FIGURE 2.5 – Diagramme des étapes de dynamique globale

la figure 2.6 détaille d'une façon algorithmique le comportement d'un agent espèce lors de processus de résolution :

```
Initialiser les connaissances locales;
Tant que Le nombre maximal de la génération n'est pas atteint faire
    Pour Chaque particule  $j$  de la sous population faire
         $FV_j$  Calcule de la valeur de fitness augmenté ( $Particule_j$ );
    Fin
    Si  $FV_j$  meilleure que Best-FV alors
        Best-FV  $\leftarrow FV_j$ ;
        Best-Particule  $\leftarrow Particule_j$  ;
    Fin
    Si  $FV_j \notin range_i$  alors
        Si Existe-agent ( $Espèce_{FV}$ ) alors
            Envoyer
                Msg ( $Espèce_i, Espèce_{FV}, take - into - account( Particule_j)$ )
        Sinon
            Envoyer Msg ( $Espèce_i, Interface, create - agent( Particule_j)$ )
    Fin
Fin
```

FIGURE 2.6 – Algorithme de comportement d'un agent espèce

2.1.5 Structure des agents

Chaque agent dans cette approche possède une structure simple qui se résume dans deux types de connaissance : dynamiques et statiques.

2.1.5.1 L'agent interface

L'agent interface possède :

- Le problème à résoudre.
- Les différents paramètres introduits par l'utilisateur.
- Connaissance statique : la meilleure position visitée par les différentes particules.
- Connaissance dynamique : les accointances (liste de tous les agents espèces et leur spécificité).

2.1.5.2 L'agent espèce

Chaque agent espèce possède :

- Une instance de problème à résoudre.
- Les paramètres de son algorithme PSO.
- Sa spécificité.
- Connaissance statique : la sous-population qu'il gère à chaque génération.
- Connaissance dynamique : les accointances (les adresses de tous les agents espèces et de l'agent interface).

2.2 Conception

2.2.1 Architecture globale

Pour commencer le processus d'optimisation, l'utilisateur commence par donner les paramètres nécessaires (Nombre des générations, Nombre des variables, Nombre des particules ...) à l'agent interface. Puis il lance l'exécution. Enfin, il récupère la solution trouvée.

La figure 2.7 donne une idée générale sur l'architecture globale de notre application permettant la résolution d'un CSP à l'aide de PSO sous une architecture multi-agents.

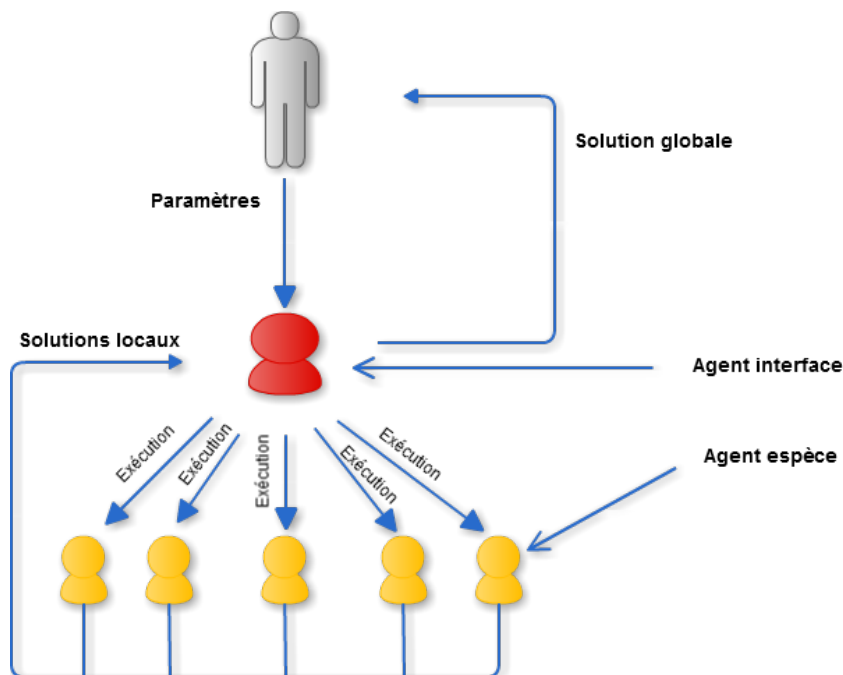


FIGURE 2.7 – Architecture globale de l'application

2.2.2 Langage de Modélisation

a) Choix de langage de modélisation

Il est certes que nous adoptons UML comme langage de modélisation puisque nous allons utiliser le concept de l'orienter objet pour développer notre application. Ainsi, la méthodologie de conception adoptée se base sur le choix de diagrammes UML adéquats. Nous avons utilisé trois types des diagrammes différents : diagrammes de cas d'utilisation, diagrammes de séquence et diagramme de classes.

b) Définition

UML (Unified Modeling Language) est un langage de modélisation des systèmes standard, qui utilise des diagrammes pour représenter chaque aspect d'un système (statique, dynamique, ...) en s'appuyant sur la notion d'orienté objet qui est un véritable atout pour ce langage.

UML permet de modéliser d'une manière claire et précise la structure et le comportement d'un système. Il est un moyen d'exprimer des modèles objets en faisant abstraction de leur implémentation, c'est-à-dire que le modèle fournit par UML est valable pour n'importe quel langage de programmation .

2.2.3 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est une représentation du comportement du système de point de vue de l'utilisateur, c'est une définition des besoins qu'attend un utilisateur du système, il contient tous les cas d'utilisation en liaison directe ou indirecte avec les acteurs.

La figure 2.8 représente le diagramme de cas d'utilisation générale de notre application :

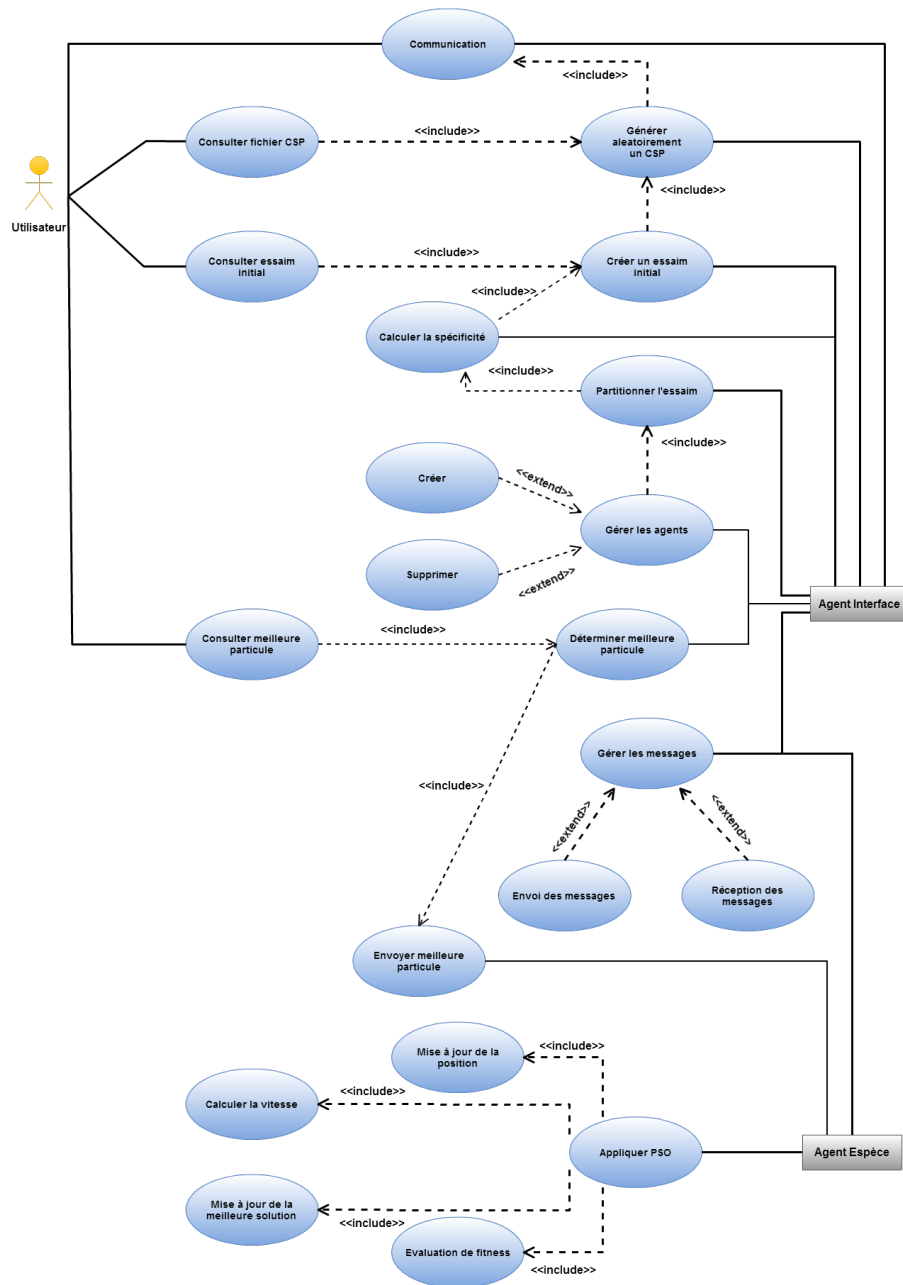


FIGURE 2.8 – Diagramme de cas d'utilisation générale de l'application

les acteurs :

En se basant sur les besoins de notre application on distingue deux types des acteurs qui sont :

- **Acteur humain**

- **Utilisateur** : représente la personne qui interagit avec l'application, il a le droit d'effectuer plusieurs tâches telles que :
 - * La définition les paramètres
 - * Consulter les résultats (consulter fichier CSP, consulter essaim initiale ...)
 - * Etc...

- **Acteur système**

- **L'agent interface** : c'est une entité logicielle qui peut :
 - * récupérer les paramètres définis par l'utilisateur
 - * Générer aléatoire un problème CSP
 - * Générer une population initiale des particules d'une manière aléatoire
 - * Partitionner l'essaim initial selon une spécificité
 - * Créer autant d'agents espèces que le nombre des spécificités distinctes des particules générées aléatoirement
 - * Déterminer la meilleur solution globale
 - * Etc...
- **L'agent espèce** : c'est une entité logicielle qui :
 - * Initialise le Template de chaque particule de sous population
 - * Exécute l'algorithme d'optimisation PSO
 - * Vérifier la valeur de fitness de chaque particule de sous population
 - * Envoyer la meilleure solution locale
 - * Etc...

2.2.4 Diagrammes de séquences

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique. Le diagramme de séquences permet de cacher les interactions d'objets dans le cadre de scénario d'un diagramme de cas d'utilisation. Donc dans cette partie, nous allons choisir quelques cas d'utilisation que nous présentons sous forme de diagramme des séquences .

Cas d'utilisation :

- **Communication**

- *Titre* : communication

- *Acteur principale* : utilisateur
- *Acteur secondaire* : agent interface
- *L'objectif* : permettre l'utilisateur de définir les valeurs de paramètres selon son choix.
- *Pré-condition* : l'interface de saisir de paramètres est affiché.
- *Post-condition* : l'interface des résultats de la résolution du problème est affichée.

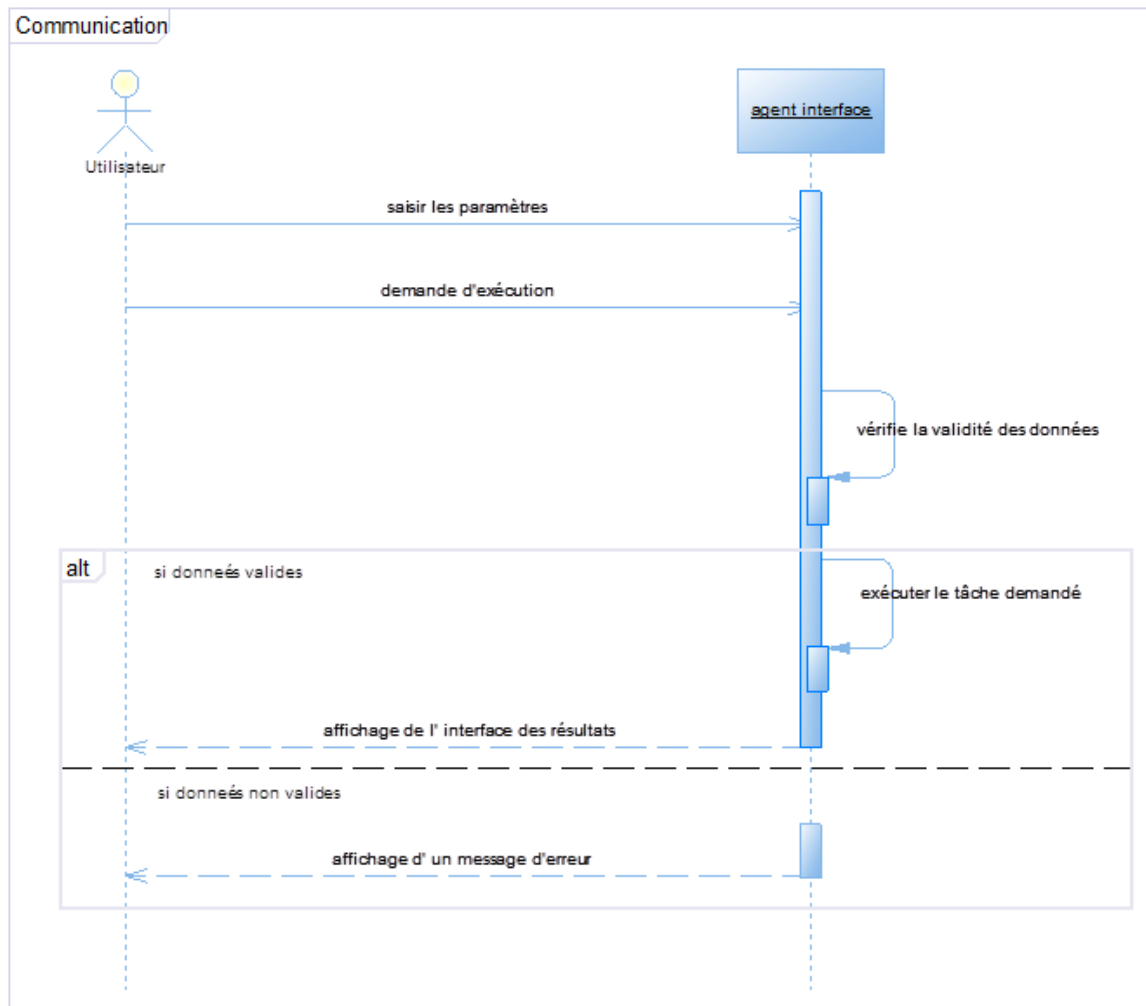


FIGURE 2.9 – Diagramme de séquence de Communication

- Générer aléatoirement un CSP

- *Titre* : générer aléatoirement un CSP
- *Acteur* : agent interface
- *L'objectif* : modéliser un problème CSP.
- *Pré-condition* : la saisie des paramètres par l'utilisateur s'est déroulée avec succès.
- *Post-condition* : le CSP est généré.

