UNIVERSITÉ SAAD DAHLEB DE BLIDA

Faculté des Sciences de l'Ingénieur

Département d'Electronique

THÈSE DE DOCTORAT D'ÉTAT

Spécialité : Contrôle

API, UNE METAHEURISTIQUE POUR L'OPTIMISATION DIFFICILE. APPLICATION A L'IDENTIFICATION DE LA MACHINE ASYNCHRONE

Par

M'hamed HADJ SADOK

Devant le jury composé de :

M. BENSEBTI	Professeur, U.S.D de Blida	Président
F. BOUDJEMA	Professeur, E.N.P d'Alger	Examinateur
B. BOUZOUIA	Directeur de Recherche, C.D.T Alger	Examinateur
D. BOUKHETALA	Maître de Conférence, E.N.P d'Alger	Examinateur
M. BOUNEKHLA	Maître de Conférence, U.S.D de Blida	Examinateur
A. GUESSOUM	Professeur, U.S.D de Blida	Rapporteur

Blida, Décembre 2007

REMERCIEMENTS

Qu'il me soit d'abord permis de remercier et d'exprimer ma gratitude envers le bon dieu, qui m'a donné la patience et le courage pour que je puisse terminer ce travail.

Je remercie vivement Monsieur **Messaoud Bensebti**, Professeur au département d'Electronique de l'Université de Blida, pour m'avoir fait l'honneur de présider mon jury.

Je tiens également à remercier Messieurs **Boudjema Farés** et **Boukhetala Djamel** respectivement, Professeur et Maître de Conférence à E.N.P d'Alger, ainsi que Messieurs **Bouzouia Brahim**, Directeur de recherche au C.D.T d'Alger et **Bounekhla M'hamed**, Maître de conférence à l'université de Blida, pour l'intérêt qu'ils ont porté à mon travail, pour les enrichissantes observations faites dans leurs rapports, ainsi que pour m'avoir fait l'honneur d'être membres de mon jury.

Je tiens ensuite à remercier Monsieur **Abderrezak Guessoum**, Professeur au département d'Electronique de l'Université de Blida, pour avoir dirigé ce travail ainsi que pour ses nombreux conseils et son soutien tout au long de cette thèse.

Je ne veux pas oublier, dans mes remerciements, Monsieur **Salhi Hassen**, Maître de Conférence et Directeur du laboratoire de recherche **S.E.T** du Département d'Electronique de l'Université de Blida. Les conseils avisés qu'il m'a prodigué témoignent de ses qualités scientifiques et humaines.

Je ne peux pas oublier, mes amis et collègues du département d'Electronique, **Ferdjouni Abdelaziz**, **Bounekhla M'hamed**, **Chikhi Lazhar** et **Nedjmi Omar** avec qui j'ai passé d'excellents moments et dont les conseils m'ont beaucoup aidé.

Enfin je remercie toute ma famille pour l'affection, la patience et le soutien inconditionnel tout au long de ces longues années d'études.

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

TABLE DES MATIERES

INTRODUCTION

Ces dernières années, les algorithmes à base de population de fourmis ont connus des succès croissants auprès des scientifiques spécialisés en informatique et en recherche opérationnelle [1,2]. On peut trouver, aujourd'hui, une multitude d'applications de ces algorithmes dans différents domaines tel que la robotique, la classification des objets, les réseaux de communication et l'optimisation combinatoire [3,4,5,6,7].

Les implémentations les plus réussies utilisant ces types d'algorithmes sont ceux pour l'optimisation combinatoire et pour le routage des réseaux. Ces réalisations partagent une structure commune qui a permis la définition d'une nouvelle métaheuristique appelée ACO, pour "Ant Colony Optimization " [8,9]. Dans ces algorithmes ACO, le problème est représenté par un jeu de solutions, une fonction objective et un jeu de contraintes. L'objectif étant de trouver l'optimum global satisfaisant les contraintes.

Ces algorithmes sont généralement utilisés comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans la procédure employée et la majorité d'entre eux s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromone) et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

Cette métaheuristique, d'origine discrète, a été élaborée par les informaticiens pour la résolution des problèmes d'optimisation combinatoires [9,10,11,12,13,14]. Cependant, il existe, en ingénierie, une large gamme de problèmes où la fonction objectif est à variables continues et pour lesquelles, la métaheuristique peut être d'un grand secours (fonction non dérivable, multiples minimums locaux, non convexe, nombre de variables important,..). Dés lors, l'adaptation de ces algorithmes, aux cas continus devient une priorité et plusieurs tentatives "d'adaptation" sont apparues [15,16,17,18].

En plus des problèmes classiques liés au perfectionnement de l'adaptation d'une métaheuristique, les algorithmes de colonies de fourmis posent un certains nombre de

problèmes reliés à la construction des solutions (composant par composant), ce qui nous fait dire que les algorithmes produits n'ont pas atteint leur pleine maturité.

Dans ce travail, on s'intéresse à l'algorithme de colonie de fourmis API, déduit de la modélisation du comportement de fourragement de l'espèce *Pachycondyla* API*calis* et son application au problème général de l'optimisation (discret et continu) [19,20,21,22]. L'intérêt porté à cette espèce de fourmis dans le domaine de l'optimisation provient du fait que ceux ci utilisent des principes relativement simples à la fois d'un point de vue global et local pour rechercher leurs proies [23]. A partir de leur nid, les fourmis couvrent globalement une surface donnée en la partitionnant en plusieurs sites de chasse individuels. Chaque fourmi effectue une exploration aléatoire locale de ses sites de chasse et le site choisi est sensible au succès précédemment rencontré sur les autres sites. Cela correspond en optimisation à un algorithme effectuant plusieurs recherches aléatoires en parallèle et localisées uniformément dans un sous-espace centré en un point (le nid).

Le déplacement du nid correspond à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé. Ces principes peuvent être repris pour la recherche d'un minimum global d'une fonction f dans un espace de recherche S.

Le premier chapitre, décrit d'abord le cadre de l'optimisation « difficile » et des différents aspects des problèmes qui peuvent se présentés à l'ingénieur, puis présente un état de l'art sur les métaheuristiques dites « de colonies de fourmis » en mettant en évidence le fait que ces algorithmes ont été élaborés pour la résolution des problèmes combinatoires.

Le chapitre deux, présente une étude sur l'adaptation de ces algorithmes de colonies de fourmis aux problèmes (les plus souvent rencontrés en ingénierie) où la fonction objectif est à variables continues. Les tentatives d'adaptation de la métaheuristique, les plus connus dans le domaine, sont recensées et étudiées en mettant en avant les difficultés émergentes.

Le chapitre trois est consacré à l'étude de l'algorithme de fourmis *Pachycondyla* API*calis* (API) et son application au problème général d'optimisation. Cet algorithme a retenu notre attention, du fait qu'il soit adaptable au cas continu et discret et il ne fait pas usage de la communication indirecte par pistes de phéromone.

Dans le chapitre quatre, nous mettons en œuvre l'algorithme de colonie de fourmis API pour déterminer simultanément les paramètres électriques et mécaniques d'un moteur asynchrone triphasé à partir du courant de démarrage et de la tension simple

correspondante. Nous choisissons la méthode du modèle de référence (erreur de sortie) comme technique d'identification.

D'une technologie simple, la machine asynchrone, ou moteur à induction, est largement utilisée dans le domaine des grandes puissances, sa robustesse et son coût d'achat et d'entretien lui ont permis de conquérir un espace de plus en plus grand au détriments des machines synchrones et à courant continu. Par ailleurs, les progrès réalisés en matière de commande et les développements technologiques, tant dans le domaine de l'électronique de puissance que celui de la micro électronique, ont rendu possible l'usage de commandes performantes faisant de la machine asynchrone un concurrent potentiel dans les domaines de la vitesse variable. La mise au point de commandes performantes requière une bonne modélisation et une bonne identification de la machine asynchrone. Dans les installations de grandes puissances où les essais directs sont coûteux et difficilement réalisables, la simulation s'impose comme une bonne alternative pour la prédiction des caractéristiques de la machine. Ces caractéristiques peuvent servir à un dimensionnement judicieux des différents éléments constituant le système globale ou à la mise en œuvre d'un système de diagnostique et de surveillance. Dans toutes ces situations, la modélisation et l'identification de la machine ont un impact non négligeable sur la précision des résultats obtenus.

On commence par écrire les équations régissant le fonctionnement de la machine asynchrone triphasée, puis on déduit le modèle du moteur à cinq puis à quatre paramètres électriques, avant de rappeler succinctement les différentes techniques d'identification pour opter finalement pour la méthode du modèle de référence.

Les paramètres de la machine sont déterminés par minimisation de l'erreur quadratique entre les courants mesurés et ceux calculés à l'aide des paramètres estimés par notre algorithme. Nous utiliserons d'abord des données simulées, puis des données expérimentales relevées sur un moteur réel, pour valider et étudier l'efficacité de la procédure implémentée.

La conclusion récapitule, enfin nos contributions dans le domaine de l'identification paramétrique de la machine asynchrone et propose quelques perspectives sur des améliorations possibles de l'algorithme de fourmis API, dans le but d'augmenter son efficacité.

CHAPITRE 1
ALGORITHMES DE COLONIES DE FOURMIS EN OPTIMISATION : ETAT DE L'ART

1.1. Introduction

Les ingénieurs et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs techniques très divers, comme la recherche opérationnelle, la conception de systèmes mécaniques, le traitement des images, et tout particulièrement en électronique (C.A.O. de circuits électriques, placement et routage de composants, amélioration des performances ou du rendement des systèmes, caractérisation de schémas équivalents, apprentissages des bases de connaissances des systèmes flous ou de réseaux de neurones…). Le problème à résoudre peut souvent s'exprimer comme un *problème d'optimisation* : on définit une fonction objectif, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser ou à maximiser par rapport aux paramètres concernés. La définition du problème d'optimisation est souvent complétée par la donnée de *contraintes* : tous les paramètres des solutions retenues doivent respecter ces contraintes, faute de quoi ces solutions ne sont pas réalisables. Nous nous intéressons dans ce travail à la classe de *métaheuristiques* dites de « colonies de fourmis » et à son application aux problèmes d'optimisation difficile.

Ce chapitre décrit tout d'abord le cadre de *l'optimisation difficile* et des métaheuristiques dans lequel nous nous plaçons dans ce travail, puis présente un état de l'art sur les métaheuristiques de « colonies de fourmis ».

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour les problèmes d'optimisation difficile. Ces algorithmes s'efforcent de simuler la capacité collective de résolution de certains problèmes, observée chez une colonie de fourmis, dont les membres sont pourtant individuellement dotés de facultés très limitées. Apparues sur terre il y'a quelques centaines de millions d'années, les fourmis sont en effet l'une des espèces les plus prospère et c'est principalement, ce succès qui a provoqué un nombre considérables de travaux sur l'espèce. En particulier, les

entomologistes ont analysé la collaboration qui s'établi entre les fourmis lors de la recherche de nourriture à l'extérieur de la fourmilière, en effet, il est remarquable que les fourmis suivent toujours le même chemin, et que ce chemin soit le plus court possible. Cette conduite est le résultat d'un mode de communication indirecte, via l'environnement : la *stigmergie*. Chaque fourmis dépose, le long de son chemin, une substance chimique, dénommée *phéromone* et tous les membres de la colonie perçoivent cette substance puis orientent préférentiellement leur marche vers les régions les plus odorantes. Il en résulte notamment la faculté collective de retrouver le plus court chemin par la colonie de fourmis.

1.2. Métaheuristiques pour l' "optimisation difficile"

1.2.1. Optimisation "difficile"

1.2.1.1. Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de solutions possibles *S,* dont la qualité peut être décrite par une fonction objectif *f.* On cherche alors à trouver la solution s^* possédant la meilleure qualité $f(s^*)$ (par la suite, on peut chercher à minimiser ou à maximiser $f(s)$). Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur s, être dynamique si $f(s)$ change avec le temps ou encore multi objectif si plusieurs fonctions objectifs doivent être optimisées.

Il existe des méthodes déterministes (dites « exactes ») permettant de résoudre certains problèmes en un temps fini. Ces méthodes nécessitent généralement un certain nombre de caractéristiques de la fonction objectif, comme la stricte *convexité,* la *continuité* ou encore la *dérivabilité*. On peut citer comme exemples de méthodes, la programmation linéaire, quadratique ou dynamique, la méthode du gradient, la méthode de Newton, etc.

1.2.1.2. Optimisation difficile

Certains problèmes d'optimisation demeurent cependant hors de portée des méthodes exactes. Un certain nombre de caractéristiques peuvent en effet être problématiques, comme l'absence de *convexité stricte* (multimodalité), l'existence de *discontinuités*, une fonction *non dérivable,* présence de bruit, etc.

Dans de tels cas, le problème d'optimisation est dit "*difficile"*, car aucune méthode exacte n'est capable de le résoudre exactement en un temps raisonnable, on devra alors faire appel à des *heuristiques* permettant une optimisation approchée.

L'optimisation difficile peut se découper en deux types de problèmes : les problèmes *discrets* et les problèmes à variables *continus*. Pour fixer les idées, citons deux exemples. Parmi les problèmes discrets, on trouve le célèbre problème du voyageur de commerce : il s'agit de minimiser la longueur de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. Un exemple classique de problème continu est celui de la recherche des valeurs à affecter aux paramètres d'un modèle numérique de processus, pour que ce modèle reproduise au mieux le comportement réel observé. En pratique, on peut rencontrer des problèmes mixtes, comportant à la fois des variables discrètes et des variables continues.

Cette différenciation est nécessaire pour cerner le domaine de l'optimisation difficile. En effet, deux sortes de problèmes reçoivent, dans la littérature, cette appellation, non définie strictement :

- Certains problèmes d'optimisation discrète, pour lesquels on ne connaît pas d'algorithme exact *polynomial* (c'est-à-dire dont le temps de calcul est proportionnel à N^n, ou N désigne le nombre de paramètres inconnus du problème, et *n* est une constante entière). C'est le cas des problèmes dits " *NP*-difficiles", pour lesquels on conjecture qu'il n'existe pas de constante *n*, telle que le temps de résolution soit borné par un polynôme de degré *n*.
- Certains problèmes d'optimisation à variables continues, pour lesquels on ne connaît pas d'algorithme permettant de repérer un optimum global (c'est-à-dire la meilleur solution possible) à coup sûr et en un nombre fini d'itérations (temps de calculs).

Des efforts ont été menés, séparément, pour résoudre ces deux types de problèmes. Dans le domaine de l'optimisation continue, il existe ainsi un nombre important de méthodes classiques dites d'optimisation globale [24], mais ces techniques sont souvent inefficaces si la fonction objectif ne possède pas une propriété structurelle particulière, telle que la convexité. Dans le domaine de l'optimisation discrète, un grand nombre d'heuristiques, qui produisent des solutions proches de l'optimum, ont été développées ; mais la plupart d'entres elles ont été conçues spécifiquement pour un problème donné.

1.2.2. Algorithmes d'optimisation approchée

1.2.2.1. Heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adaptée à un problème donné. Sa capacité à optimiser un problème avec un minimum

d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée.

Du point de vue de la recherche opérationnelle, ce défaut n'est pas toujours un problème, tout spécialement quand seule une approximation de la solution optimale est recherchée.

1.2.2.2. Métaheuristiques

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de métaheuristiques.

Les métaheuristiques marquent une réconciliation des deux principaux domaines d'optimisation : en effet, celles-ci s'appliquent à toutes sortes de problèmes discrets et elles peuvent s'adapter aussi aux problèmes continus. Ces méthodes ont en commun, en outre, les caractéristiques suivantes :

- Elles sont, au moins pour partie, *stochastiques* : ce qui permet de faire face au problème de l*'explosion combinatoire* des possibilités (domaine discret) ;
- généralement, d'origine discrète, elles ont l'avantage décisif dans le cas continu, d'être directes, c'est-à-dire qu'elles ne recourent pas au calcul, souvent problématique, des gradients de la fonction objectif ;
- elles sont inspirées par des analogies : avec la physique (Recuit simulé,..), avec la biologie (algorithmes évolutionnaires, recherche avec tabou,..) ou avec l'éthologie (algorithmes de fourmis, essaims particulaires,..) ;
- elles surmontent le piége des minimums locaux en autorisant, de temps en temps, des mouvements de *remontée*, autrement dit, en acceptant une dégradation temporaire de la fonction objectif au cours de leur progression. Un mécanisme de contrôle des dégradations, spécifique à chaque métaheuristique, permet d'éviter la divergence du procédé.

Les métaheuristiques, du fait de leur capacité à être utilisées sur un grand nombre de problèmes différents, se prêtent facilement à des extensions. Pour illustrer cette caractéristique, citons notamment :

- l'optimisation multiobjectif (dites aussi multicritère) [25], où il faut optimiser plusieurs objectifs contradictoires. La recherche vise alors non pas à trouver un optimum global, mais un ensemble d'optima « au sens de Pareto » formant la « surface de compromis » du problème.

- l'optimisation multimodale, où l'on cherche un ensemble des meilleurs optima globaux et/ou locaux.
- l'optimisation de problèmes bruités, où il existe une incertitude sur le calcul de la fonction objectif. Incertitude dont il faut alors tenir comptes dans la recherche de l'optimum.
- l'optimisation dynamique, où la fonction objectif varie dans le temps. Il faut alors approcher au mieux l'optimum à chaque pas de temps.
- la parallélisation, pour accélérer la vitesse de l'optimisation en répartissant la charge de calcul sur des unités fonctionnant de concert. On parlera de métaheuristiques distribuées.
- l'hybridation, qui vise à tirer parti des avantages respectifs de métaheuristiques différentes en les combinant [26].

Enfin, la grande vitalité de ce domaine de recherche ne doit pas faire oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable. Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème donné. Du fait du foisonnement de la recherche, un grand nombre de classes de métaheuristiques existent dans la littérature. Le présent travail faisant plus particulièrement l'objet des métaheuristiques dites de « colonies de fourmis ».

1.3. Algorithmes de colonies de fourmis en optimisation

Le premier algorithme de ce type (le "Ant System") a été conçu pour la résolution du problème du voyageur de commerce [5], mais n'a pas permis de produire des résultats compétitifs. Cependant, l'intérêt pour la métaphore était lancé et de nombreux algorithmes s'en inspirant ont depuis été proposés dans divers domaines, certains atteignant des résultats très convaincants.

1.3.1. Optimisation naturelle : pistes de phéromone

Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à

résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourriture.

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées phéromones. Elles sont attirées par ces substances, qu'elles perçoivent grâce à des récepteurs situes dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères (figure 1.1).

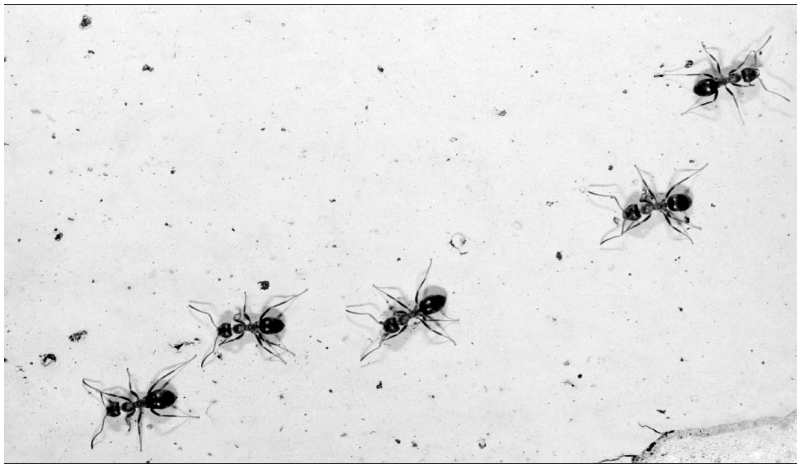


Figure 1.1 : Des fourmis suivant une piste de phéromone
(photographie de Taillard, E., tirée de [27]).

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter [28,23], sans que les individus aient une vision globale du trajet.

En effet, comme l'illustre la figure 1.2, les fourmis le plus rapidement arrivées au nid, après avoir visité la source de nourriture, sont celles qui empruntent le chemin le plus court. Ainsi, la quantité de phéromone présente sur le plus court trajet est plus importante que celle présente sur le chemin le plus long. Or, une piste présentant une plus grande concentration en phéromone est plus attirante pour les fourmis, elle a une probabilité plus grande d'être empruntée. La piste courte va alors être plus renforcée que la longue, et, à terme, sera choisie par la grande majorité des fourmis.

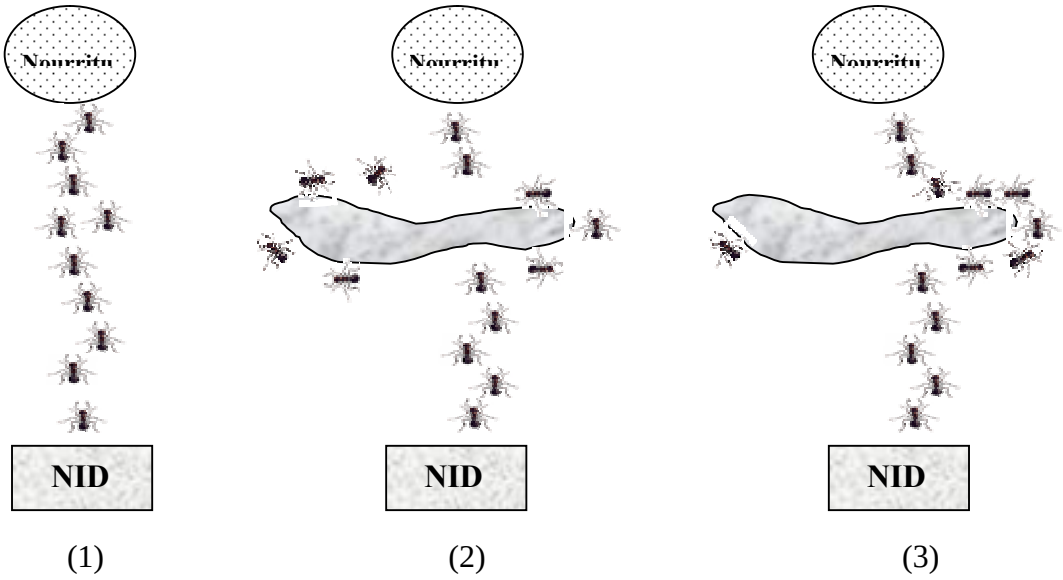


Figure 1.2 : Faculté d'une colonie de fourmis de retrouver le plus court chemin, obstrué fortuitement par un obstacle.

1. Des fourmis suivent un chemin entre le nid et une source de nourriture.
2. Un obstacle survient sur le chemin, les fourmis choisissent de tourner à gauche ou à droite avec des probabilités égales ; le phéromone est déposé plus fréquemment sur le chemin le plus court.
3. Presque la totalité des fourmis ont choisi le chemin le plus court.