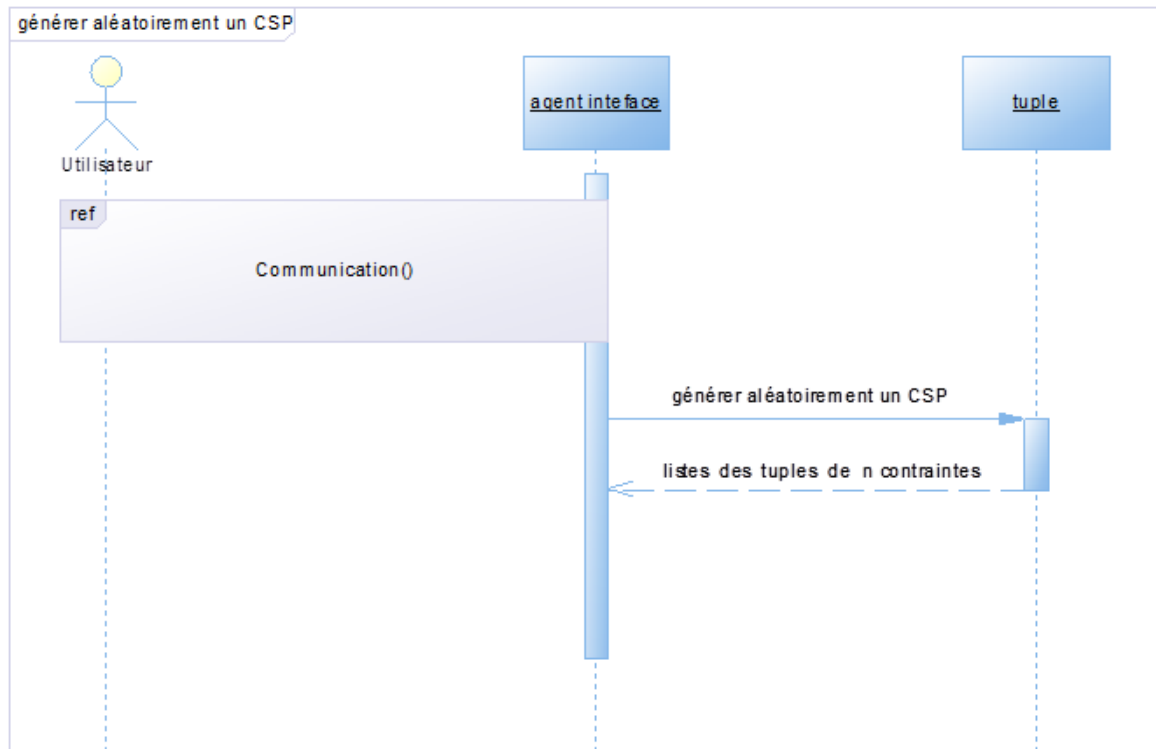


FIGURE 2.10 – Daigramme de séquence de génération aléatoire de CSP

- Créer un essaim initial

- *Titre* : créer un essaim initial
- *Acteur* : agent interface
- *L'objectif* : générer une population initiale des particules d'une manière aléatoire.
- *Pré-condition* : le CSP est généré.
- *Post-condition* : l'essaim initial est créé.

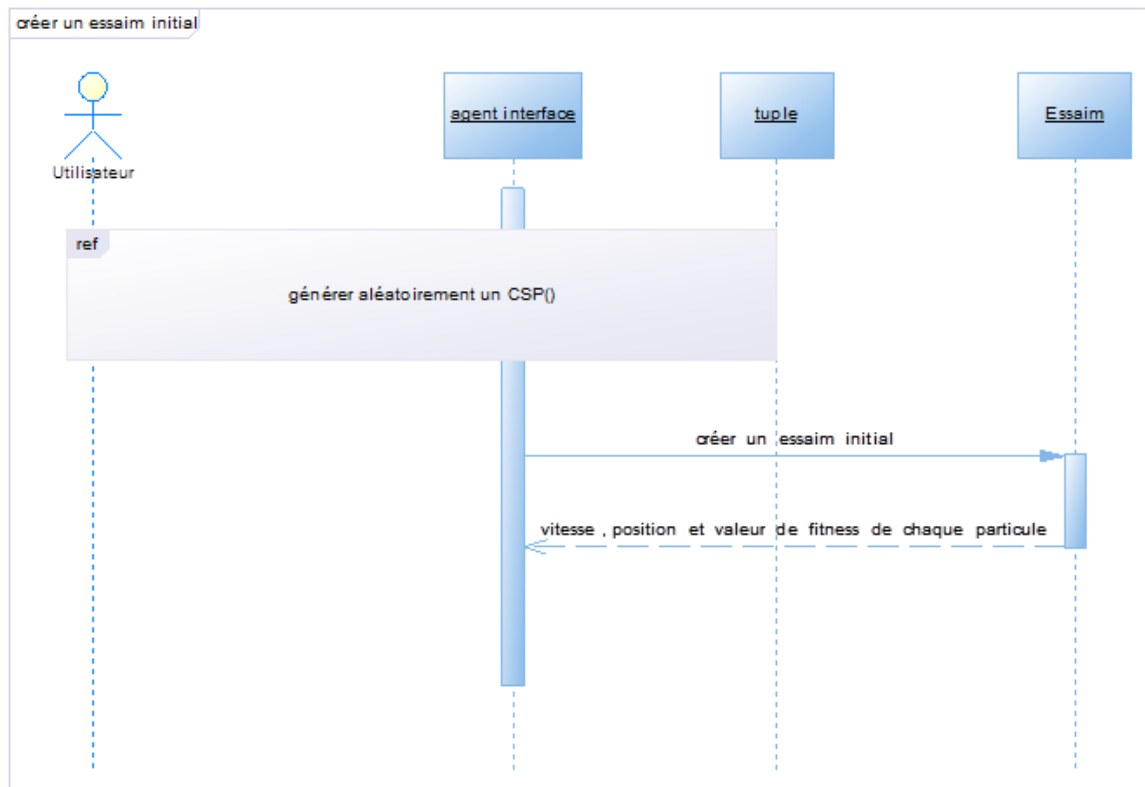


FIGURE 2.11 – Diagramme de séquence de création d'un essaim initial

- Calculer la spécificité

- **Titre** : Calculer la spécificité
- **Acteur** : agent interface
- **L'objectif** : généraliser un critère de choix appelé spécificité selon lequel on partitionne l'essaim initial.
- **Pré-condition** : l'essaim initial est créé .
- **Post-condition** : la spécificité est calculée.

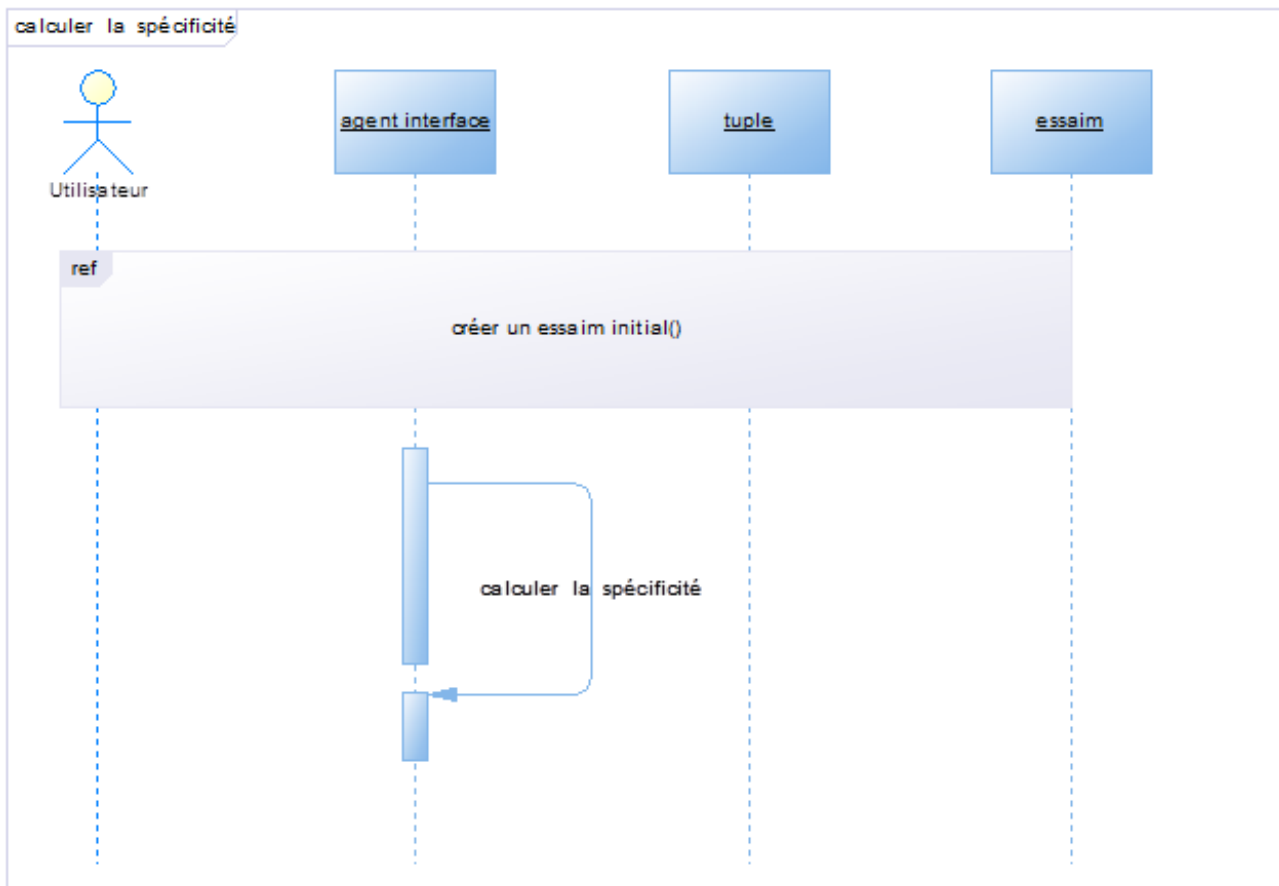


FIGURE 2.12 – Diagramme de séquence de calcul de la spécificité

- Partitionner l'essaim initial

- *Titre* : partitionner l'essaim initial
- *Acteur* : agent interface
- *L'objectif* : partitionner l'essaim initial en de sous essais selon une spécificité.
- *Pré-condition* : la spécificité est calculée.
- *Post-condition* : l'essaim initial est créé.