On constate qu'ici le choix s'opère par un mécanisme d'amplification d'une fluctuation initiale. Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'expérience, la colonie choisisse le plus long parcours.

D'autres expériences [28], avec une autre espèce de fourmis, ont montre que si les fourmis sont capables d'effectuer des demi-tours sur la base d'un trop grand écart par rapport à la direction de la source de nourriture, alors la colonie est plus flexible et le risque d'être piégé sur le chemin long est plus faible.

Il est difficile de connaître avec précision les propriétés physico-chimiques des pistes de phéromone, qui varient en fonction des espèces et d'un grand nombre de paramètres. Cependant, les métaheuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomène d'évaporation des pistes de phéromone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet d'heuristiques leur apportant un peu plus d'informations sur le problème (par exemple une information sur la direction). Il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture)

peut être en concurrence avec l'intérêt adaptatif de tels comportements. Si l'on prend en compte l'ensemble des contraintes que subit une colonie de fourmis (prédation, compétition avec d'autres colonies, etc.), un choix rapide et stable peut être meilleur, et un changement de site exploité peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une telle option.

1.3.2. Optimisation par colonie de fourmis et problème du voyageur de commerce

1.3.2.1. Définition du problème

Le Problème du Voyageur de Commerce (PVC), ou *Travelling Salesman Problem* (*TSP*), est un classique du genre. Rappelons tout de même sa formulation générale :

Définition : *Problème du voyageur de commerce. Un voyageur de commerce doit visiter un ensemble* $\{v_1, \ldots, v_n\}$ *de n villes dont on connaît les distances respectives* $d(v_i, v_j)$, $\forall (i, j) \in \{1, \ldots, n\}$. *Le problème consiste à trouver la permutation* σ *telle que la séquence* $s = (v_{\sigma(1)}, \ldots, v_{\sigma(n)})$ *minimise la distance totale* $D(\sigma)$ *parcourue par le voyageur :*

$$D(\sigma) = \sum_{i=1}^{n-1} d(v_{\sigma(i)}, v_{\sigma(i+1)}) + d(v_{\sigma(n)}, v_{\sigma(1)}) \qquad (1.1)$$

L'espace de recherche est l'ensemble des combinaisons possibles des n villes, soit au total $n!$ combinaisons. Ce problème, NP-difficile [9], peut être aussi considère comme la recherche d'un circuit hamiltonien de longueur minimale dans un graphe complet pouvant être anti-symétrique dans le cas général ($\exists (i, j)$ tel que $d(v_i, v_j) \neq d(v_j, v_i)$).

Ce problème a fait l'objet de la première implémentation d'un algorithme de colonies de fourmis : le "Ant System" (AS) [2]. Le passage de la métaphore à l'algorithme est ici relativement facile et il est intéressant d'approfondir le principe de ce premier algorithme pour bien comprendre le mode de fonctionnement des algorithmes de colonies de fourmis. Il y a deux façons d'aborder ces algorithmes. La première, la plus évidente au premier abord, est celle qui a historiquement mené au "Ant System" original ; nous avons choisi de la décrire dans cette section. La seconde est une description plus formelle des mécanismes communs aux algorithmes de colonies de fourmis, elle sera décrite dans les prochaines sections.

1.3.2.2. Principe général : l'algorithme Ant System

Du côté algorithmique, quelques modifications sont apportées aux capacités des fourmis décrites précédemment :

- elles possèdent une mémoire ;
- elles ne sont pas totalement aveugles ;
- le temps est discret.

Dans [1], Colorni a introduits trois algorithmes qui mettent à profit ce comportement collectif. Ils sont appliques au PVC. De ces trois algorithmes, on retiendra celui qui a donné naissance à l'algorithme Ant System [5].

Dans cet algorithme, les fourmis sont placées sur les sommets d'un graphe (chaque sommet représentant une ville). Elles se déplacent d'un sommet à l'autre en empruntant les arêtes du graphe. On note par $b_i(t)$ le nombre de fourmis dans la ville i à l'instant t et soit $m = \sum_{i=1}^{n} b_i(t)$, le nombre total de fourmis.

Chaque fourmi possède les caractéristiques suivantes :

- la fourmi dépose une trace de phéromones sur l'arête (i, j) quand elle se déplace de la ville i à la ville j ;
- elle choisit la ville de destination suivant une probabilité qui dépend de la distance entre cette ville et sa position et de la quantité de phéromones présente sur l'arête (règle de transition) ;
- afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville qu'elle a déjà traversée, c'est pour cela que la fourmi doit être dotée d'une mémoire.

Pour éviter qu'une fourmi ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée liste-tabou est remise à zéro chaque fois que la fourmi a terminé un tour. La liste-tabou constitue la mémoire de la fourmi.

Les traces de phéromones sont modélisées par les variables $\tau_{ij}(t)$ qui donnent l'intensité de la trace sur le chemin (i, j) à l'instant t. La probabilité de transition du sommet i vers le sommet j par la fourmi k est donnée par :

$$p_{ij} = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}.(v_{ij})^{\beta}}{\sum_{l \notin L_k(i)} (\tau_{il}(t))^{\alpha}.(v_{il})^{\beta}} & \text{si} \quad j \notin L_k(i) \\ 0 & \text{sinon} \end{cases} \qquad (1.2)$$

Où :

- $L_k(i)$ représente la liste- tabou de la fourmi *k* située sur le sommet *i*
- $v_{ij} = 1/d_{ij}$ représente une mesure de la *visibilité,* cette information statique qui correspond à l'inverse de la distance entre les villes *i* et *j*, est utilisée pour diriger le choix des fourmis vers les villes les plus proches et éviter les villes trop lointaines ;
- α et β sont deux paramètres contrôlant l'importance relative des phéromones $\tau_{ij}(t)$ (intensité de la piste) et de la visibilité v_{ij}.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromones $\Delta\tau_{ij}^{k}$(t) sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L^k} & \text{si la fourmi } k \text{ est passée par l'arc } (i,j) \\ 0 & \text{sinon} \end{cases} \qquad (1.3)$$

où Q une constante et L^k la longueur du chemin parcouru par la fourmi *k.*

L'algorithme ne serait pas complet sans le processus d'évaporation des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous optimales, il est nécessaire de permettre au système *"d'oublier"* les mauvaises solutions. La mise à jours des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les villes :

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad (1.4)$$

Où ρ est un coefficient représentant l'évaporation des traces de phéromones. $\Delta\tau_{ij}^{k}$ représente le renforcement de l'arc (*i, j*) pour la fourmi *k.* L'algorithme 2.1 donne la structure générale de AS pour le PVC (noté Ant System-TSP).

Algorithme 1.1 : Algorithme de colonies de fourmis de base : le “Ant System-TSP”.

Pour $t = 1,. . . . . ., t_{max}$

Pour chaque fourmi $k = 1,. . . . . ., m$

Choisir une ville au hasard

Pour chaque ville non visitée i

Choisir une ville j, dans la liste des villes restantes, selon la formule (1)

Fin Pour

Déposer une piste $\Delta\tau_{ij}^{k}(t)$ sur le trajet conformément à l’équation (2)

Fin Pour

Evaporer les pistes des phéromones selon la formule (3)

Fin Pour

La valeur initiale de τ_{ij} est τ_0. Concernant le nombre de fourmis, il est raisonnablement proposé d’utiliser autant de fourmis que de villes ($m = n$). La suite de cette section présente un certain nombre d’extensions proposées autour de Ant System.

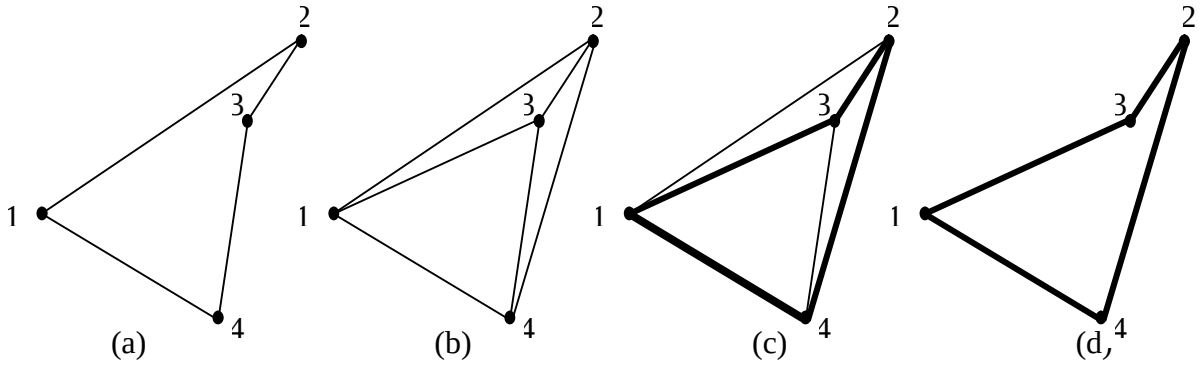


Figure 1.3 : Le problème du voyageur de commerce optimisé par l'algorithme AS, les points représentent les villes et l'épaisseur des arêtes la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi, (b) au début du calcul, tous les chemins sont explorés, (c) le chemin le plus court est plus renforcé que les autres, (d) l'évaporation permet d'éliminer les moins bonnes solutions.

1.3.2.3. Variantes

Ant System & élitisme. Une première variante du “Ant System" a été proposée dans [5] : elle est caractérisée par l'introduction de fourmis “ élitistes". Dans cette version, la meilleure fourmi (celle qui a effectué le trajet le plus court) dépose une quantité de

phéromone plus grande, dans l'optique d'accroître la probabilité des autres fourmis d'explorer la solution la plus prometteuse.

"Ant-Q" ou Ant System avec apprentissage par renforcement. Dans cette variante de l'Ant System, la règle de mise à jour locale est inspirée de l'algorithme d'apprentissage par renforcement "Q-learning" [9]. Cependant, aucune amélioration par rapport à l'algorithme Ant System n'a pu être démontrée. Cet algorithme n'est d'ailleurs, de l'aveu même des auteurs, qu'une pré-version du "Ant Colony System".

Ant Colony System. L'algorithme "Ant Colony System" (ACS) a été introduit pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles [6]. ACS est fondé sur des modifications de l'Ant System :

- ACS introduit une règle de transition dépendant d'un paramètre q_0 ($0 \leq q_0 \leq 1$), qui définit une balance *diversification/intensification*. Une fourmi k placée dans une ville i choisira une ville j par la règle :

$$j = \begin{cases} \arg\max_{l \in j_k(i)} \left\{ [\tau_{il}(t)].[v_{il}]^{\beta} \right\} & \text{si } q \leq q_0 \\ J & \text{sinon} \end{cases} \qquad (1.5)$$

où q est une variable aléatoire uniformément distribuée sur l'intervalle [0,1] et J est une ville sélectionnée aléatoirement selon la probabilité :

$$p_{iJ}^{k}(t) = \frac{[\tau_{iJ}(t)].[v_{iJ}]^{\beta}}{\sum_{l \in J_k(i)} [\tau_{il}(t)].[v_{il}]^{\beta}} \qquad (1.6)$$

où β est un paramètre servant à moduler la prise en compte des phéromones par rapport à la visibilité ;

- des listes de villes candidates sont utilisées pour accélérer le processus de construction d'un chemin ;
- une heuristique locale est utilisée pour améliorer les solutions générées par les fourmis (2-opt ou 3-opt) ;
- la mise à jour des phéromones n'est faite qu'à partir du meilleur chemin généré ;

- une règle de mise à jour locale des phéromones est utilisée à chaque transition d'une fourmi.