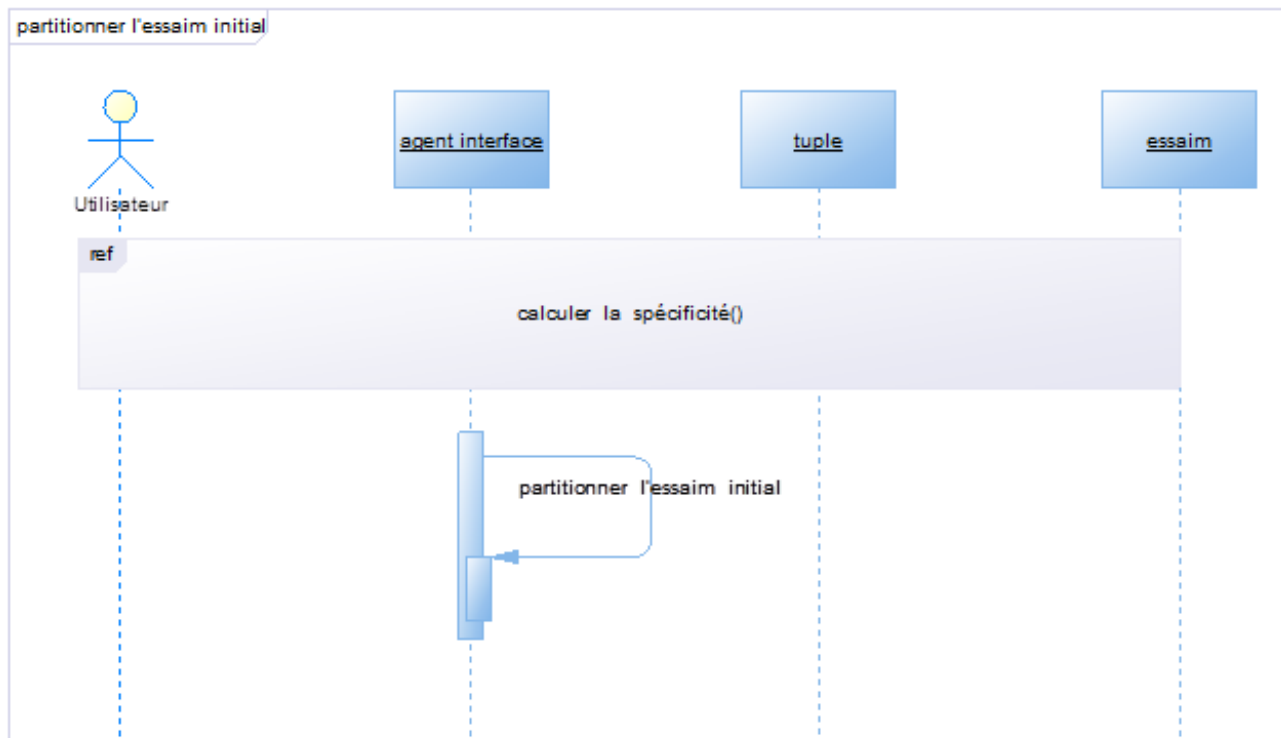


FIGURE 2.13 – Diagramme de séquence de partitionnement de l'essaim initial

- Créer les agents espèces

- *Titre* : créer les agents espèces
- *Acteur* : agent interface
- *L'objectif* : créer des agents espèces en tant que le nombre de sous essais qu'on a et faire repartir les sous essais sur les agents espèces.
- *Pré-condition* : l'essaim initial est partitionné.
- *Post-condition* : les agents espèces sont créés .



FIGURE 2.14 – Diagramme de séquence de création des agents espèces

- Appliquer PSO

- *Titre* : appliquer PSO
- *Acteur principale* : agents espèce
- *Acteur secondaire* : agent interface
- *L'objectif* : exécuter de processus d'optimisation PSO.
- *Pré-condition* : les agents espèces sont créés.
- *Post-condition* : PSO est appliqué .

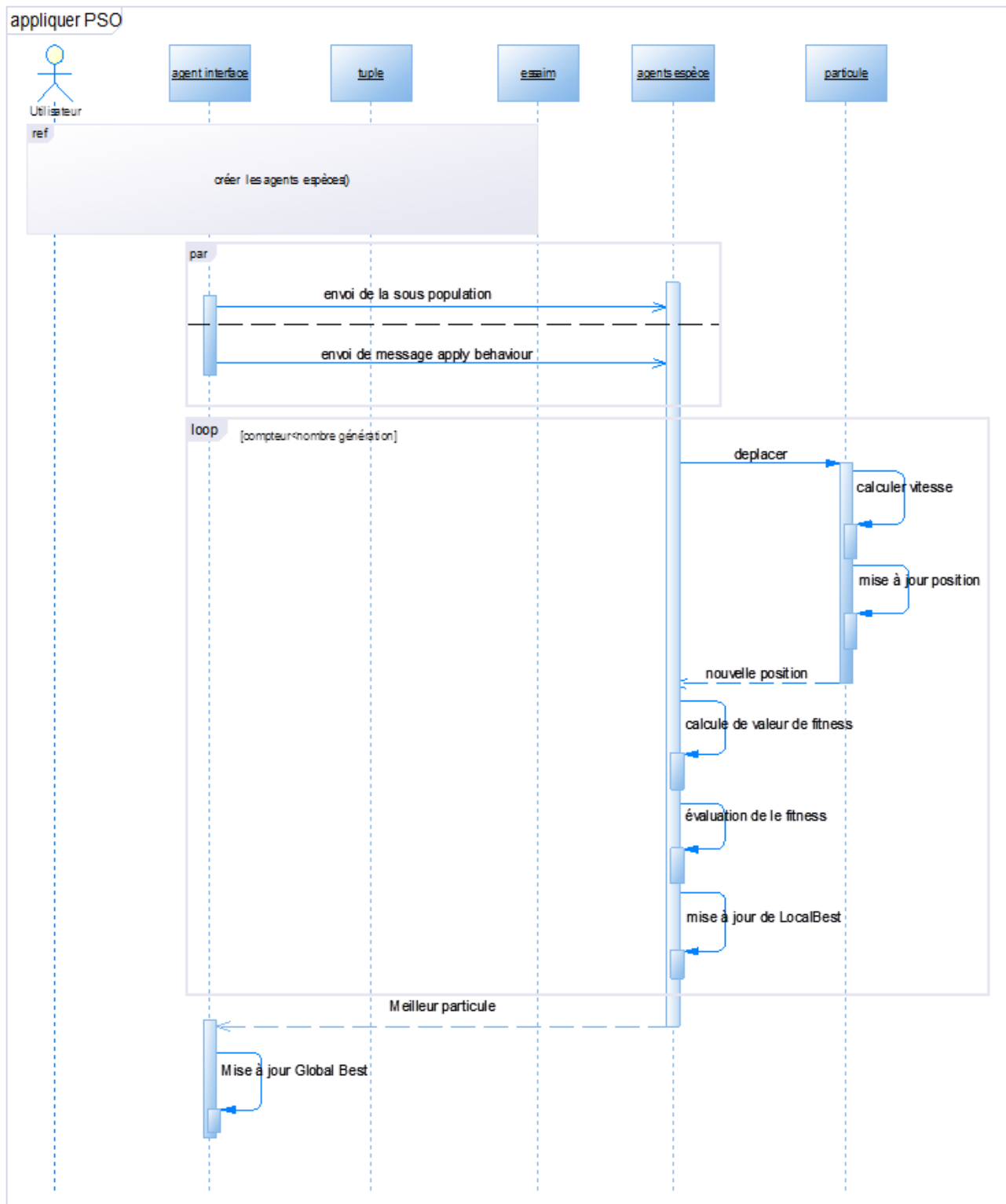


FIGURE 2.15 – Diagramme de séquence de l'application de PSO

• Evaluation de fitness

- *Titre* : évaluation de fitness
- *Acteur principale* : agents espèce
- *Acteur secondaire* : agent interface
- *L'objectif* : vérifier la valeur de fitness de chaque particule appartenant à la sous population d'un agent espèce .
- *Pré-condition* : calcule de la valeur de fitness de chaque particule dans le sous population d'une espèce .
- *Post-condition* : Mise à jour de la meilleure particule locale .

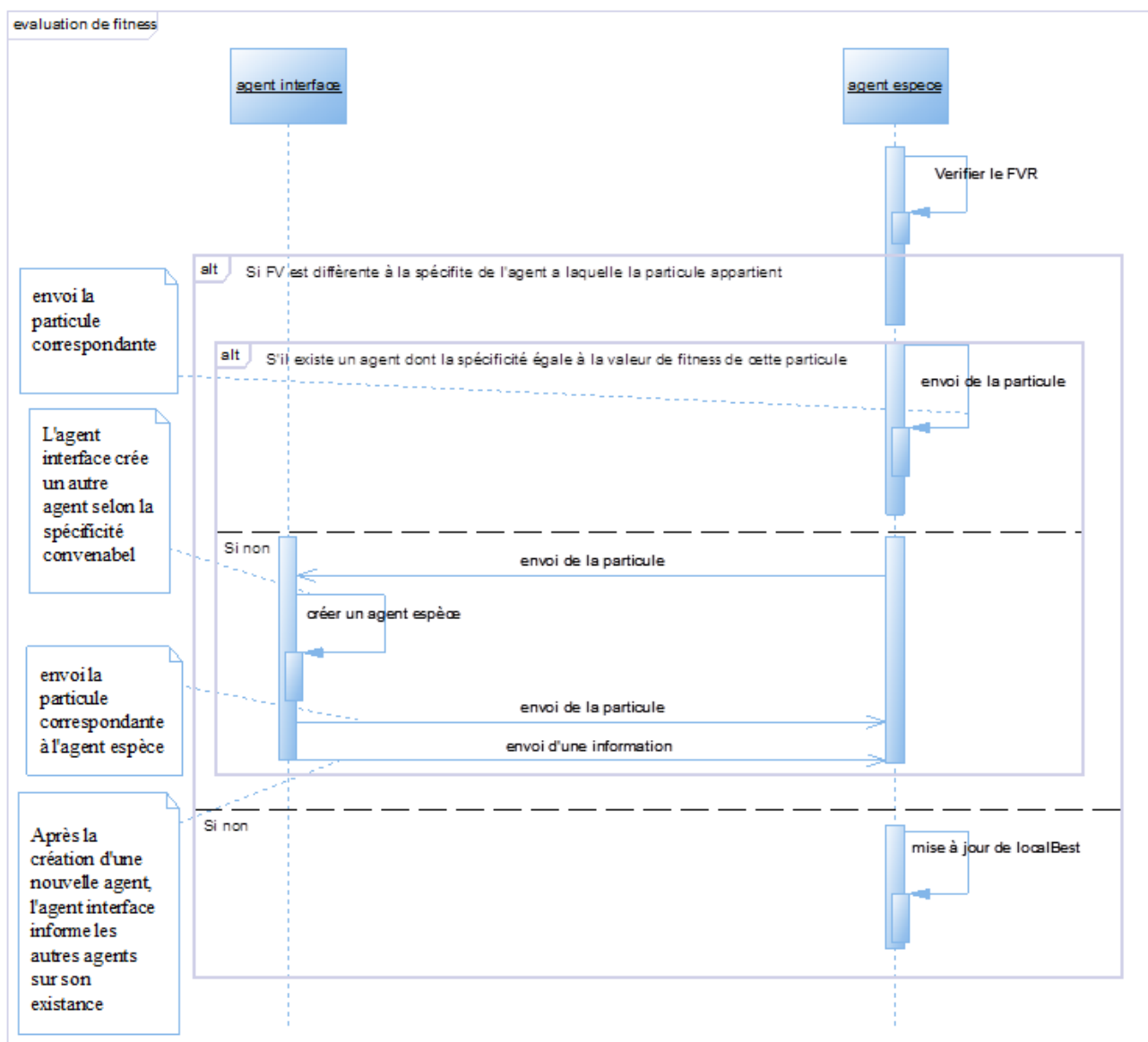


FIGURE 2.16 – Diagramme d'évaluation de fitness

- consulter fichier CSP

- *Titre* : consulter fichier CSP
- *Acteur principale* : utilisateur
- *Acteur secondaire* : agent interface
- *L'objectif* : affichage de fichier CSP .
- *Pré-condition* : le PSO est appliqué .
- *Post-condition* : le fichier CSP est affiché.



FIGURE 2.17 – Diagramme de séquence de consultation de fichier CSP

2.2.5 Diagramme de classe

Un diagramme de classe dans le langage de modélisation unifié (UML) est un type de diagramme de structure statique qui décrit la structure d'un système en montrant le système des classes, leurs attributs, les opérations(ou) les méthodes et les relations entre les classes.

la figure 2.17 ci-dessous, représente le diagramme de classes de notre système.

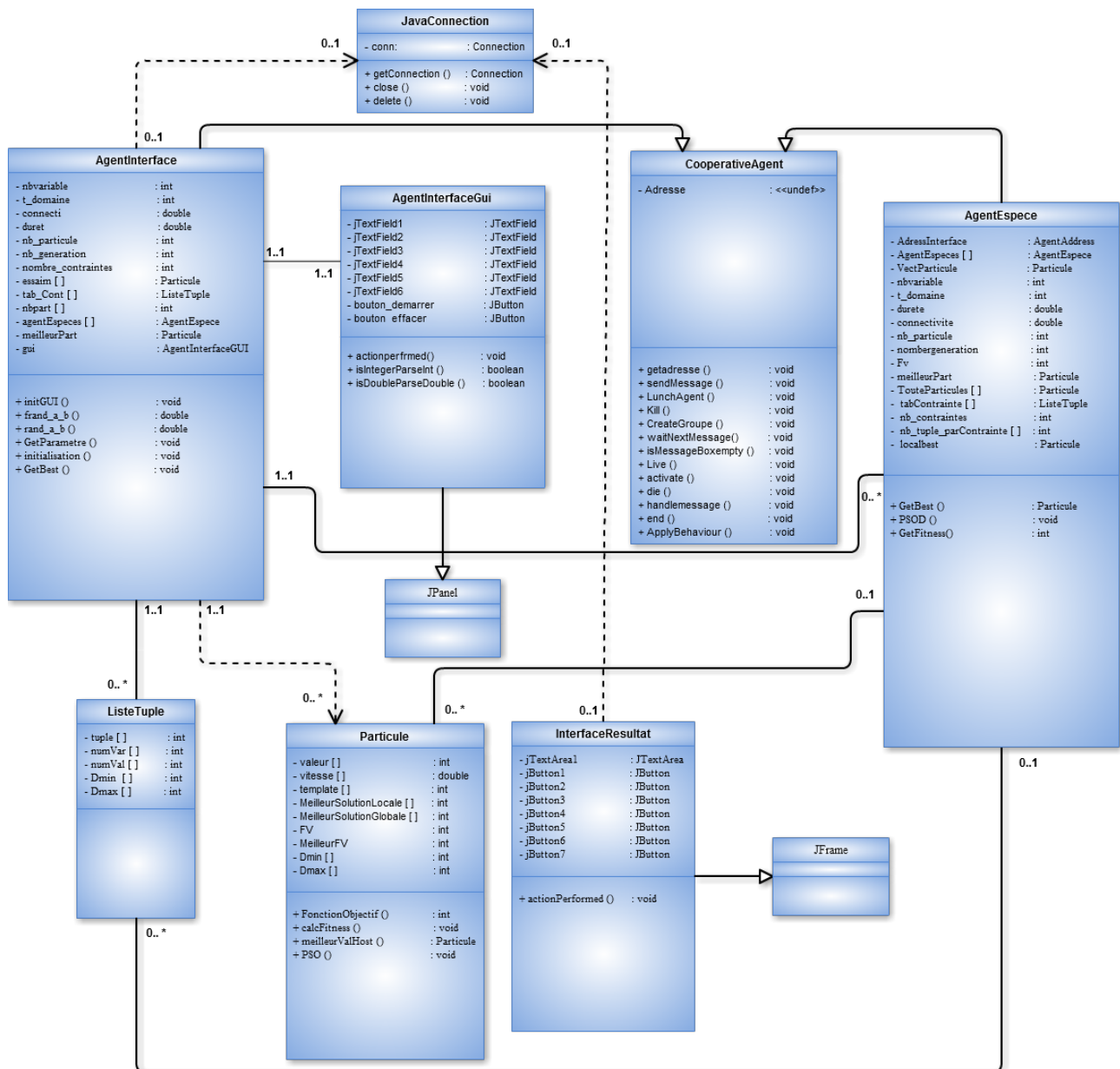


FIGURE 2.18 – Diagramme de classes

Les classes

- **JavaConnection** : cette classe permet l'établissement de connexion à la base de données, elle comporte un seul attribut *conn* de type *Connection*. Les méthodes fournies par cette classe sont *getConnection* qui permet de retourner un objet de type *Connection*, *close* qui assure la fermeture de connexion et *delete* qui permet la suppression de données.
- **AgentInterface** : est une classe qui hérite de la classe mère *CooperativeAgent* et qui permet de représenter un agent interface, elle comporte les attributs ; *nbvariables*, *t_domaine*, *connecti*, *duret*, *nbpart*, *nombre_constraints* permet la génération d'un CSP. Elle comporte en outre les attributs ; *agentEspece* qu'est un tableau de type *AgentEspece*, *meilleurPart* qui est de type particule et *essaim* qu'est un tableau de particules. Cette classe encapsule les méthodes ; *initGUI* qui assure l'affichage de l'interface graphique *AgentInterfaceGui*, *GetParametre* permet de retourner les paramètres définis par l'utilisateur, *initialisation* assure la création et l'initialisation des différents agents espèces et *GetBest* permet de déterminer la meilleure solution globale.
- **AgentEspece** : est une classe qui hérite de la classe mère *CooperativeAgent* et qui permet de représenter un agent espèce, elle comporte les attributs ; *AdressInterface* qui représente l'adresse de l'agent interface, un tableau des agents espèces appelé *AgentEspece*, un vecteur de particule appelé *VectParticule*, *Nombergeneration* contient le nombre d'application de l'algorithme d'optimisation sur la sous-population des particules d'un agent espèce et *localbest* que de type particule et représente la meilleure solution locale. Les méthodes de cette classe sont ; *GetBest* permet de déterminer la meilleure solution locale, *DPSO* permet l'application de l'algorithme d'optimisation sur une sous-population de particule espèce et *GetFitness* détermine la fitness de chaque particule dans le sous-essaim d'un agent espèce.
- **ListeTuple** : permet de représenter un tuple, elle comprend comme des attributs ; *tuple* qui est un tableau de type *int* et représente un tuple de variables, *numVar* qui représente le numéro d'une variable dans un CSP, *numVal* qui représente le numéro d'une valeur d'une variable dans un CSP, *Dmin* et *Dmax* qui représente respectivement la valeur minimale et maximale qui peut prendre une variable.
- **Particule** : cette classe permet de représenter une particule, elle contient les attributs ; *valeur* qui définit la position d'une particule, *vitesse* qui contient la vitesse d'une particule, *template* qui représente la template d'une particule, *MeilleurSolutionLocale* et *MeilleurSolutionGlobale* qui représentent respectivement la meilleur solution locale et la meilleur solution globale pour une particule, *Fv* contient la valeur de fitness d'une particule (nombre des contraintes violées) , *MeilleurFv* contient la meilleur valeur de fitness et *Dmin* et *Dmax* représentant les deux bornes de l'intervalle de déplacement d'une particule dans l'espace de recherche. Les méthodes de cette classe sont ; *FonctionObjectif* permet de calcule la valeur de fitness d'une particule, *calcFitness* permet le calcul de fitness de toutes la population des particules, *meilleurValHost* permet de déterminer la meilleur particule dans l'essaim et *PSO* assure l'application de l'algorithme d'optimisation. .

Les interfaces `AgentInterfaceGui` et `InterfaceResultat` sont des classes comportant des composants graphiques et qui héritent respectivement de `JPanel` et `JFrame` .

Navigabilité :

- Un `AgentInterface` peut contenir plusieurs `ListeTuple`
- Une `ListeTuple` peut appartenir à un seul `AgentInterface`
- Un `AgentInterface` contient un et un seul `AgentInterfaceGui`
- Un `AgentInterfaceGui` comporte un à un seul `AgentInterface`
- Un `AgentInterface` peut contenir des `AgentEspece`
- Un `AgentEspece` appartient un et un seul `AgentInterface`
- Un `AgentInterface` peut utiliser de `Particule`
- Une `Particule` peut être utilisé par un et un seul `AgentEspece`
- Un `AgentEspece` peut contenir un `JavaConnection`
- Un `JavaConnection` peut être utilisé par un `AgentInterface`
- Un `AgentEspece` peut contenir plusieurs `ListeTuple`
- Une `ListeTuple` peut appartenir à un `AgentEspece`
- Un `AgentEspece` peut contenir plusieurs `Particule`
- Une `Particule` peut appartenir à un `AgentEspece`
- Un `InterfaceResultat` peut utiliser un `JavaConnection`

2.3 Conclusion

Dans ce chapitre, nous avons présenté le principe de base de l'approche, qui comporte les différentes techniques, heuristiques et les structures des agents utilisé. Puis nous avons présenté une conception de notre application destiné à l'optimisation d'un CSP grâce à l'optimisation par essaim des particulaires sous une architecture multi-agents. le chapitre suivant sera consacré à la réalisation de notre application et à l'analyse des résultats .

Chapitre 3

Réalisation et expérimentations

Introduction

Ce chapitre est la dernière étape du rapport, dans lequel nous mettons l'accent sur la réalisation de l'application. Nous commençons par décrire l'environnement de développement matériel et logiciel, nous présentons ensuite quelques captures d'écran de notre application. Puis nous présentons des résultats numériques relatifs à la résolution d'un CSP par notre algorithme. Enfin, ces résultats seront analysés et commentés.

3.1 Environnement de travail

3.1.1 Environnement matériel

Du point de vue matériel nous avons développé et expérimenté l'algorithme sur un pc portable ayant les caractéristiques suivantes :

- Intel core 2 Duo 2.10 GHz.
- 2 Go de mémoire vive.
- Windows 7 - 32 bits.

3.1.2 Environnement logiciel

La mise en place de notre application sous une architecture multi-agents nécessite l'utilisation de divers logiciels :

- Outils de développement et d'exécution : MadKit (Multi agents developpement KIT) [24]
- Outils de développement et de compilation : Plateforme IDE Eclipse [25]
- Un moteur de base des données : SQLite [26]

Ainsi pour la réduction de notre rapport, nous avons utilisé divers logiciels :

- Outil de modélisation : PowerAMC [27]
- Outil de formatage de texte : Latex[28]
- Outil ergonomique de dessin : Cacao[29]

3.1.2.1 Plate-forme multi-agents

a) Définition d'une plateforme multi-agents

Une plate-forme de développement des systèmes multi-agents est une infrastructure de logiciels utilisée comme environnement pour le déploiement et l'exécution d'un ensemble d'agents.

Elle devrait fournir des manières confortables les outils pour créer et tester des agents,

et elle peut être vue comme une collection de services offerts aux développeurs, mais également aux agents en exécution. Cependant, une plateforme est un environnement d'exécution pour un agent dans le sens qu'elle devrait permettre de créer, exécuter et supprimer des agents. D'autre part, une plateforme devrait agir en tant qu'un médiateur entre le système d'exploitation et les applications (agents) tournant dessus.

Le développeur pourra alors créer des agents pour une plateforme et les employer sur tous les systèmes qui supportent cette plateforme sans changer le code. De plus, une plateforme devrait cacher au développeur les détails d'implémentation des protocoles de communications. Si, par exemple, il y a plusieurs protocoles de communications disponibles sur le media de communication (par exemple HTTP, Bluetooth, etc.), la plateforme devrait choisir le protocole adéquat pour chaque situation. Ceci permettra de créer des agents indépendamment des protocoles de transmission disponibles sur la machine.

b) Exemple de Plates-formes multi-agent

Il existe un grand nombre de plateformes multi-agents dédiées à différents modèles d'agents. Puisque la plupart d'entre elles sont conçues, soit par rapport à un modèle d'agent particulier comme Zeus, soit par rapport à un domaine d'application particulier comme les agents mobiles Aglets, ou la simulation Swarm. Nous ne prétendons pas donner ici une liste exhaustive de l'ensemble des plateformes disponibles actuellement. Nous donnons une description succincte de quelques-unes choisies selon trois angles de vues différents :

- ❖ **Plates-formes pour simulation** : telles que Swarm, Cormas, StarLogo, Geamas, Mice...
- ❖ **Plates-formes pour implémentation** : comme Zeus, AgentBuilder, JACK, MadKit, DECAF, MAGIQUE, MACE, Mask, MoCAH, OSACA ...
- ❖ **Plates-formes pour mobilité** : Aglets, Concordia, Gossip, Grasshopper, Gypsy, JumpingBeams, GenA, KLAIM, Mole

3.1.2.2 Choix de la plate-forme multi-agents utilisée

Notre système Multi-agents a été implémenté sur la plateforme MadKit. Cette dernière constitue un puissant outil logiciel pour le développement de ce genre de système. Elle permet à des milliers d'agents de fonctionner tous indépendamment en parallèle. Permettant ainsi d'explorer la connexion entre le comportement de l'entité autonome au niveau micro, avec le comportement qui peut émerger de l'interaction entre ces entités au niveau macro.

La figure 3.1 illustre notre choix de la plate-forme MadKit. Où, pour chaque critère d'évaluation la plate-forme est noté comme suit :

- 4 si l'outil répond très bien au critère ;
- 3 si l'outil répond bien au critère ;

- 2 si l'outil répond moyennement au critère ;
- 1 si l'outil répond peu au critère ;
- 0 si l'outil ne répond pas du tout au critère.

Nous remarquons que la plate-forme Madkit nous permet d'avoir de bons résultats.

| Plate forme Critères | <i>JADE</i> | <i>DECAF</i> | <i>AgentBuilder</i> | <i>Zeus</i> | <i>JAFMAS/JIVE</i> | <i>Jack</i> | <i>AgentTool</i> | <i>Madkit</i> |
|-----------------------------|-------------|--------------|---------------------|-------------|--------------------|-------------|------------------|---------------|
| Méthodologie | 0 | 0 | 4 | 4 | 3 | 0 | 3 | 3 |
| Facilité d'apprentissage | 0 | 3 | 1 | 1 | 1 | 0 | 3 | 2 |
| Transition entre les étapes | 0 | 0 | 3 | 2 | 2 | 0 | 3 | 3 |
| Souplesse de l'outil | 3 | 0 | 1 | 1 | 2 | 3 | 0 | 3 |
| Communication inter-agents | 4 | 2 | 4 | 4 | 2 | 3 | 2 | 3 |
| Outil de « débuggage » | 3 | 2 | 4 | 4 | 1 | 0 | 2 | 4 |
| Support développement | 0 | 2 | 4 | 4 | 2 | 1 | 4 | 3 |
| Support implémentation | 0 | 0 | 4 | 4 | 2 | 1 | 2 | 3 |
| Gestion du SMA | 4 | 0 | 3 | 3 | 0 | 0 | 1 | 4 |
| Effort et simplicité | 2 | 3 | 2 | 2 | 1 | 2 | 3 | 2 |
| Bases de données | 0 | 0 | 1 | 2 | 0 | 3 | 0 | 0 |
| Génération de code | 0 | 0 | 1 | 3 | 1 | 0 | 1 | 2 |
| Extensibilité du code | 4 | 1 | 1 | 2 | 1 | 4 | 0 | 4 |
| Déploiement | 4 | 1 | 2 | 2 | 1 | 2 | 1 | 3 |
| Documentation disponible | 3 | 1 | 4 | 4 | 1 | 3 | 1 | 3 |
| | | | | | | | | |
| Total (sur 60) | 27 | 15 | 39 | 42 | 20 | 22 | 26 | 42 |

FIGURE 3.1 – Evaluation comparative de plate-formes multi-agents

3.1.2.3 Plateforme Madkit

- **Présentation**

MadKit est une plate-forme de développement de systèmes multi-agents [22] destinée au développement et à l'exécution de systèmes multi-agents et plus particulièrement à des systèmes multi-agents fondés sur des critères organisationnels AGR (Agent Group Role).