Les résultats obtenus par ACS sur le PVC sont les meilleurs obtenus par les heuristiques à base de fourmis sans toutefois dépasser les meilleures heuristiques dédiées à ce problème.

Max Min Ant System. Cette variante (notée MMAS) est fondée sur l'algorithme Ant System et présente quelques différences notables [29] :

- Seule la meilleure fourmi met à jour une piste de phéromone.
- Les valeurs des pistes sont bornées par τ_{min} et τ_{max}.
- Les pistes sont initialisées à la valeur maximum τ_{max}.
- La mise à jour des pistes se fait de façon proportionnelle, les pistes les plus fortes étant moins renforcées que les plus faibles.
- Une re-initialisation des pistes peut être effectuée.

Les meilleurs résultats sont obtenus en mettant à jour la meilleure solution avec une *fréquence* de plus en plus forte au cours de l'exécution de l'algorithme.

L'heuristique ACO. Toutes les variantes que nous venons d'exposer ont été regroupées sous l'acronyme ACO (Ant Colony Optimization), pour une description plus large et afin de faciliter le rapprochement des méthodes entre elles et de se soustraire aux spécificités du PVC [8,30].

1.3.2.4. Choix des paramètres

Pour l'algorithme AS, les auteurs préconisent que, bien que la valeur de Q ait peu d'influence sur le résultat final, cette valeur soit du même ordre de grandeur qu'une estimation de la longueur du meilleur trajet trouvé. D'autre part, la ville de départ de chaque fourmi est typiquement choisie par un tirage aléatoire uniforme, aucune influence significative du placement de départ n'ayant pu être démontrée.

En ce qui concerne l'algorithme ACS, les auteurs conseillent d'utiliser $\tau_0 = (n.L_{nn})^{-1}$, ou n est le nombre de villes et L_{nn} la longueur d'un tour trouvé par la méthode du plus proche voisin. Le nombre de fourmis m est un paramètre important ; les auteurs suggèrent d'utiliser autant de fourmis que de villes (i.e. $m = n$) pour de bonnes performances sur le problème du voyageur de commerce. Il est possible de n'utiliser qu'une seule fourmi, mais l'effet d'amplification des longueurs différentes est alors perdu, de même que le

parallélisme naturel de l'algorithme, ce qui peut s'avérer néfaste pour certains problèmes. En règle générale, les algorithmes de colonies de fourmis semblent assez peu sensibles à un réglage fin du nombre de fourmis.

1.3.3. Autres problèmes combinatoires

Les algorithmes de colonies de fourmis sont très étudies depuis quelques années et il serait trop long de faire ici une liste exhaustive de toutes les applications et variantes qui ont été produites, même en se restreignant au domaine de l'optimisation. Dans les deux principaux champs d'application (problèmes NP-diffciles et problèmes dynamiques), certains algorithmes ont cependant donnés de très bons résultats. On peut notamment retenir des performances particulièrement intéressantes dans le cas de l'affectation quadratique [12,13,29], de problèmes de planification [7], de coloriage de graphes [14], d'affectation de fréquence du [11], ou du routage sur réseau [8]. Il existe une littérature importante sur toutes sortes de problèmes : voyageur de commerce, ordonnancement séquentiel, routage de véhicule, affectation généralisé, sac à dos multidimensionnel, satisfaction de contraintes, etc.

1.3.4. Formalisation et propriétés d'un algorithme de colonie de fourmis

Une description élégante a été proposée [7], qui s'applique aux problèmes (combinatoires) ou une construction partielle de la solution est possible. Ce cas, bien que restrictif, permet de dégager les apports originaux des métaheuristiques ACO, nous en avons traduit ci-dessous un extrait :

> *Une métaheuristique de colonie de fourmis est un processus stochastique construisant une solution, en ajoutant des composants aux solutions partielles. Ce processus prend en compte (i) une heuristique sur l'instance du problème (ii) des pistes de phéromone changeant dynamiquement pour refléter l'expérience acquise par les agents.*

Une formalisation plus précise existe [7]. Elle passe par une représentation du problème, un comportement de base des fourmis et une organisation générale de la métaheuristique. Plusieurs concepts sont également à mettre en valeur pour comprendre les principes de ces algorithmes, notamment la définition des pistes de phéromone en tant que mémoire adaptative, la nécessite d'un réglage intensification/diversification et enfin l'utilisation d'une recherche locale. Nous traitons ci-après ces différents sujets.

1.3.4.1. Formalisation

Représentation du problème. Le problème est représenté par un jeu de solutions, une fonction objectif assignant une valeur à chaque solution et un jeu de contraintes. L'objectif est de trouver l'optimum global de la fonction objectif satisfaisant les contraintes. Les différents états du problème sont caractérises comme une séquence de composants. On peut noter que, dans certains cas, un coût peut être associé à des états autres que des solutions. Dans cette représentation, les fourmis construisent des solutions en se déplaçant sur un graphe G = (C; L), où les noeuds sont les composants de C et où l'ensemble L connecte les composants de C. Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

Comportement des fourmis. Les fourmis artificielles peuvent être caractérisées comme une procédure de construction stochastique construisant des solutions sur le graphe G tel que : G = (C; L). En général, les fourmis tentent d'élaborer des solutions faisables mais, si nécessaire, elles peuvent produire des solutions infaisables. Les composants et les connexions peuvent être associés à des pistes de phéromone (mettant en place une mémoire adaptative décrivant l'état du système) et à une valeur heuristique (représentant une information a priori sur le problème, ou venant d'une source autre que celle des fourmis ; c'est bien souvent le coût de l'état en cours). Les pistes de phéromone et la valeur de l'heuristique peuvent être associées soit aux composants, soit aux connexions.

Chaque fourmi dispose d'une mémoire utilisée pour stocker le trajet effectué, d'un état initial et de conditions d'arrêt. Les fourmis se déplacent d'après une règle de décision probabiliste fonction des pistes de phéromone locales, de l'état de la fourmi et des contraintes du problème. Lors de l'ajout d'un composant à la solution en cours, les fourmis peuvent mettre à jour la piste associée au composant ou à la connexion correspondante. Une fois la solution construite, elles peuvent mettre à jour la piste de phéromone des composants ou des connexions utilisées. Enfin, une fourmi dispose au minimum de la capacité de construire une solution du problème.

Organisation de la métaheuristique. En plus des règles régissant le comportement des fourmis, un autre processus majeur a cours : l'évaporation des pistes de phéromone. En effet, à chaque itération, la valeur des pistes de phéromone est diminuée. Le but de cette

diminution est d'éviter une convergence trop rapide et le piégeage de l'algorithme dans des minimums locaux, par une forme d'oubli favorisant l'exploration de nouvelles régions.

Selon les auteurs du formalisme ACO, il est possible d'implémenter d'autres processus nécessitant un contrôle centralisé (et donc ne pouvant être directement pris en charge par des fourmis), sous la forme de processus annexes. Ce n'est, à notre sens, que peu souhaitable ; en effet, on perd alors la caractéristique décentralisée du système. De plus, l'implémentation de processus annexes entre difficilement dans une formalisation rigoureuse.

1.3.4.2. Phéromones et mémoire

L'utilisation de la stigmergie est cruciale pour les algorithmes de colonies de fourmis. Le choix de la méthode d'implémentation des pistes de phéromone est donc important pour obtenir les meilleurs résultats. Ce choix est en grande partie lié aux possibilités de représentation de l'espace de recherche, chaque représentation pouvant apporter une façon différente d'implémenter les pistes. Par exemple, pour le problème du voyageur de commerce, une implémentation efficace consiste à utiliser une piste τ_{ij} entre deux villes *i* et *j* comme une représentation de l'intérêt de visiter la ville *j* après la ville *i*. Une autre représentation possible, moins efficace en pratique, consiste à considérer τ_{ij} comme une représentation de l'intérêt de visiter *i* en tant que *j* éme ville.

En effet, les pistes de phéromone décrivent à chaque pas l'état de la recherche de la solution par le système, les agents modifient la façon dont le problème va être représenté et perçu par les autres agents. Cette information est partagée par le biais des modifications de l'environnement des fourmis, grâce à une forme de communication indirecte : la stigmergie. L'information est donc stockée un certain temps dans le système, ce qui a amène certains auteurs à considérer ce processus comme une forme de mémoire adaptative [31], où la dynamique de stockage et de partage de l'information va être cruciale pour le système.

1.3.4.3. Intensification/diversification

Le problème de l'emploi relatif de processus de *diversification* et d'*intensification* est un problème extrêmement courant dans la conception et l'utilisation de métaheuristiques. Par *intensification*, on entend l'exploitation de l'information accumulée par le système à un

moment donné. La *diversification* est au contraire l'exploration de régions de l'espace de recherche imparfaitement prises en compte. Bien souvent, il va s'agir de choisir où et quand injecter de l'aléatoire dans le système (*diversification*) et/ou améliorer une solution (*intensification*).

Dans les algorithmes de type ACO, comme dans la plupart des cas, il existe plusieurs façons de gérer l'emploi de ces deux facettes des métaheuristiques d'optimisation. La plus évidente passe par le réglage via les deux paramètres α et β, qui déterminent l'influence relative des pistes de phéromone et de l'information heuristique. Plus la valeur de α sera élevée, plus l'*intensification* sera importante, car plus les pistes auront une influence sur le choix des fourmis. A l'inverse, plus α sera faible, plus la *diversification* sera forte, car les fourmis éviteront les pistes. Le paramètre β agit de façon similaire. On doit donc gérer conjointement les deux paramètres, pour régler ces aspects.

On peut également introduire des modifications de la gestion des pistes de phéromone Par exemple, l'emploi de stratégies *élitistes* (les meilleures solutions contribuent plus aux pistes, voir : l'algorithme AS avec élitisme) favorise l'*intensification*, alors qu'une réinitialisation de l'ensemble des pistes favorisera l'exploration (l'algorithme MMAS).

Ce choix *diversification*/*intensification* peut s'effectuer de manière statique avant le lancement de l'algorithme, en utilisant une connaissance a priori du problème, ou de manière dynamique, en laissant le système décider du meilleur réglage Deux approches sont possibles : un réglage par les paramètres ou l'introduction de nouveaux processus. Dans ces algorithmes fondés en grande partie sur l'utilisation de l'auto-organisation, ces deux approches peuvent être équivalentes, un changement de paramètre pouvant induire un comportement complètement différent du système, au niveau global.

1.3.4.4. Recherche locale et heuristiques

Les métaheuristiques de colonies de fourmis sont souvent plus efficaces quand elles sont hybridées avec des algorithmes de recherche locale. Ceux-ci optimisent les solutions trouvées par les fourmis, avant que celles-ci ne soient utilisées pour la mise à jour des pistes de phéromone. Du point de vue de la recherche locale, utiliser des algorithmes de colonies de fourmis pour engendrer une solution initiale est un avantage indéniable. Ce qui différencie une métaheuristique de type ACO intéressante d'un algorithme réellement efficace est bien souvent l'hybridation avec une recherche locale.

Une autre possibilité pour améliorer les performances est d'injecter une information heuristique plus pertinente. Cet ajout a généralement un coût élevé en terme de calculs supplémentaires.

Il faut noter que ces deux approches sont similaires de par l'emploi qu'elles font des informations de coût pour améliorer une solution. La recherche locale le fait de façon plus directe que l'heuristique, cependant que cette dernière est peut-être plus naturelle pour utiliser des informations a priori sur le problème.