L'organisation AGR, développé dans le contexte du projet Aalaadan, propose que le système doive être composé de :

- **Agent** : qui est une entité informatique qui envoie et reçoit des messages, qui peuvent entrer et sortir des groupes et qui joue des rôles dans ces groupes. Cette entité est autonome et communicante.
- **Groupe** : qui est constitué des rôles et regroupe des agents qui ont soit un aspect en commun par exemple des agents qui possèdent le même langage de communication, soit un ensemble d'agents qui travaillent ensemble, qui collaborent à une même activité (des agents qui essaient de résoudre un problème).
- **Rôle** : qui correspond soit à un statut d'un agent, soit à une fonction particulière qu'un agent joue dans un groupe. Les rôles sont locaux aux groupes et n'ont de sens qu'à l'intérieur d'un groupe.

La figure 3.2 montre les concepts de base de modèle Aalaadin

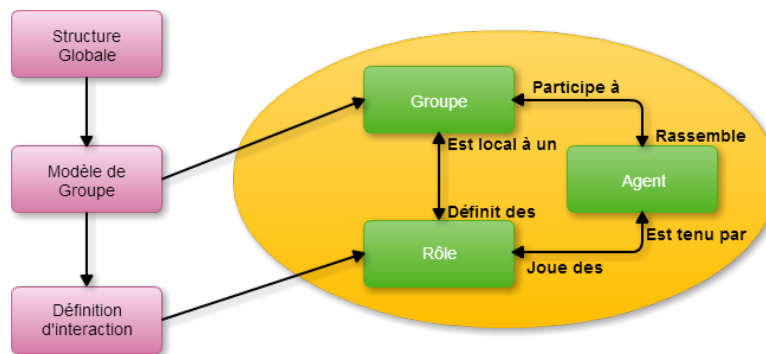


FIGURE 3.2 – Modèle de Madkit

• Caractéristiques

La plateforme MadKit est reconnue par les qualités suivantes :

- Simplicité de mise en œuvre et de prise en main.
- Pas de langage prédéfini de communication.
- Plusieurs types des messages peuvent être utilisés (*StringMessage*, *Act-Message*, *XMLMessage*, *ObjectMessage*).
- Possibilité d'avoir des vues (graphiques).
- Plate-forme distribuée open source (GPL/LGPL).
- Les agents peuvent être programmés à partir de plusieurs langages de programmation et des scripts (Java, Python, Scheme, Jess (langage de règles), BeanShell (interprète Java),...).
- Intégration très facile de la distribution au sein d'un réseau.
- L'aspect pratique et efficace des concepts organisationnels pour créer différents types d'applications.

- Hétérogénéité des applications et des types d'agents utilisables : on peut faire tourner sur MadKit aussi bien des applications utilisant des agents réactifs simples de type fournis plus de 50000 agents ayant un comportement simple ont tourné en MadKit, que des applications disposant d'agents cognitifs sophistiqués.
- Aussi MadKit est écrit en Java et fonctionne en mode distribué de manière transparente à partir d'une architecture "Peer to Peer" sans nécessiter de serveur dédié. Il est ainsi possible de faire communiquer des agents à distance sans avoir à se préoccuper des problèmes de communication qui sont gérés par la plateforme.
- MadKit est construit autour du concept de "micro-noyau" et "d'agentification de services". Le micro-noyau de MadKit est très petit moins de 100Ko de code, car il ne gère que les organisations (groupes et rôles) et les communications à l'intérieur de la plateforme. Le mécanisme de distribution, les outils de développement et de surveillance des agents, les outils de visualisation et de présentation sont des outils supplémentaires qui viennent sous la forme d'agents que l'on peut ajouter au noyau de base

La figure 3.3 présente l'architecture générale de notre plateforme :

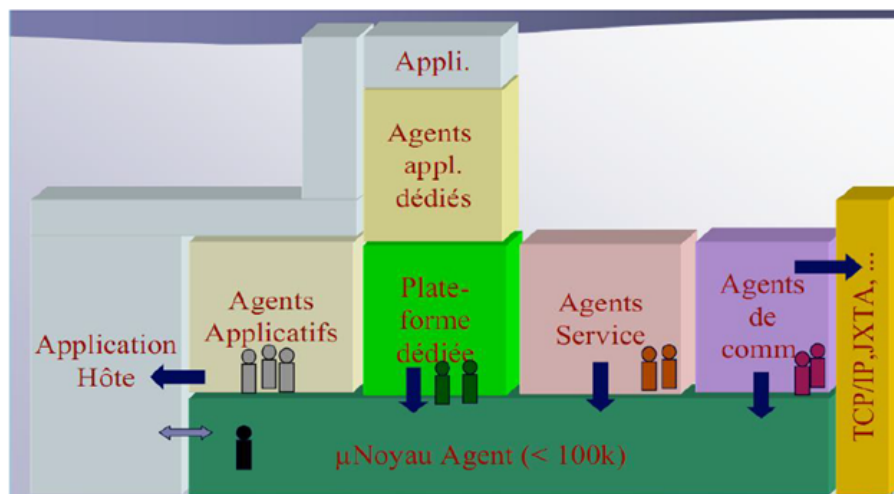


FIGURE 3.3 – Architecture générale de Madkit

3.2 Expérimentation et illustration

3.2.1 Description de processus d'expérimentation

3.2.1.1 Démarrage de l'application

Afin d'exécuter l'application l'utilisateur doit lancer la plateforme multi-agents MadKit. Puis, lancer l'agent Interface. Enfin, il doit remplir les champs nécessaires comme illustre la figure 3.4 :

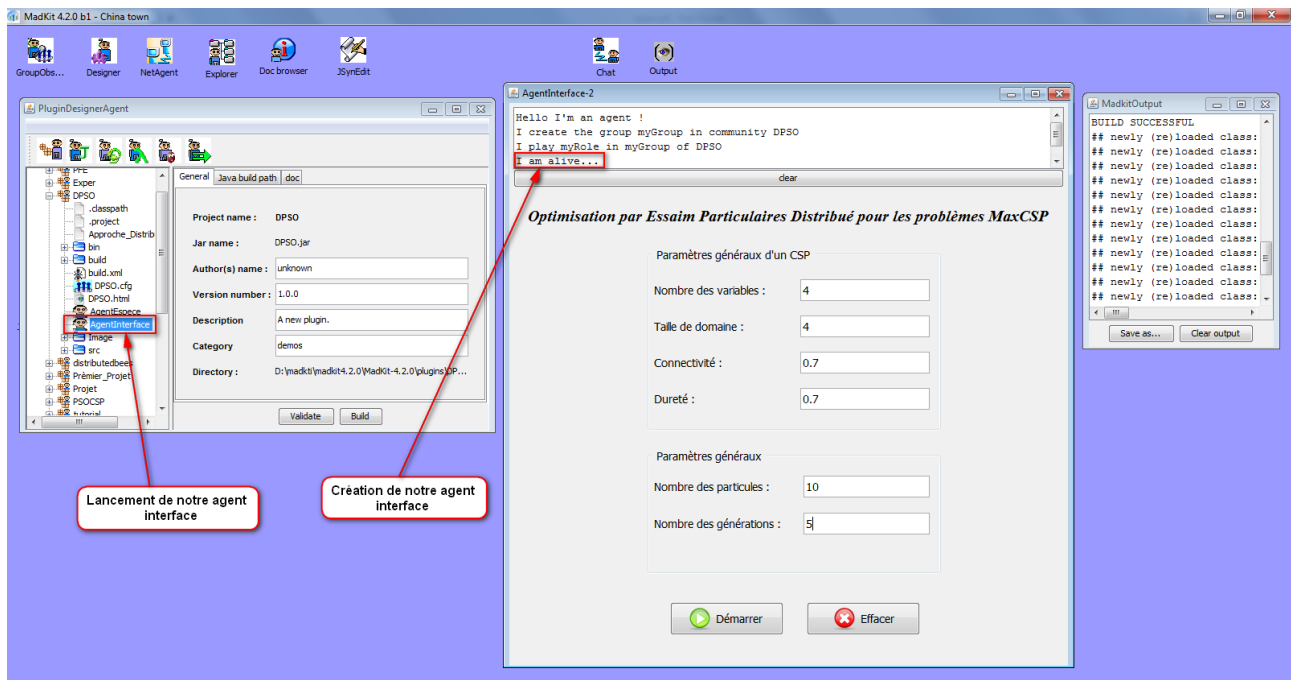


FIGURE 3.4 – Interface de l'application

3.2.1.2 Affichage du résultat

a) Génération d'un CSPs

Nous présentons ici un exemple d'exécution du générateur de CSP. L'exécution a généré 4 variables, dont le domaine D_i de chacune est de taille maximale 4, et 4 contraintes binaires représentées par l'ensemble des tuples autorisés.

La figure 3.5 illustre le CSP ainsi décrit :

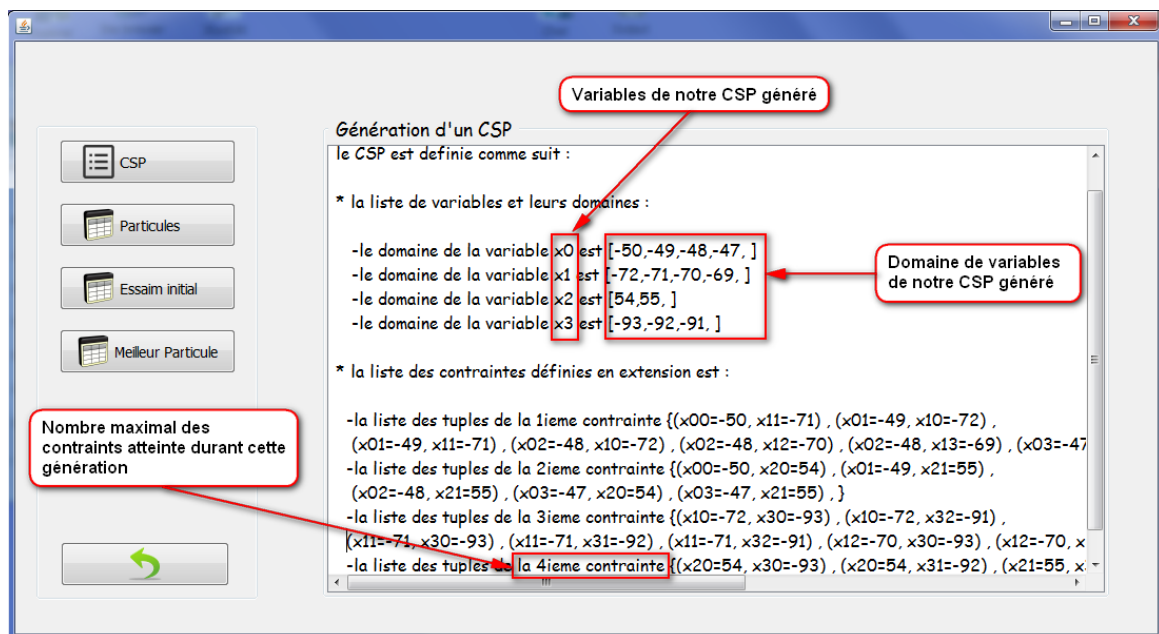


FIGURE 3.5 – Exemple d'un CSP généré

Notation :

les valeurs des variables figurant dans les contraintes sont noté x_{ij} ; i étant le numéro de la variable x_i , et j sa valeur du domaine D_i .

Le nombre d'instances initiales est choisi par l'utilisateur. Chaque instance est choisie aléatoirement à partir du CSP généré. Les instances ainsi déterminées représentent la population initiale pour l'algorithme d'optimisation par essais particuliers.

b) Génération d'une population initiale des particules

Après la génération d'un CSP, notre agent interface va mettre ce dernier sous forme exploitable pour générer une population initiale des particules, alors nous allons présenter ici cette dernière ainsi on choisit comme nombre des particules égalé à 10 pour notre essaim initial.

La figure 3.6 présente notre population initiale :

| Particule | Position | Vitesse | Dmin | Dmax |
|-----------|----------|-----------------------|------|------|
| 1 | -68 | -0.6075478659933846 | -70 | -68 |
| 1 | 14 | -0.1635958586460473 | 14 | 16 |
| 1 | 3 | 0.0 | 3 | 4 |
| 1 | -36 | 0.0 | -36 | -35 |
| 2 | -68 | 0.6742522100982782 | -70 | -68 |
| 2 | 14 | 0.10595206305133886 | 14 | 16 |
| 2 | 4 | 0.0 | 3 | 4 |
| 2 | -35 | 0.0 | -36 | -35 |
| 3 | -68 | -0.140813962895029... | -70 | -68 |

FIGURE 3.6 – Exemple de génération d'une population initiale

Notation :

une position d'une particule i est une instantiation complète des variables de générateur de CSP (dans notre cas on a pris 4 variables), c'est pour cela on remarque bien que sur l'interface de l'application, il apparait 4 valeurs dans la position du particule numéro 1 ainsi cette dernière possède 4 valeur de vitesse car chaque position X_i d'une particule i est associée à une vitesse V_i à un instant t .

c) Calcule de fitness du population initiale

Après avoir initialisé la population initiale, l'agent interface va calculer pour chaque particule sa fitness(**FV**) à partir de son template, sa meilleure solution locale (\mathbf{P}_{lbest}), sa meilleure solution globale(\mathbf{P}_{gbest}) et l'informe par la meilleure valeur de fitness dans toute

la population.

La figure 3.7 illustre notre essaim après la procédure de calcul de fitness :

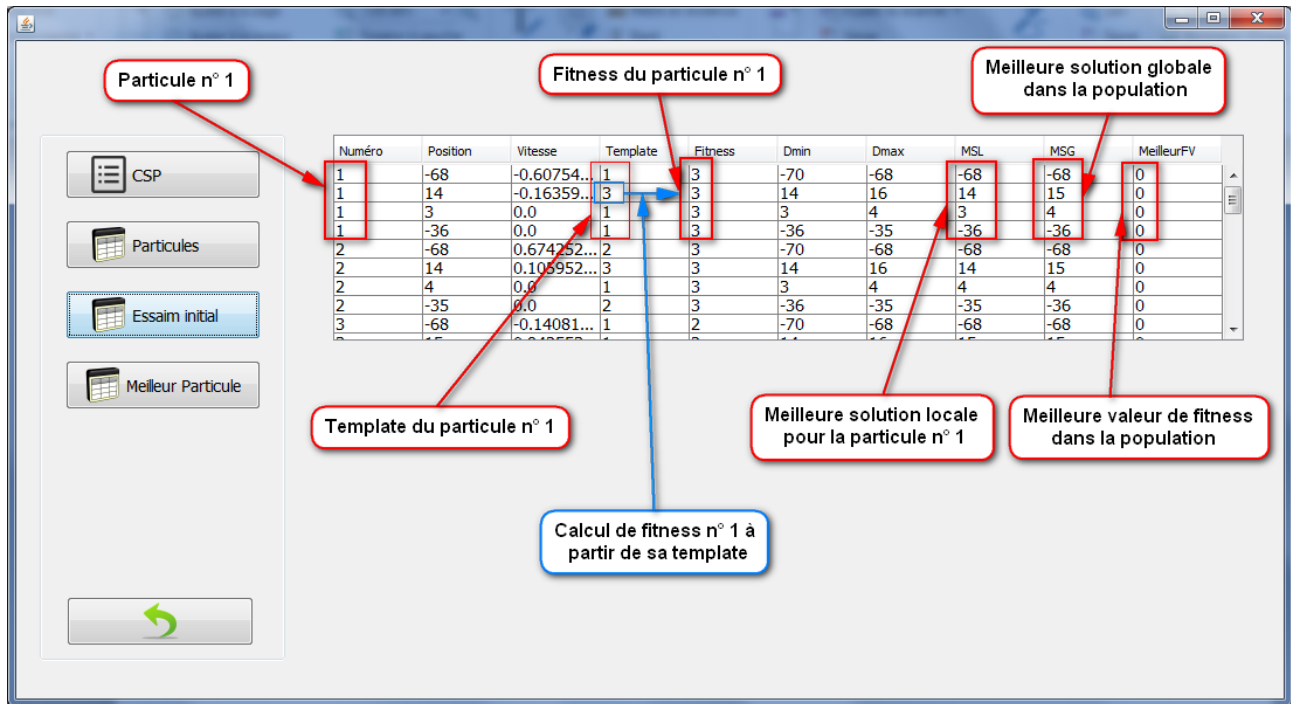


FIGURE 3.7 – Exemple de calcul de fitness du population initiale

Notation :

Après le calcul de **FV** de chaque particule, l'agent interface crée un agent Espèce pour chaque **FV**. Puis, il envoie à chaque agent les particules qu'il doit gérer et un message (**Apply Behaviour**) pour que l'agent espèce commence l'exécution de l'algorithme d'optimisation PSO.

d) Meilleure solution finale

Après la réception des particules par les agents espèces, chacun de ces derniers va appliquer l'algorithme d'optimisation jusqu'à ou la condition d'arrêt est atteinte, si cette dernière est vérifiée chaque agent espèce envoie dans un message sa meilleure particule. Puis l'agent interface détermine la P_{gbest} parmi les particules reçues.

La figure 3.8 présente notre P_{gbest} :

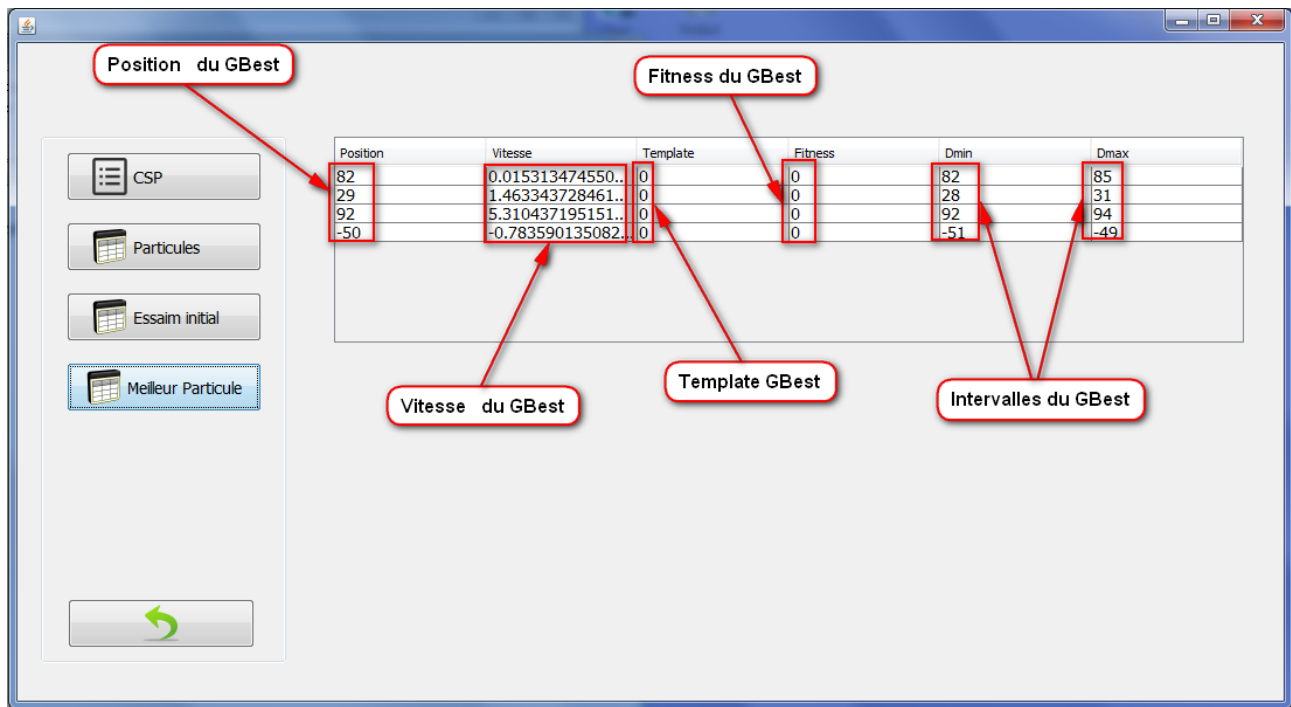


FIGURE 3.8 – Exemple de GBest

3.2.2 Mode expérimentale

L'idée est de présenter les résultats de test en appliquant les deux approches (CPSO et DPSO), en suite faire une comparaison entre ces deux approches :

- ❖ **CPSO** : il s'agit d'un algorithme d'optimisation par essaim particulaire centralisée guidée par l'utilisation de concept de template.
- ❖ **DPSO** : il s'agit d'une approche qui utilise une société d'agents créés dynamiquement, coopérant dans le but non seulement de fournir une solution optimale pour le Max-CSP, mais aussi pour réduire au minimum la complexité temporelle des PSOs. Chaque agent est responsable d'une sous-population des particules qu'il traite par un algorithme d'optimisation par essaim particulaire local guidé par le concept de template. Toutes les particules d'un agent espèce donné violent le même nombre de contraintes (ayant la même fitness).

Les performances de l'approche distribuée et centralisée sont dégagées :

- **Temps d'exécution** : temps CPU nécessaire pour résoudre une instance du problème
- **Satisfaction** : le nombre des contraintes satisfaites

La première performance montre la complexité temporelle tandis que le deuxième reflète la qualité de solution.

Dans les deux approches ; centralisée et distribuée, nous avons choisi de présenter trois

résultats obtenus en générant 12 CSPs pour chaque valeur de test fait.

Les paramètres utilisés dans les deux approches :

- Paramètres de générateurs de CSP
 - Nombre des variables = 7
 - Taille de domaine = 7
 - Connectivité = 0.5
- Paramètres de l'algorithme PSO
 - Nombre des particules = 20
 - Nombre des générations = 20
 - Inertie massique (*inertia weight*) $w = 0.5$
 - $c_1 = 2$
 - $c_2 = 2$
 - $r_1 = 0.3$
 - $r_2 = 0.6$

a) **Premier résultat : Présentation de temps d'exécution en fonction de la dureté**

Pour l'obtention de ce premier résultat, nous avons varié la valeur de la dureté de 0.1 jusqu'à 0.9, et calculer le temps d'exécution de deux algorithmes CPSO et DPSO, pour 12 CSPs générés pour chaque valeur de la dureté. Le temps enregistré pour une valeur de dureté déterminée, étant la moyenne des 30 CSPs générés

Les résultats des tests étant enregistrés dans le tableau 3.1 :

| Dureté | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|--------|--------|--------|--------|--------|
| Moyenne de 12 valeurs de temps d'exécution(ms) : CPSO | 39,577 | 39,733 | 41,185 | 42,408 | 44,173 |
| Moyenne de 12 valeurs de temps d'exécution(ms) : DPSO | 30,784 | 32,748 | 33,475 | 35,841 | 38,448 |

TABLE 3.1 – Variation de temps de CPU en fonction du dureté

Pour mieux observer les résultats ainsi obtenus, nous utilisons Excel pour les représenter dans un graphique. La figure 3.9 illustre ces résultats.

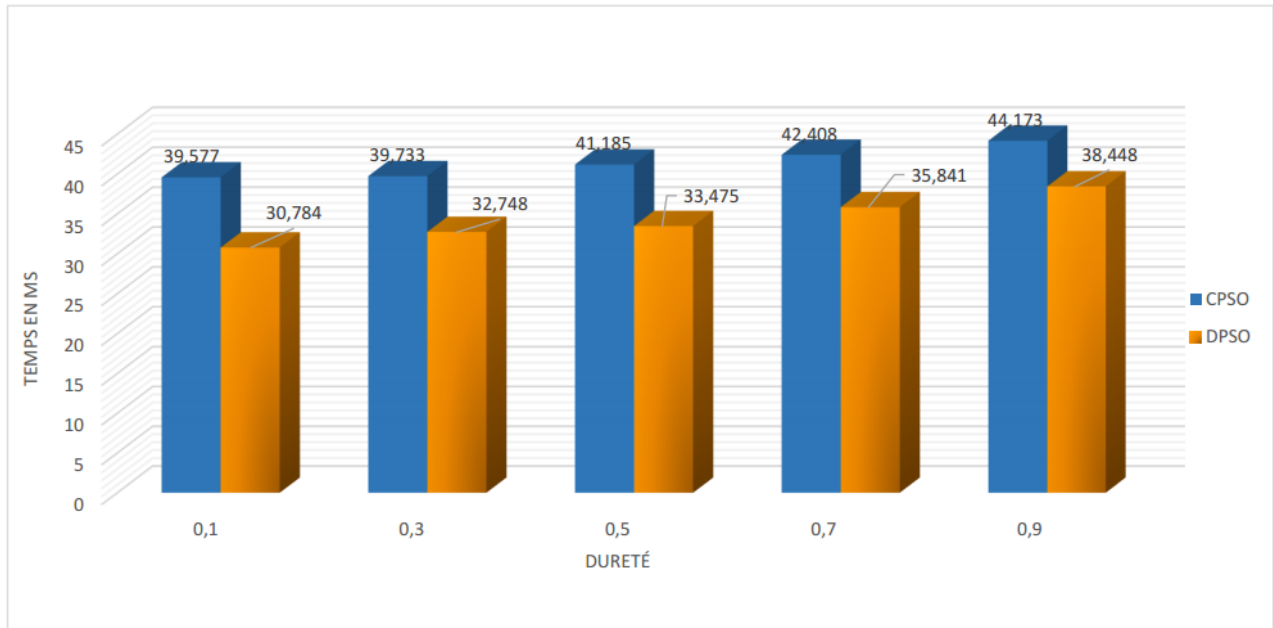


FIGURE 3.9 – Présentation du temps écoulé en fonction de la dureté

Nous remarquons qu'en faisant augmenter la dureté, le temps d'exécution augmente : plus les CSPs sont durs plus leur résolution devient complexe et demande plus de temps.

b) Deuxième résultat : Présentation du temps d'exécution en fonction du nombre de contraintes violées

Parmi les tests que nous pouvons faire pour évaluer les performances de l'algorithme PSO, nous calculons le temps d'exécution pour un nombre déterminé de contraintes violées : Pour une même valeur de la fonction objective (nombre de contraintes violées), nous calculons le temps écoulé pour atteindre cette valeur. Comme nous l'avons fait pour le résultat précédent, chaque valeur du temps obtenue est la moyenne de 12 essais de CSPs générés.