1.3.4.5. Parallélisme

La structure même des métaheuristiques de colonies de fourmis comporte un parallélisme intrinsèque. D'une manière générale, les solutions de bonne qualité émergent du résultat des interactions indirectes ayant cours dans le système, pas d'un codage explicite d'échanges. En effet, chaque fourmi ne prend en compte que des informations locales de son environnement (les pistes de phéromone) ; il est donc facile de paralléliser un tel algorithme. Il est intéressant de noter que les différents processus en cours dans la métaheuristique (i.e. le comportement des fourmis, l'évaporation et les processus annexes) peuvent également être implémentés de manière indépendante, l'utilisateur étant libre de décider de la manière dont ils vont interagir.

1.3.4.6. Convergence

Les métaheuristiques peuvent être vues comme des modifications d'un algorithme de base : une recherche aléatoire. Cet algorithme possède l'intéressante propriété de garantir que la solution optimale sera trouvée tôt ou tard, on parle alors de convergence. Cependant, puisque cet algorithme de base est biaisé, la garantie de convergence n'existe plus.

Si, dans certains cas, on peut facilement être certain de la convergence d'un algorithme de colonies de fourmis (MMAS par exemple), le problème reste entier en ce qui concerne la convergence d'un algorithme ACO quelconque. Cependant, il existe une variante dont la convergence a été prouvée [32,33] : “le Graph-Based Ant System" (GBAS). La différence entre GBAS et l'algorithme AS se situe au niveau de la mise à jour des pistes de phéromone, qui n'est permise que si une meilleure solution est trouvée. Pour certaines valeurs de paramètres, et étant donné $\varepsilon > 0$ une “faible" valeur, l'algorithme trouvera la solution optimale avec une probabilité $P_t \geq 1-\varepsilon$, après un temps $t \geq t_0$ (t_0 est fonction de ε).

1.3.5. Conclusion

La métaheuristique s'inspirant des colonies de fourmis commence à être bien cernée. L'ensemble des propriétés la décrivant est connu : construction probabiliste d'une solution par ajout de composants (dans le formalisme ACO), heuristique sur l'instance du problème, utilisation d'une forme de mémoire indirecte et d'une structure comparable à celles d'un système auto organisé. Les idées directrices qui découlent de ces algorithmes de fourmis sont puissantes ; on pourrait décrire cette métaheuristique comme un système distribué où les interactions entre composants de base, par le biais de processus stigmergiques, permettent de faire émerger un comportement global cohérent rendant le système capable de résoudre des problèmes d'optimisation difficiles.

Devant le succès grandissant, rencontré par les algorithmes de colonies de fourmis, de nombreuses pistes autres que celle de l'optimisation combinatoire, commencent à être explorées. On citera entre autres, l'utilisation de ces algorithmes pour la résolution des problèmes *continus* et/ou *dynamiques*, ou encore la mise en relation de ce type d'algorithmes dans un cadre d'*intelligence en essaim* et avec d'autres métaheuristiques.

CHAPITRE 2
ADAPTATION DE LA METAHEURISTIQUE AU DOMAINE CONTINU

2.1. Introduction

Les métaheuristiques sont bien souvent élaborées pour des problèmes combinatoires, mais il existe une classe de problèmes souvent rencontrée en ingénierie, où la fonction objectif est à *variables continues* et pour lesquels les métaheuristiques peuvent être d'un grand secours (fonction non dérivable, multiples minimums locaux, grand nombre de variables, non convexité, etc.). Plusieurs tentatives pour adapter les métaheuristiques de colonies de fourmis au domaine continu sont apparues [15-18,34,35].

Outre les problèmes classiques d'adaptation d'une métaheuristique, les algorithmes de colonies de fourmis posent un certain nombre de problèmes spécifiques. Ainsi, le principal problème vient du faite que si l'on se place dans le formalisme ACO, la solution doit être construite composant par composant. En effet, un problème continu peut “selon la perspective choisie ” présenter une infinité de composants, le problème de la construction de la solution est difficilement soluble dans ce cas. La plupart des algorithmes s'inspirent donc des caractéristiques d'auto organisation et de mémoire externe des colonies de fourmis, laissant de coté la construction itérative de la solution.

Nous avons recensé dans la littérature plusieurs algorithmes de colonies de fourmis pour l'optimisation continue, on citera entre autres :

- l’algorithme CACO pour “Continuous Ant Colony Agorithm",
- un algorithme hybride non baptisé, utilisant une approche de colonies de fourmis et un algorithme évolutionnaire,
- l’algorithme CIAC pour “Continuous Interacting Ant Colony”,
- l’algorithme de fourmis de l’espèce *Pachycondyla* API*calis* (API).

Chacun de ces algorithmes fera l’objet d’une description sommaire à la suite de ce chapitre.

2.2. Difficultés spécifiques aux problèmes continus

Pour dégager ces difficultés, on se place dans le cas suivant et on suppose que :

- le problème est mono objectif ;
- la fonction objectif à minimiser est f ;
- les variables de décision sont rassemblées dans un vecteur x ;
- les seules contraintes : $x_i^{\min} \leq x_i \leq x_i^{\max}$ (contraintes de boîte).

Les problèmes d'optimisation continus présentent souvent des difficultés spécifiques. On parle aussi de « problèmes difficile », même si cette expression ne fait pas référence, ici, à la théorie de complexité, invoquée précédemment.

Les principales sources de ces difficultés sont les suivantes :

1. Inexistence de l'expression analytique de la fonction f.
2. La fonction f est bruitée. Ce bruit peut être de nature expérimentale, si le calcul de f passe par l'exploitation de mesures. Ce bruit peut aussi être un bruit de calcul (intégration/ dérivation numérique).
3. La fonction f comporte des non linéarités.
4. Il existe des corrélations « non précisément localisées » entre certains variables du problème.

Ces difficultés condamnent le recours aux calculs de gradients et entraînent l'existence possible d'une multitude de minimums locaux.
En conséquence, les méthodes qui s'attachent à résoudre efficacement de tels problèmes continus difficiles, doivent posséder deux propriétés capitales, à savoir :

- Elles doivent être « directes », c'est-à-dire sans calcul de gradients ;
- Elles doivent pouvoir s'extraire du piége d'un minimum local.

Cette double exigence motive le recours aux métaheuristiques d'une manière générale et aux algorithmes de colonies de fourmis en particulier. En effet, ces méthodes sont toutes à la fois « directes » et « globales ».

L'aspect « direct » de ce genre d'algorithmes est lié à leur origine combinatoire et c'est un avantage déterminant pour le traitement des cas continus difficiles.
L'aspect « globales » de ces méthodes vient du fait que pour surmonter l'obstacle des minimums locaux, ces métaheuristiques autorisent, de temps en temps, des mouvements de remontée, autrement dit, acceptent une dégradation temporaire de la situation, lors du

passage d'une configuration à une autre. C'est le cas si l'on passe de la configuration C_k à la configuration C_{k+1} (figure 2.1).

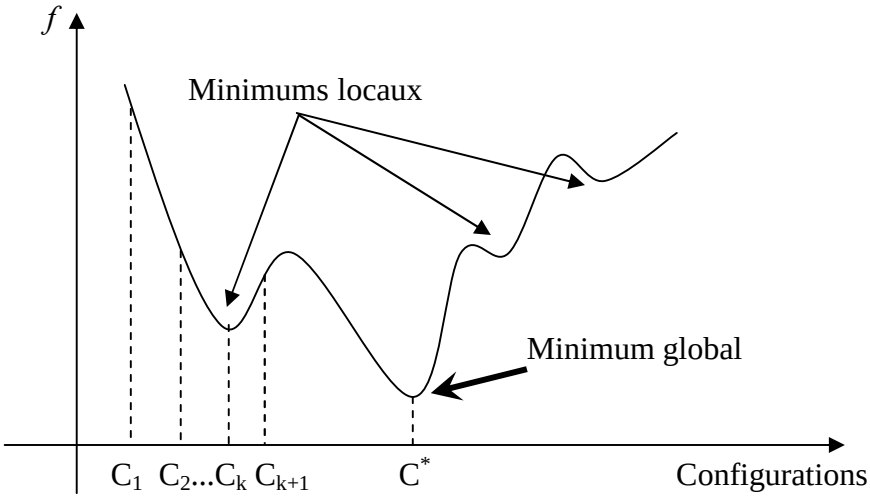


Figure 2.1 : allure d'une fonction objectif *f* d'un problème d'optimisation en fonction de la configuration.

Un mécanisme de contrôle des dégradations « spécifique à chaque métaheuristique » permet d'éviter la divergence du procédé. Il devient, dés lors possible de s'extraire que représente un minimum local, pour partir explorer une autre « vallée » plus prometteuse.

2.3. Quelques algorithmes de colonie de fourmis pour l'optimisation continue

2.3.1. L'algorithme CACO

Le premier des algorithmes de fourmis pour les cas continus, tout naturellement nommé **CACO** "Continuous Ant Colony Algorithm" [15,16], utilise deux approches : un algorithme de type évolutionnaire sélectionne et croise des régions d'intérêt, que des fourmis explorent et évaluent. Une fourmi sélectionne une région avec une probabilité proportionnelle à la concentration en phéromone de cette région, de la même manière que dans le « Ant System », une fourmi sélectionnerait une piste allant d'une ville à une autre :

$$p_i(t) = \frac{\tau_i^{\alpha}(t) \cdot \eta_i^{\beta}(t)}{\sum_{j=1}^{N} \tau_j^{\alpha}(t) \cdot \eta_j^{\beta}(t)} \qquad (2.1)$$

où N est le nombre de régions et $\eta_i^{\beta}(t)$ est utilisée pour inclure une heuristique spécifique au problème. Les fourmis partent alors du centre de la région et se déplacent selon une direction choisie aléatoirement, tant qu'une amélioration de la fonction objectif est trouvée. Le pas de déplacement utilisé par la fourmi entre chaque évaluation est donne par :

$$\delta r(t, R) = R \cdot \left(1 - u^{(1-\frac{t}{T})^c}\right) \qquad (2.2)$$

Où R est le diamètre de la région explorée, $u \in [0,1]$ un nombre aléatoire, T le nombre total d'itérations de l'algorithme et c un paramètre de refroidissement (permettant de réduire le pas à chaque itération). Si la fourmi a trouvé une meilleure solution, la région est déplacée de façon à ce que son centre coïncide avec cette solution, et la fourmi augmente la quantité de phéromone de la région proportionnellement à l'amélioration trouvée (appelée ici fitness"). L'évaporation des « pistes » se fait classiquement en fonction d'un coefficient ρ.

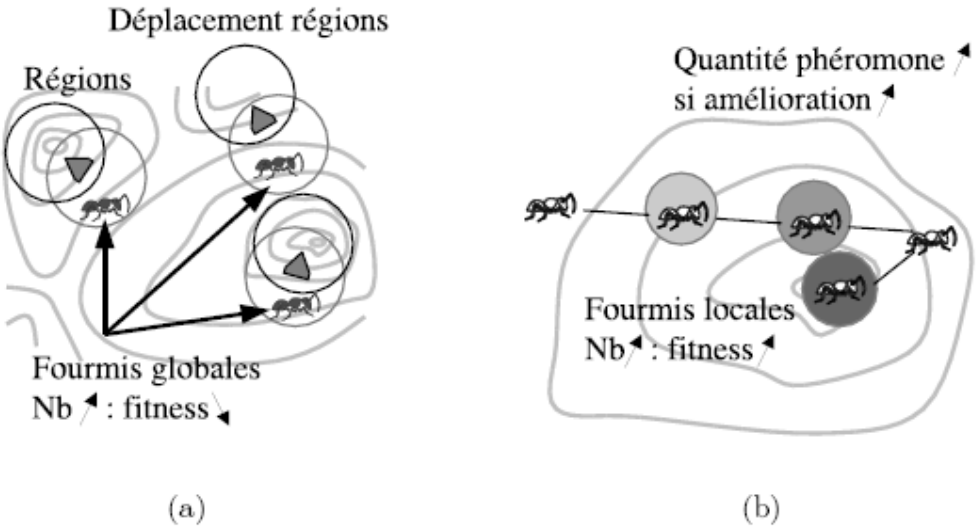


Figure 2.2 : L'algorithme CACO : les fourmis globales (a) participent au déplacement des régions que les fourmis locales (b) évaluent.