Le tableau 3.2 sauvegarde les résultats obtenus :

| Nombre de contraintes violées | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------------------------------------|--------|--------|--------|--------|--------|--------|--------|
| Moyenne de 12 valeurs de (ms) : CPSO | 39,290 | 36,841 | 33,214 | 28,842 | 24,457 | 19,751 | 16,758 |
| Moyenne de 12 valeurs de (ms) : DPSO | 31,178 | 29,673 | 25,478 | 17,147 | 13,747 | 10,451 | 8,291 |

TABLE 3.2 – Variation de temps de CPU en fonction des contraintes violées

Une représentation de ces résultats est illustrée dans la figure 3.10 :

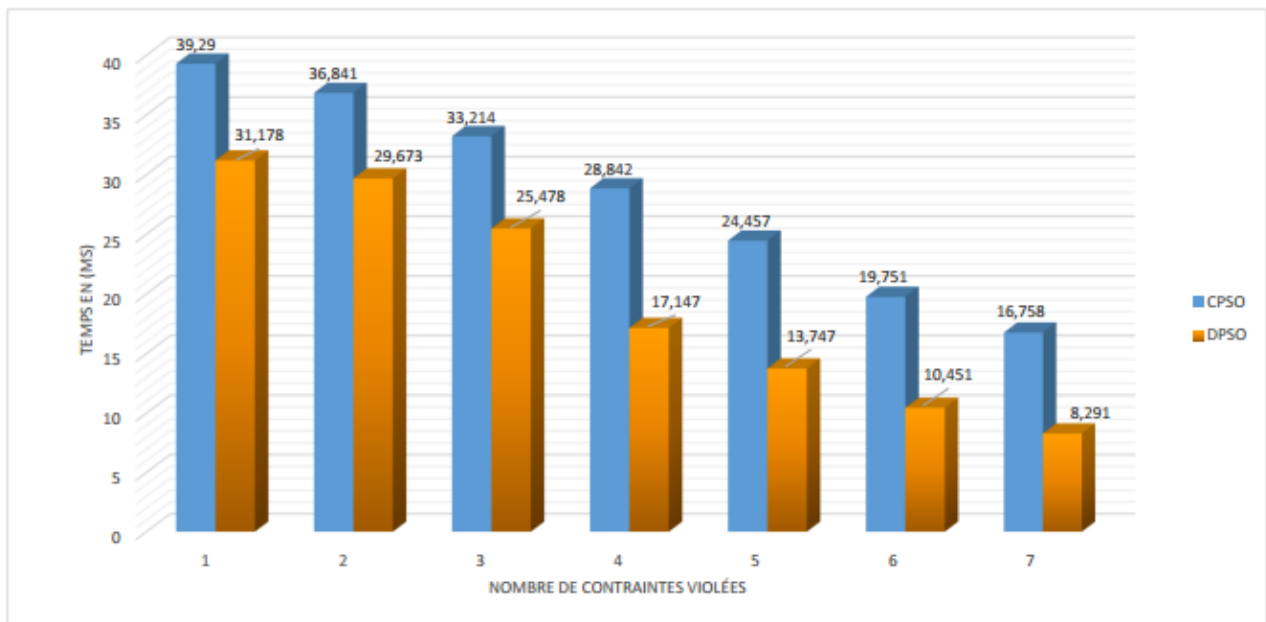


FIGURE 3.10 – Temps écoulé en fonction de la fitness(nombre des contraintes violées)

La figure 3.10 montre que la convergence vers la solution optimale (celle qui viole moins de contraintes), demande beaucoup plus de temps d'exécution.

c) **Troisième Résultat : présentation du nombre de contraintes satisfaites en fonction du durté :**

Pour l'obtention de ces résultats, nous avons varié la valeur de la durté de 0.1 jusqu'à 0.9 et calculer le moyen des contraintes satisfaites, pour 12 CSPs générés pour chaque valeur de la durté.

les résultats des tests étant enregistrés dans le tableau 3.3 :

| Durté | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|--|-------|-------|-------|--------|-------|
| Moyenne de 12 valeurs de nombre de contraintes satisfaites : CPSO | 6,583 | 7,75 | 8,583 | 9,833 | 10,08 |
| Moyenne de 12 valeurs de nombre de contraintes satisfaites) : DPSO | 7,358 | 8,712 | 9,957 | 10,933 | 11.04 |

TABLE 3.3 – Variation des contraintes satisfaites en fonction du durté

Une représentation de ces résultats est présentée par la figure 3.11

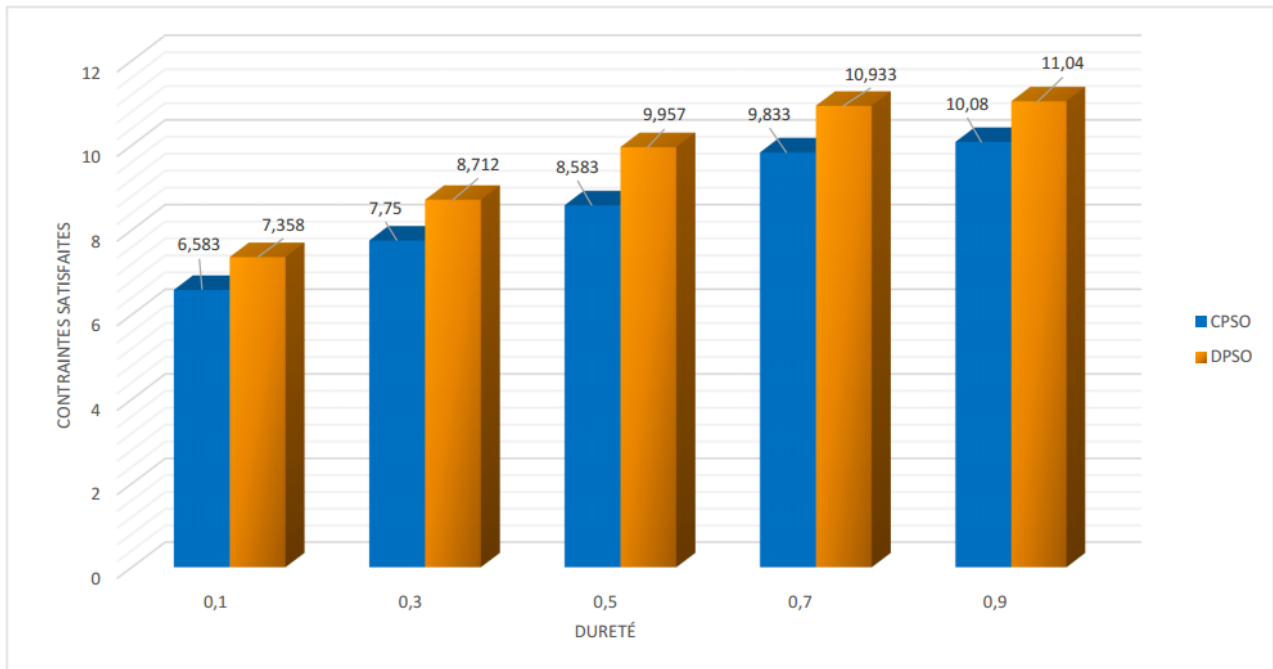


FIGURE 3.11 – Variation des contraintes satisfaites en fonction du dureté

Nous remarquons bien quand nous augmentons la dureté, le nombre de contraintes satisfaites augmente.

d) Comparaison de DPSO et CPSO

Pour avoir des comparaisons claires et rapides des performances relatives de deux approches, nous calculons les ratios des performances de deux algorithmes en utilisant le temps d'exécution et la satisfaction comme suit :

- $\text{Ratio-CPU} = \text{Temps exécution de CPSO} / \text{Temps d'exécution de DPOS}$
- $\text{Ratio-Satisfaction} = \text{Satisfaction de DPSO} / \text{Satisfaction de CPSO}$

Ainsi la performance pour DPSO est le dénominateur quand nous mesurons le ratio de temps CPU et le numérateur quand nous mesurons le ratio de satisfaction. Donc tout ratio supérieur ou égal à 1 indique la performance de DPSO.

De point de vue Ratio-CPU, DPSO a besoin de moins de temps pour les problèmes fortement durs. Pour le reste des problèmes, le ratio de temps de CPU est toujours supérieur à 1. En moyenne, ce ratio est égal à 1,211 ms. En effet cette performance est due à la communication et interaction entre les différents agents dans le but d'atteindre la meilleure solution dans une courte durée .

Tous ceci est résumé par le tableau 3.4 qui présente les différentes valeurs de ratio de Temps-CPU ainsi par la figure 3.12 :

| | | | | | |
|-----------------|-------|-------|-------|-------|-------|
| Dureté | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| Ratio Temps-CPU | 1,285 | 1,251 | 1,230 | 1,183 | 1,148 |

TABLE 3.4 – Ratio de Temps-CPU

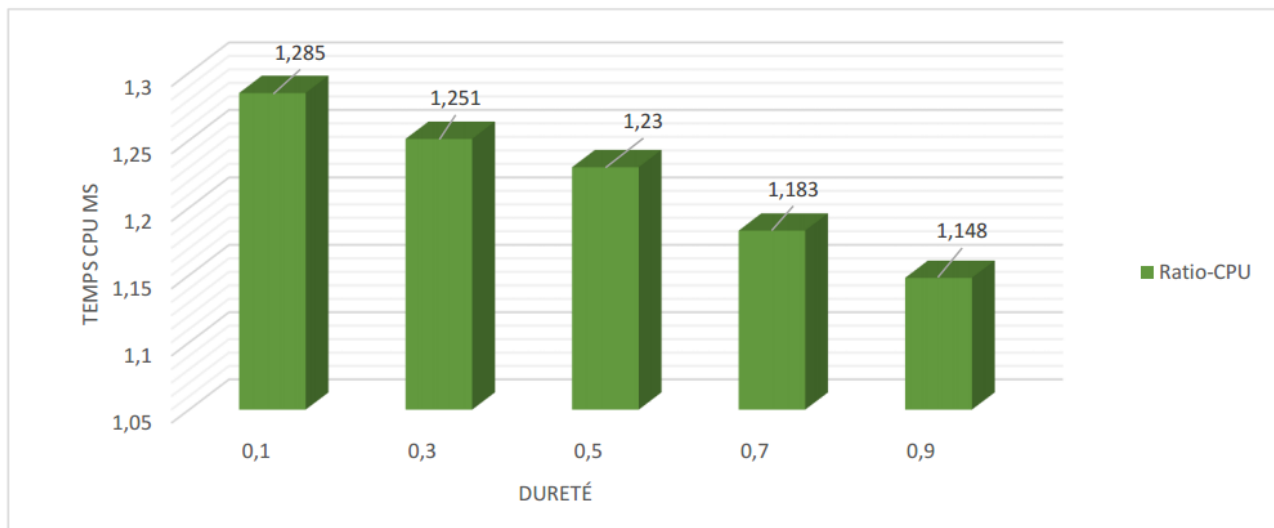


FIGURE 3.12 – Comparaison de DPSO et CPSO : Ratio-CPU

De point de vue Ratio-satisfaction, le DPSO converge toujours vers la bonne satisfaction. Il trouve environ 1,117 fois d'avantage pour les problèmes durs. La moyenne de ratio de satisfaction est environ 1,121. D'où notre approche permet un meilleur résultat dans un temps plus rapide.

Le tableau 3.5 et la figure 3.13 montrent la différence de Ratio-Satisfaction entre les deux approches

| | | | | | |
|--------------------|-------|-------|-------|-------|-------|
| Dureté | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| Ratio-Satisfaction | 1.117 | 1.123 | 1.160 | 1.111 | 1.095 |

TABLE 3.5 – Comparaison de DPSO et CPSO : Ratio-Satisfaction

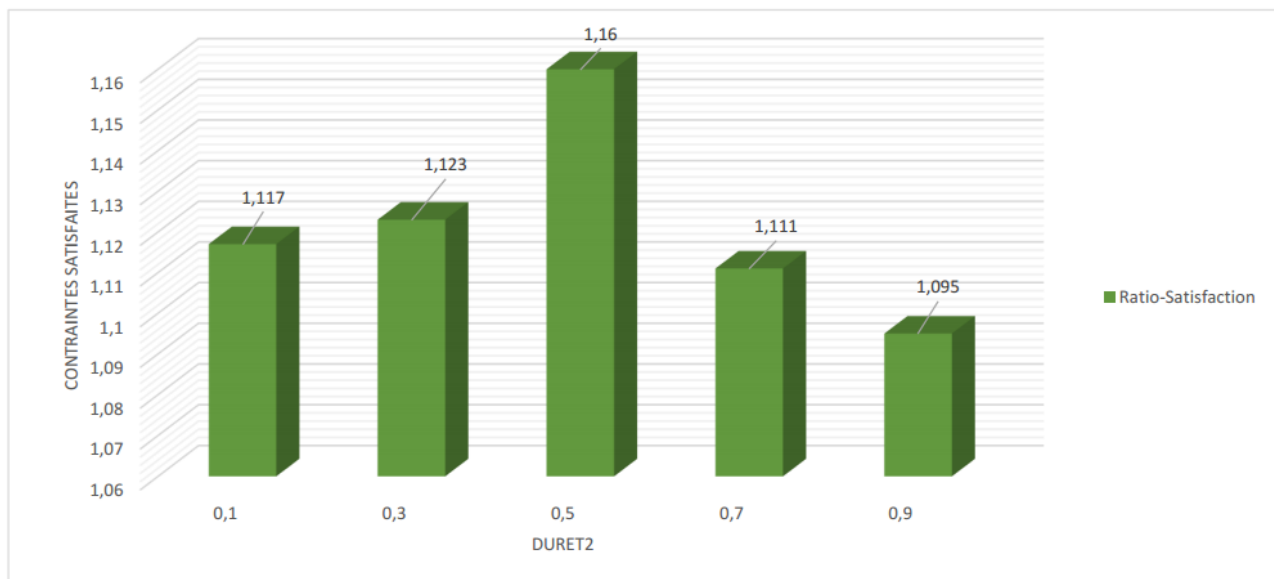


FIGURE 3.13 – Comparaison de DPSO et CPSO : Ratio-Satisfaction

On peut conclure à partir de ces résultats, que l'approche DPSO présente une énorme performance par rapport à l'approche CPSO, pour la résolution des problèmes Max-CSP, de point de vue gain de temps d'exécution et rapprochement de la solution optimale (qui viole moins de contraintes). En effet, c'est le résultat de l'interaction et la communication des agents dans le système multi-agent grâce à ces dernières la complexité temporelle est réduite et de meilleurs solutions sont obtenus.

3.3 Chronogramme

Le chronogramme représentatif de la totalité du travail réalisé est représenté sur la figure 3.14.

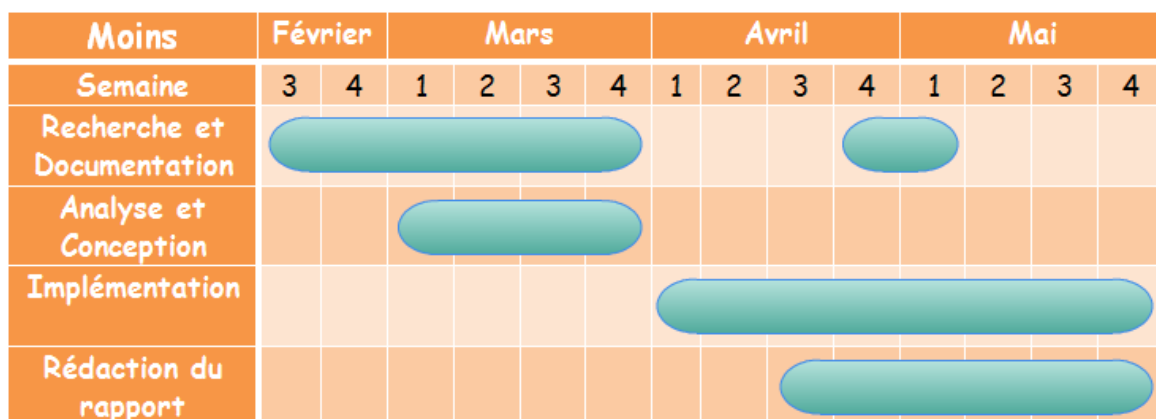


FIGURE 3.14 – Chronogramme du projet

3.4 Conclusion

Dans ce chapitre, nous avons présenté les environnements de travail. Puis, l'aperçu de l'application. Enfin, les différents résultats des expérimentations effectuées. A partir de ces expérimentation, nous avons pu montrer la performance et l'efficacité de notre approché proposée dans la résolution des problèmes Max-CSP par rapport à l'approche centralisée.

Conclusion générale et perspectives

Les problèmes de satisfaction de contraintes font l'objet de recherches intenses à la fois en intelligence artificielle et en recherche opérationnelle. Malheureusement, ces problèmes sont NP-difficiles. C'est pourquoi, de nouvelles méthodes sont inclinées pour leur résolution. Parmi ces méthodes nous citons les métaheuristiques telles que l'algorithme des essaims particulaires.

Cependant, même avec les algorithmes PSO centralisés la résolution des CSPs aboutit à une explosion combinatoire dans le cas des CSPs durs.

De ce fait, vient l'intérêt de notre projet de fin d'études. Nous avons implémenté l'algorithme PSO distribué pour améliorer l'efficacité et la robustesse des problèmes CSPs, grâce à l'utilisation d'une architecture multi-agents.

Dans ce cadre, nous avons utilisé deux types d'agents : un agent Interface et un agent Espèce. Chaque agent Espèce est responsable d'un sous-ensemble des particules, sur laquelle il applique PSO par une nouvelle structure de données, dite " Template ".

Nous avons commencé le travail par une étude bibliographique, qui nous a permis de nous situer par rapport à la littérature. En deuxième lieu nous avons donné un aperçu sur l'approche proposée et une conception de l'application à développer. Enfin, nous avons expérimenté et comparé notre approche par rapport à l'approche centralisée.

Nous prévoyons comme perspectives de ce travail, l'amélioration de l'approche distribuée DPSO par l'utilisation de la notion de probabilité de guidage au niveau du calcul de la vitesse des particules. Ensuite, l'application de DPSO sur des problèmes réels comme le problème d'allocation des fréquences, problème de tournée des véhicules... Étendre l'approche pour résoudre des autres extensions de CSP comme les fuzzy-CSPs, les dynamic CSPs. L'ajout des autres mécanismes pour rendre l'approche plus efficace.

Bibliographie

- [1] Michalewicz, Z.Slawomir et Zbigniew, Evolutionary algorithms,homomorphous mappings, and constrained parameter optimization,Evolutionary Computation, 2004, 19-44 .
- [2] F.Grimaccia M.Mussetta R,E.Ziche, Alfassia Gradldi et A.Grandelli, New hybrid technique for the optimization of large-domaine,IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, 2007, 781-785
- [3] J.Ferber, Multi-Agent System : An Introduction to Distributed Artificial Intelligence,InterEditions,Paris ,1995 .
- [4] K.Ghedira et G.Verfaillie, A multi-agent model for the ressource allocation problem : a reactive approch, The Scheduling of production process journal, Vienna University, 1995
- [5] K.Ghedira, S.Bouamama , A Dynamic . Distributed Double Guided Genetic Algorithm for Optimization and Constraint Reasoning, International Journal of Computational Intelligence Research, 2006, 181-190 .
- [6] Steiglitz, H. Papadimitriou et Kenneth, Combinatorial Optimization : Algorithms and Complexity, Courier Corporation, 2000.
- [7] S.Bouamama, k.Ghedraik et N.Zaeir, Load Balancing for the Dynamic Distribueted Double guided Genetic Algorithm for Max-CSP,Intenational Journal of Articial life, 2010.
- [8] J.Chia-Feng , A hybrid ofgenetic algorithm and particule Swarm Optimization for recurrent network design,IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, 2004
- [9] Rao, V.Kumar et V. Nageshwara, Parallel depth first search,International Journal of Parallel Programming, 2007, 501-509 .
- [10] Lin, V.Kumar et Yow-Jian, A data-dependency-based intelligent backtracking scheme for PROLOG, The Journal of Logic Programming, 2003, 165–181 .
- [11] N.Jussien, R.Debruyne et P.Boizumault, Maintaining Arc-Consistency within Dynamic Backtracking, 2000 .
- [12] E. Borrett, P.Edward et K. Tsang , Adaptive constraint satisfaction : the quickest first principle, Wiley University, 2005 .

- [13] S. KIRKPATRICK, C.GELATT et M. VECCHI, Optimization by simulated annealing, Science, 1993, 671-680.
- [14] Baptiste et Autin, Les métaheuristiques en optimisation combinatoire, Conservatoire National Des Arts et Metiers-PARIS, 2006 .
- [15] Glover et Fred, Future paths for integer programming and links to artificial intelligence, Computers et Operations Research, 1986, 533-549 .
- [16] Laguna et F.Glover, Tabu Search, Springer US, 1999 .
- [17] Maurice Clerc et Patrick Siarry, Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particuliers, France Télécom R et D, 2004 .
- [18] Stan Franklin et Art Graesser, Is It an agent, or just a program? : A taxonomy for autonomous agents, Springer Berlin Heidelberg, 2005 .
- [19] John Bigham, Autonomous Agents and Multi-Agent Systems, A Roadmap of Agent Research and Development, 1998 .
- [20] Moes et Pettie, Communications of the ACM, Artificial life meets entertainment : lifelike autonomous agents, 1997 .
- [21] Demazeau et Costa, Populations and organisations in open multi-agent systems, In 1st Symposium on Parallel and Distributed AI ,1998 .
- [22] Chaib-draa et Imed Jarras Brahim, Aperçu sur les systèmes multiagents, CIRANO, 2002 .
- [23] S.Bouaman, A new Distributed Particule Swarm Optimisation for constraint Reasoning, International Conference on Knowledge based and Intelligent Information et Engineering System, 2010.
- [24] Madkit Platfrom : [http ://www.madkit.net/madkit/](http://www.madkit.net/madkit/).
- [25] [http ://www.eclipse.org/](http://www.eclipse.org/).
- [26] [http ://www.sqlite.org/](http://www.sqlite.org/).
- [27] [http ://telecharger.logiciel.net/poweramc/](http://telecharger.logiciel.net/poweramc/).
- [28] [http ://www.xmlmath.net/texmaker/download_fr.html](http://www.xmlmath.net/texmaker/download_fr.html).
- [29] [https ://cacao.com/diagrams/](https://cacao.com/diagrams/)

Annexes

Les classes principales de noyau Madkit

La classe `AbstractAgent` est la super-classe de tout agent créée sous Madkit. Sous Madkit, un agent possède un cycle de vie composé de trois étapes : activation, vie et mort. La classe `AbstractAgent` contient trois méthodes permettant de gérer ce cycle de vie :

- La méthode `activate()` contient les instructions à effectuer lorsque l'agent est activé (via une autre méthode : `launchAgent()`). Cette méthode permet, par exemple d'initialiser les données qui possède l'agent, de lui faire des requêtes d'admission dans un groupe,...
- La méthode `live()` contient le bloc d'instruction correspondant à la vie d'un agent. Un exemple de contenu d'une méthode `live()` peut être une boucle infinie dans laquelle l'agent répète les instructions : attente d'un message, traitement du message et réponse au message.
- La méthode `end()` contient le bloc d'instructions à exécuter lors de l'arrêt de l'agent. Une méthode `end()` peut, par exemple, contenir les instructions nécessaires à la sauvegarde des données que possède l'agent.

La classe `AbstractAgent` permet également de gérer le modèle relationnel Agent/Group/Rôle. Pour cela, elle définit un ensemble de méthodes qui permettent à un agent de rejoindre un groupe, puis d'y demander un rôle particulier. On peut citer les méthodes :

- `createGroup()`, `createGroupIfAbsent()` permettant de créer un groupe.
- `requestRole()` permettant de demander un rôle au sein d'un groupe particulier, dont le nom est passé en paramètre.

Enfin, puisque les agents agissent au sein d'un modèle relationnel, il est nécessaire qu'ils puissent communiquer entre eux. Pour permettre cette communication, deux classes entrent en jeu : `AgentAdress` et `Message`.

La classe `AgentAdress` permet de représenter l'adresse d'un agent, c'est-à-dire l'endroit vers où doivent être expédiés les messages destinés à cet agent. Point important, un agent peut posséder plusieurs adresses. En effet, l'`AgentAdress` représente le triplet $\langle \text{communauté}, \text{groupe}, \text{rôle} \rangle$. Comme un agent peut occuper différents rôles au sein d'un ou plusieurs groupes et communautés, il est nécessaire qu'il possède une adresse pour chacun des rôles qu'il tient. Dans le but de simplifier la communication entre agents conçus par différents développeurs, la classe `Message<T>` définit la structure d'un message :

- Comme l'indique le type paramétrisé "*T*", la classe *Message* est une classe générique, il est donc possible pour un agent d'envoyer comme message une instance de n'importe quelle classe, par exemple un *Integer*, une *String*, mais aussi n'importe quelle classe définie par le programmeur.
- La classe *Message* définit également les deux méthodes suivantes : *getReceiver()* et *getSender()*. Comme leurs noms l'indiquent, ces deux méthodes permettent d'obtenir l'adresse de l'expéditeur et du destinataire d'un message, cette adresse étant représentée par une instance de la classe *AgentAdress*.

L'envoi des messages sous Madkit est asynchrone, ce qui signifie que l'expéditeur envoie un message au destinataire, mais ne se bloque pas en attendant la réponse.