2.3.2. Une méthode hybride

Une méthode hybride non baptisée, utilisant à la fois une approche de colonies de fourmis et un algorithme évolutionnaire, a été proposée par Ling et al. [17], mais peu de résultats sont disponibles jusqu'à ce jours. L'idée principale de cette méthode est de considérer les écarts entre deux individus sur chaque dimension comme autant de parties d'un chemin où

les phéromones sont déposées, l'évolution des individus étant prise en charge par des opérateurs de *mutation* et de *croisement*. D'un certain point de vue, cette méthode tente donc de reproduire le mécanisme de construction de la solution, composant par composant.

La méthode procède précisément comme décrit dans l'algorithme (2.1). Chaque fourmi x_i de la population contenant m individus est considérée comme un vecteur à n dimensions. Chaque élément $x_{i,e}$ de ce vecteur peut donc être considéré comme un candidat à l'élément $x^*_{i,e}$ de la solution optimale. L'idée est d'utiliser le chemin entre les éléments $x_{i,e}$ et $x_{j,e}$ « noté (i,j) » pour déposer une piste de phéromone dont la concentration est notée $\tau_{ij}(t)$ au pas de temps t.

Algorithme 2.1 : Un algorithme de colonies de fourmis hybride pour le cas continu.

---

1. À chaque itération, sélectionner pour chaque fourmi une valeur initiale dans le groupe de valeurs candidates avec la probabilité :

$$p_{ij}^{k}(t) = \frac{\tau_{ij}(t)}{\sum \tau_{ir}(t)}$$

2. Utiliser des opérateurs de mutation et de croisement sur les m valeurs, afin d'obtenir m nouvelles valeurs ;

3. Ajouter ces nouvelles valeurs aux valeurs candidates pour le composant $x_{i,e}$;

4. Les utiliser pour former m solutions de la nouvelle génération ;

5. Calculer la « fitness » de ces solutions ;

6. Quand m fourmis ont parcouru toutes les arêtes, mettre à jour les pistes de phéromone des valeurs candidates de chaque composant par :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ir}(t) + \sum \tau_{ij}^{k}$$

7. Si la k^{éme} fourmi choisit la j^{éme} valeur candidate du groupe de composants, alors $\Delta\tau_{ij}^{k}(t+1) = W f_k$, sinon $\Delta\tau_{ij}^{k} = 0$. Avec W une constante et f_k la fitness de la solution trouvée par la k^{éme} fourmi ;

8. Effacer les m valeurs ayant les plus basses intensités de phéromone dans chaque groupe de candidats.

2.3.3. L’algorithme ACO pour l’optimisation continue

Cet algorithme [18] tente de maintenir la construction itérative des solutions dans le cas de variables continues, en adoptant un point de vue différent des précédents. En effet, il prend le parti de considérer que les composants de toutes les solutions sont formés par les différentes variables optimisées. De plus, plutôt que de considérer l'algorithme du point de vue de la fourmi, il prend le parti de se placer au niveau de la colonie, les fourmis n'étant plus que des points à évaluer.

Dans cette méthode, dénommée simplement “ACO pour l'optimisation continue", on tire aléatoirement une population de fourmis dans une distribution de probabilité à chaque itération. De cet ensemble de points ne sont conservés que les meilleurs points, qui servent alors à construire une “meilleure" distribution de probabilité. La distribution de probabilité utilisée est un “amalgame pondéré de noyaux normaux". Soit un ensemble de distributions normales combinées :

$$P(x) = \sum_{j=1}^{k} w_j \cdot \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) \qquad (2.3)$$

en désignant par k le nombre de noyaux utilisés, μ et σ^2 la moyenne et la variance d'un noyau et w_i la pondération. Chaque distribution n'est utilisée que sur une variable, sans dépendance avec les autres. La modification des distributions est nommée “mise à jour de phéromone" et consiste à renforcer ou à diminuer l'influence des noyaux correspondant aux solutions. L’algorithme 2.2 illustre le principe de la méthode.

2.3.4. L’algorithme CACS

Cet algorithme appelé “Continuous Ant Colony System" est très proche du précédent, bien qu'ayant été présentée simultanément. En effet, dans CACS comme dans « ACO pour l'optimisation continue », le coeur de l'algorithme consiste à faire évoluer une distribution de probabilité. Le principe de la méthode est le même que celui présenté sur l'algorithme 2.2. Dans CACS, la distribution utilisée est dite “normale" mais la formule

Algorithme 2.2 : Algorithmes du type colonies de fourmis, utilisant l'estimation de distribution pour l'optimisation en variables continues.

Construire la distribution de probabilité initiale : $\tau_i^0 = P_i^0(x_i), i \in \{1 \dots n\}$

Tant que (critère d'arrêt non atteint) :

Pour chaque fourmi, de a = 1 à m :

Pour chaque variable, de i = 1 à n :

Choisir aléatoirement une valeur x_i selon la distribution $P_i(x_i)$

Ajouter à la solution en construction : $S^a = \{s_1^a, \dots, s_{i-1}^a\} \cup \{x_i\}$

Fin

Fin

Mémoriser les k meilleures solutions trouvées : $S^* = \{s_1^*, \dots, s_k^*\}$

Reconstruire la distribution de probabilité selon les meilleures solutions :

$$\tau = P(S^*)$$

Fin

de la distribution diffère légèrement de la formule classique (équation 1.5) et la variance utilisée est en fait un nouvel indice de dispersion (voir équation 1.6).

$$P(x) = e^{-\frac{(x-x_{min})^2}{2\sigma^2}} \tag{2.4}$$

en désignant par x_{min} le mode de la distribution et σ^2 l'indice de dispersion suivant :

$$\sigma^2 = \frac{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}} (x_j - x_{min})^2}{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}}} \tag{2.5}$$

Où m le nombre de fourmis, f_j la valeur de la fonction associée à la fourmi j; et f_{min} la meilleure valeur trouvée.

Dans CACS, la seule distribution utilisée est centrée sur le mode de la distribution de l'itération précédente, et non sur la moyenne.

2.3.5. L'algorithme CIAC

Un autre algorithme, se focalisant sur les principes de communication des colonies de fourmis a été proposé pour l'optimisation des fonctions continues dans [36,37]. Cet algorithme, appelé CIAC pour "Continuous Interacting Ant Colony", inspiré du concept d'*hétérarchie dense* introduite par Wilson en 1988 [38], utilise deux canaux de communication observables dans la nature:

- Le canal stigmergique classique : ce canal fait appel à des spots de phéromone, déposés sur l'espace de recherche, qui vont être plus ou moins attractifs pour les fourmis, selon leurs concentrations et leurs distances. Les caractéristiques de ce canal sont donc : le nombre d'individus mis en cause dans l'échange d'information (portée) est maximum, toutes les fourmis peuvent prendre en compte l'information, il y a utilisation de mémoire puisque les spots persistent sur l'espace de recherche, enfin l'information évolue avec le temps puisque les spots s'évaporent. L'information portée par un spot contient implicitement la position d'un point et explicitement la valeur de l'amélioration trouvée par la fourmi.
- Le canal direct est implémenté sous la forme d'échange de messages entre deux individus. Une fourmi possède une pile de messages reçus et peut en envoyer à une autre fourmi. La portée de ce canal est de un puisque seule une fourmi reçoit le message, la mémoire est implémentée dans la pile de message mémorisée par la fourmi, enfin l'information n'est pas altérée au cours du temps.

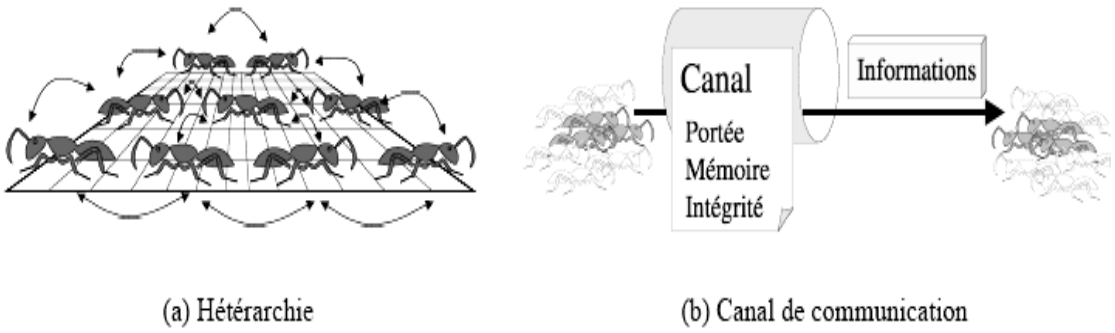


Figure 2.3 : Les notions d'hétérarchie et de canal de communication

L'algorithme ainsi définit, est un algorithme d'optimisation *hétérarchique* qui tire avantage du flot d'information passant dans une population d'agents. Ces informations sont

échangées à travers des *canaux de communication* et permettent qu'une certaine description de la fonction objectif émerge du système.

2.3.6. L'algorithme API

Dans tous les algorithmes évoqués jusqu'ici, le terme "colonies de fourmis" s'entend de par l'utilisation de la stigmergie comme processus d'échange d'information. Il existe cependant un algorithme adapté au cas continu qui s'inspire du comportement de fourmis primitives de l'espèce *Pachycondyla Apicalis* (**API***calis*), et qui ne fait pas usage de la communication indirecte par pistes de phéromone : l'algorithme **API** [20].

Dans cette méthode, on commence par positionner un nid aléatoirement sur l'espace de recherche, puis des fourmis sont distribuées aléatoirement dans l'espace. Ces fourmis vont alors explorer localement leur "site de chasse" en évaluant plusieurs points dans un périmètre donné (figure 2.4). Chaque fourmi mémorise le meilleur point trouvé. Si, lors de l'exploration de son site de chasse, elle trouve un meilleur point, alors elle reviendra sur ce site, sinon, après un certain nombre d'explorations, elle choisira un autre site. Une fois les explorations des sites de chasse terminées, des fourmis tirées au hasard comparent deux à deux (comme peuvent le faire les fourmis réelles) leurs meilleurs résultats, puis mémorisent le meilleur des deux sites de chasse. Le nid est finalement réinitialisé sur le meilleur point trouvé après un temps donné, la mémoire des sites des fourmis est remise à zéro, et l'algorithme effectue une nouvelle itération.

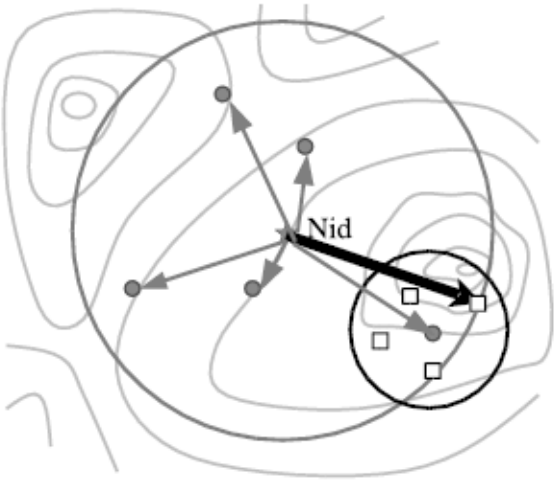


Figure 2.4 : L'algorithme **API**, une méthode à démarrage multiple inspirée par une espèce de fourmi primitive. Les fourmis (cercles pleins) explorent des sites de chasse (petits carrés) dans un périmètre (grand cercle) autour du nid. Le nid est déplacé sur le meilleur point au moment de la réinitialisation (flèche en trait gras).

2.3.7. Conclusion

Les applications continues sont le plus souvent du ressort de l'ingénieur, en revanche, le savoir-faire concernant les métaheuristiques est plutôt du côté des informaticiens, qui sont moins intéressés par les problèmes continus. Ce décalage explique sans doute la pauvreté des résultats théoriques dans ce domaine.

De nombreuses méthodes ont été proposées dans la littérature pour adapter les métaheuristiques de colonies de fourmis au cas continus, on en a vu, dans cette thèse, six. Un certain nombre d'entre eux sont plus ou moins hybridés avec un algorithme de type évolutionnaire, d'autres utilisent une approche différente des précédentes, plus centrée sur l'aspect probabiliste que comportemental des algorithmes à colonies de fourmis. D'une manière générale, la recherche en est encore à ses débuts et les algorithmes produits n'ont pas atteint leur pleine maturité, et ne sont donc pas encore vraiment compétitifs par rapport à d'autres classes de métaheuristiques plus élaborées sur les problèmes continus.

La suite de ce travail sera consacrée à l'étude de l'algorithme de colonie de fourmis **API** déduit de la modélisation du comportement de fourragement d'une population de fourmis primitives (*Pachycondyla apicalis*) et à son application au problème d'optimisation « difficile » à variables continues.

CHAPITRE 3
L'ALGORITHME API

3.1. Introduction

Nous présentons dans cette section une modélisation du comportement de fourragement d'une population de fourmis primitives (*Pachycondyla apicalis*) et son application au problème général d'optimisation. Ces fourmis sont caractérisées par une stratégie de recherche de proie relativement simple où les individus chassent en solitaire et tentent de couvrir uniformément un espace donné autour de leur nid par des recherches locales sur des sites de chasse. Le nid peut être déménagé périodiquement. Cela correspond en optimisation à un algorithme effectuant plusieurs recherches aléatoires en parallèle et localisées uniformément dans un sous-espace centré en un point. Le déplacement du nid correspond à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé.

Ce chapitre présente d'abord, les travaux menés autour de la modélisation des fourmis de l'espèce *Pachycondyla apicalis*. Cette étude trouve son origine dans les travaux de Dominique Fresneau sur la stratégie de fourragement originale de cette espèce de fourmis ponérine [22]. Nous commencerons donc par présenter les caractéristiques biologiques qui ont pu être exploitées du point de vue d'une modélisation algorithmique avec comme objectif de l'appliquer aux problèmes d'optimisation.

L'intérêt de ces fourmis pour l'optimisation vient du fait qu'elles utilisent des principes relativement simples à la fois d'un point de vue global et local pour rechercher leurs proies. A partir de leur nid, elles couvrent globalement une surface donnée en la partitionnant en plusieurs sites de chasse individuels. Pour une fourmi donnée, on observe une stratégie d'exploration aléatoire des sites, sensible au succès rencontré. Ces principes peuvent être repris pour résoudre un problème analogue qui est la recherche d'un minimum global, par exemple d'une fonction f définie de R^l dans R.

La suite de ce chapitre est organisée de la façon suivante : la section 3.2 décrit de manière succincte les principes utilisés dans la recherche de proies de *Pachycondyla apicalis*. La section 3.3 donne une modélisation de ces principes, à la fois du point de vue

mathématique de l'optimisation que du point de vue algorithmique de l'informatique. Un certain nombre d'extensions tirées du comportement naturel des fourmis sont proposées dans la section 3.4 dans le but d'enrichir le modèle. La section 3.5 propose une étude sur l'influence des principaux paramètres du modèle sur le problème de l'optimisation en générale. Les propriétés fondamentales de ce nouvel algorithme, appelé API (*Pachycondyla* API*calis*), sont également passées en revue, et notamment ses liens avec d'autres méthodes d'optimisation à base de population (section 3.6). La section 3.7 décrit les tests expérimentaux qui ont été réalisés sur un problème d'optimisation classique (le problème du Voyageur de Commerce (PVC)). Puis nous analyserons les faiblesses et les atouts de API et présenterons les perspectives possibles (section 3.8).

3.2. Biologie de *Pachycondyla apicalis*

Cette section donne un aperçu de la biologie de *Pachycondyla apicalis* et notamment de leur stratégie de recherche de nourriture (le fourragement). Une étude très complète leur est consacrée par Dominique Fresneau de l'Université Paris XIII, dans [22]. *Pachycondyla apicalis* est une ponérine néotropicale que l'on rencontre en Amérique du Sud, en particulier au Mexique. Sa morphologie et le peu de communication entre individus la classe parmi les fourmis dites primitives mais certains aspects de son adaptation démentent ce jugement. Les fourmis *Pachycondyla apicalis* vivent en petites colonies comportant quelques dizaines d'individus (40 à 100 ouvrières).

Le nid est installé dans de vieilles souches ou dans des arbres morts en décomposition, ce qui correspond un habitat instable pour des fourmis qui ne savent pas construire de fourmilière. Quand la vétuste de leur nid devient trop importante, elles doivent déménager et chercher un nouveau refuge.

Leur nourriture se compose de petits insectes qu'elles capturent ou de cadavres d'insectes. Les ouvrières prospectent individuellement autour de la fourmilière et ramènent les proies au nid. Toutes les fourmis ne participent pas à la recherche de la nourriture, celles restant au nid s'occupent principalement du couvain. Le comportement de recherche de nourriture n'utilise pas de comportement collectif direct. Par exemple, les ouvrières ne déposent pas de message chimique (traces de phéromone) sur le sol pour indiquer à d'autres fourrageuses le chemin menant à une source de nourriture. Elles ne mettent donc pas en oeuvre des comportements de recrutement de masse.

On peut supposer que la nature des proies recherchées n'encourage pas spécialement ce genre de communication inter individus : la présence des proies étant relativement aléatoire, la capture d'une proie ne donne que peu d'informations sur la stabilité spatiale de la source de nourriture. Autrement dit, la probabilité de retrouver une proie dans la même zone qu'une précédente capture n'est pas suffisante pour y canaliser les forces de fourragement de la colonie. Cependant, d'un point de vue individuel, les ouvrières mémorisent leur site de capture et lors de leur prochaine sortie du nid, elles retournent systématiquement sur le dernier site de chasse fructueux. Cette spécialisation sectorielle est une réponse pour l'adaptation nécessaire à la découverte et l'exploitation de sources de nourriture. Ce type de fourragement solitaire se retrouve particulièrement chez les espèces peu populeuses, les espèces à population importante utilisent des mécanismes de recrutement massif beaucoup plus couramment [39]. Les fourrageuses solitaires développent en conséquence des mécanismes d'apprentissage plus évolués.

De sorties en sorties, les ouvrières s'éloignent du nid car la probabilité de trouver une proie est inversement proportionnelle à la densité de fourrageuses qui décroît évidemment quand la distance au nid augmente. Ainsi, les fourrageuses ont un comportement collectif indirect puisque de manière statistique elles coopèrent pour couvrir au mieux leur espace de recherche que constitue le voisinage du nid. Elles construisent de cette façon une mosaïque de zones de chasse qui couvre la périphérie du nid. La figure 3.1 présente les aires de fourragement d'une colonie *Pachycondyla apicalis*.

Le comportement de fourragement de *Pachycondyla apicalis* peut être résumé en trois règles essentielles [22] :

1. La découverte d'une proie entraîne toujours le retour sur le site lors de la sortie suivante. C'est là que la fourrageuse reprend ses nouvelles prospections ;
2. la découverte d'une proie pèse sur la décision de sortie des fourrageuses en réduisant l'intervalle de temps qu'elles passent au nid ;
3. les fourrageuses semblent progressivement apprendre une association entre une direction de recherche opposée au nid et l'augmentation de la probabilité de succès.

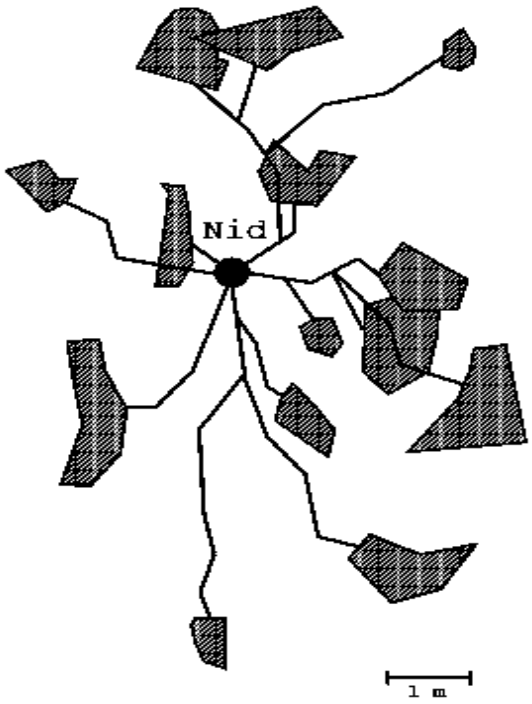


Figure 3.1 : Exemple (fictif) de carte des trajets et aires de récolte des fourrageuses.

Ces trois règles ont l'avantage d'être très simples, on peut cependant les renforcer par quelques points importants pour la mise en œuvre de l'algorithme :

- Lors de ses premières sorties, la fourmi choisit une direction aléatoire, s'éloigne peu du nid et retourne à l'abri de celui-ci à la moindre alerte ;
- si la fourmi capture une proie, elle retourne directement au nid et mémorise visuellement le chemin ;
- le retour sur un site de chasse se soldant par un échec peut se produire plusieurs fois de suite ;
- un site de chasse ne produisant plus le renforcement que constitue la capture de proie est abandonné mais pas obligatoirement oublié par la fourrageuse ;
- l'exploration d'un site de capture privilégie les directions qui éloignent la fourmi du nid dans une limite de périmètre imposé par le coût énergétique prohibitif que représente un échec à une grande distance du nid.
- Le déménagement du nid sur la position du meilleur site de chasse a pour effet de réinitialiser la mémoire des fourrageuses.

Comme nous l'avons déjà mentionné, la fragilité du nid impose des déménagements réguliers. Des fourmis éclaireuses partent alors à la recherche d'un nouvel abri. Le déménagement s'effectue grâce à des mécanismes de *recrutement en tandem* où une fourmi se fait guider par une de ses congénères jusqu'au nouvel emplacement. Ce changement de nid, a pour effet de réinitialiser la mémoire des fourrageuses. C'est à cette occasion que l'utilisation de repères visuels prend toute son importance : le marquage de chemins avec des phéromones perturberait l'adaptation des fourmis à la situation de leur nouveau nid, car les anciens chemins interféreraient avec les nouveaux. Avec une mémoire visuelle, les fourrageuses reconstruisent un réseau de sites de chasse autour du nouveau nid plus aisément, plus rapidement. On peut donc parler d'apprentissage en ce qui concerne la recherche de proies : la fourmi apprend les sites qui lui rapportent de la nourriture et elle est capable de les oublier quand elle n'y trouve plus de proie ou que le nid a changé de localisation.

L'aspect individuel de la recherche est particulièrement adapté à la fréquence d'apparition des proies : si une proie tombe sur le sol et est capturée, la fourrageuse reviendra explorer le site toute seule. Si la proie est trop lourde, elle sera découpée puis transportée en plusieurs voyages par la même fourmi qui utilise de cette façon sa mémoire. Si le site de chasse se trouve être un gisement (par exemple de termites) la fourrageuse reviendra systématiquement jusqu'à épuisement du site.

L'intérêt de la stratégie de fourragement des fourmis *Pachycondyla apicalis* réside dans sa simplicité et la bonne couverture de l'espace de recherche qui en résulte. On a ici un phénomène d'émergence de règles de recherche simples et individuelles, qui ne tiennent pas compte du travail des autres fourmis, on obtient une exploration radiale de l'espace centrée sur le nid.

Dans la réalité, l'efficacité de la stratégie n'est pas parfaite si on considère le nombre de proies trouvées relativement au nombre de proies présentes [19]. Cependant, la taille d'une colonie de *Pachycondyla apicalis* étant relativement réduite, les besoins en nourriture, sont relativement modestes. Il semble que l'adaptation de ces fourmis ne réside pas seulement dans leur comportement de fourragement mais aussi dans le maintien de colonies peu populeuses. Ceci permet aux *Pachycondyla apicalis* de survivre dans des secteurs comportant de nombreux prédateurs concurrents et de ne pas nécessiter une grande quantité d'insectes.

3.3. Modélisation algorithmique

Nous présentons, dans cette section, l'adaptation de la stratégie de fourragement des *Pachycondyla apicalis* à la résolution de problèmes d'optimisation. Nous confirmerons la généricité de la méthode relativement à l'espace de recherche avant d'étudier l'influence des différents paramètres sur le rendement de l'algorithme déduit.

3.3.1. Espace de recherche et fonction d'évaluation

On considère une population de n fourmis fourrageuses $a_1, \ldots, a_n$ de l'espèce *Pachycondyla apicalis.* Ces agents sont positionnés dans l'espace de recherche, noté S, et vont tenter de minimiser une fonction d'évaluation f définie de S dans **R.**

Chaque point $s \in S$ est une solution valide du problème, ce qui signifie que f est défini en tout point de S. Cet espace de recherche peut être un espace continu, binaire ou un espace de permutations. L'algorithme **API**, est générique en ce qui concerne l'espace de recherche S, c'est là un de ses principaux atouts. La définition des deux opérateurs suivants est suffisante pour déterminer le déplacement des fourmis :

- l'opérateur O_{rand} qui génère un point s de S de manière uniformément aléatoire ;
- l'opérateur O_{explo} qui génère un point s' dans le voisinage du point s.

Concernant le deuxième opérateur, la taille du voisinage de s est paramétrée par une amplitude, notée A telle que $A \in [0, 1]$. Cette amplitude fixe la portée de l'exploration autour de s relativement à la taille de l'espace. O_{explo} peut être une exploration aléatoire tout comme une heuristique inspirée par le domaine de recherche.

3.3.2. Comportement local des fourmis

Après chaque déplacement du nid **N**, la fourrageuse quitte le nid pour formuler une nouvelle liste de p sites de chasse qu'elle mémorise (figure 3.2), un site de chasse est un point s de l'espace de recherche S construit par un opérateur O_{explo} utilisant un rayon de recherche A_{sit} autour du nid **N**. Puis la fourmi va commencée une exploration locale au voisinage d'un des p sites de chasse mémorisés (figure 3.3).

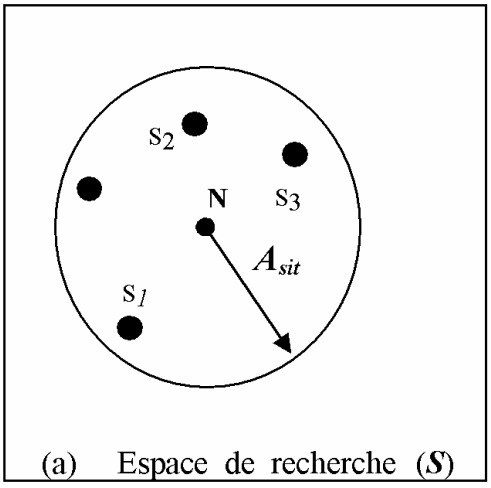


Figure 3.2 : Recherche de sites de chasse. La fourmi se constitue une liste de $p = 4$ sites de chasse au voisinage du nid **N** à une distance maximale A_{sit}.

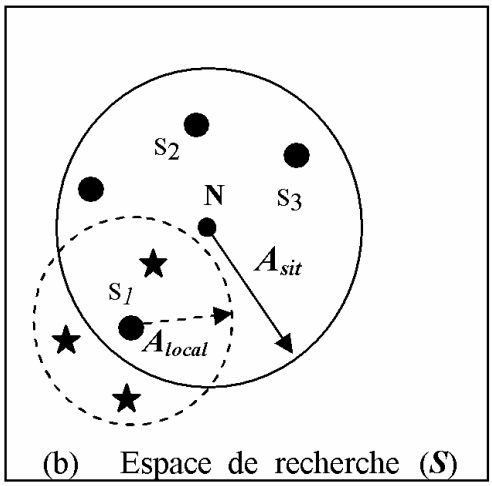


Figure 3.3 : Exploration locale de la fourmi autour du site s_1. Les étoiles représentent les explorations menées dans la zone de rayon A_{local}.

Initialement, quand l'intérêt des sites est inconnu, la fourmi choisi un site s au hasard parmi les p dont elle dispose. L'exploration locale consiste à construire un point s' de l'espace de recherche S grâce à l'opérateur O_{explo} avec un rayon de recherche A_{local}. La fourrageuse capture une proie si cette exploration locale conduit à une meilleure valeur de f, soit : $f(s') < f(s)$. A chaque fois qu'une fourmi parvient à améliorer $f(s)$, elle mémorise le site s' à la place de s et sa prochaine exploration locale se fera dans le voisinage de s'.

Si l'exploration locale est infructueuse, la fourmi choisira pour une exploration future, un site au hasard parmi les p sites qu'elle a en mémoire.

Quand un site a été exploré plus de P_{local} fois sans succès, il est définitivement oublié et sera remplacé par un nouveau site à la prochaine sortie (prochaine itération). Le paramètre P_{local} représente une patience locale. L'organigramme de la figure 3.4 résume le comportement individuel d'une fourrageuse. n_s représente le nombre de sites que la fourmi mémorise à un instant donné et e_j le nombre d'échecs successifs dans le site s_j mémorisé.

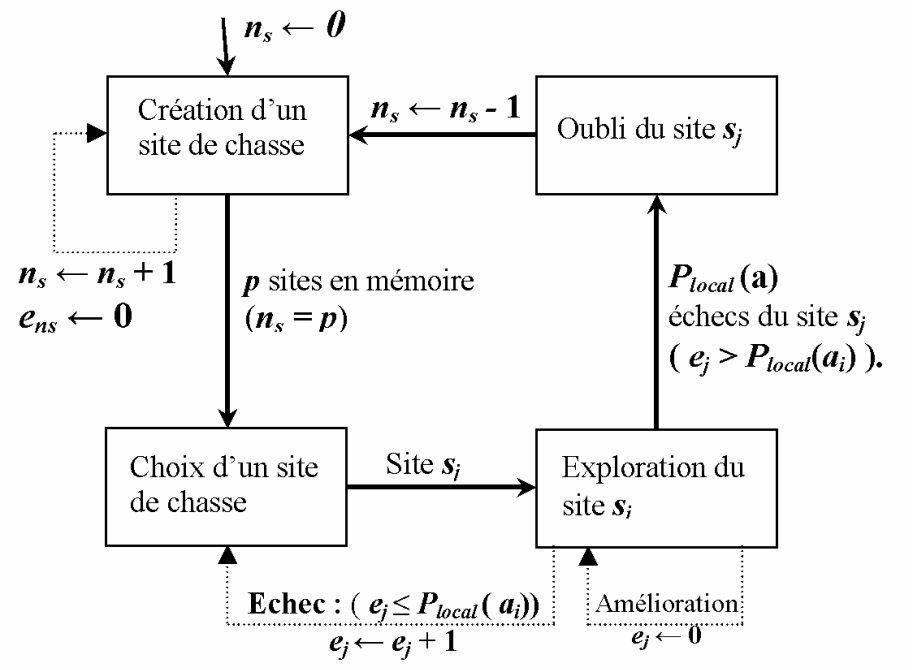


Figure 3.4 : Organigramme résumant le comportement individuel d'une fourrageuse.

3.3.3. Exploration globale de l'espace de recherche.

D'un point de vue global, **API** place aléatoirement le nid à une position **N** de l'espace ***S***, et commence l'exploration de l'espace au voisinage de **N**. L'exploration est déterminée par le comportement local des fourmis. A chaque pas de l'algorithme les ***n*** fourmis sont simulées en parallèle. À l'initialisation, le nid est placé dans l'espace de recherche ***S*** d'une manière uniformément aléatoire par l'opérateur O_{rand}. Puis le nid est déplacé toutes les P_N déplacements des ***n*** fourmis. Il est alors placé sur le meilleur point s^+ trouvé depuis son dernier déplacement. A chaque déplacement du nid les fourmis reprennent leur exploration à partir de la nouvelle position du nid. La figure 3.5 montre la stratégie globale de l'exploration. P_N représente un paramètre de patience pour le nid. Ce paramètre est fixé suivant la patience locale d'une fourmi (P_{local}) et la taille de sa mémoire (p) est :

$$P_N = 2 \times (P_{local} + 1) \times p \qquad (3.1)$$

Ce calcul a pour but de laisser suffisamment d'itérations entre chaque déplacement de nid pour que les fourmis puissent créer et explorer leurs ***p*** sites de chasse. Une autre solution serait de ne déplacer le nid que si l'optimum n'a pas été amélioré depuis un certain nombre d'itérations P_N.

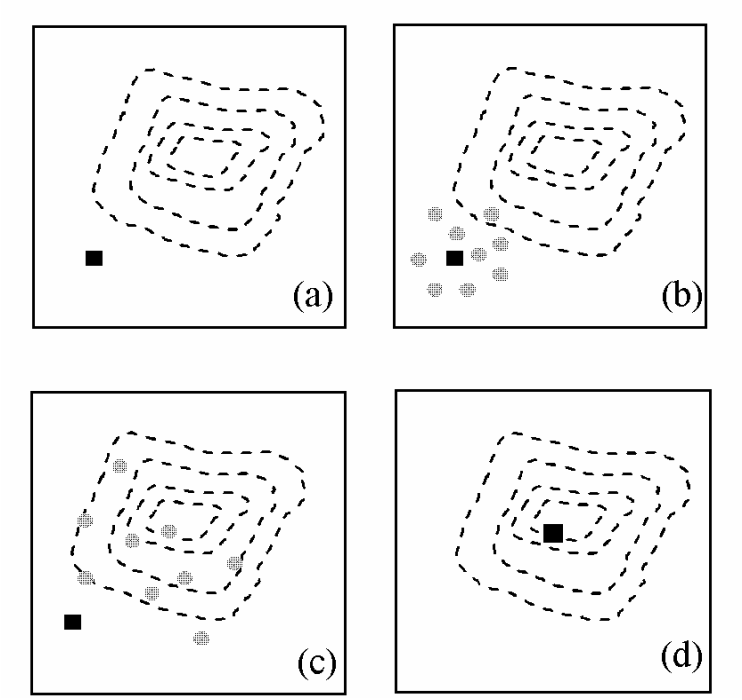
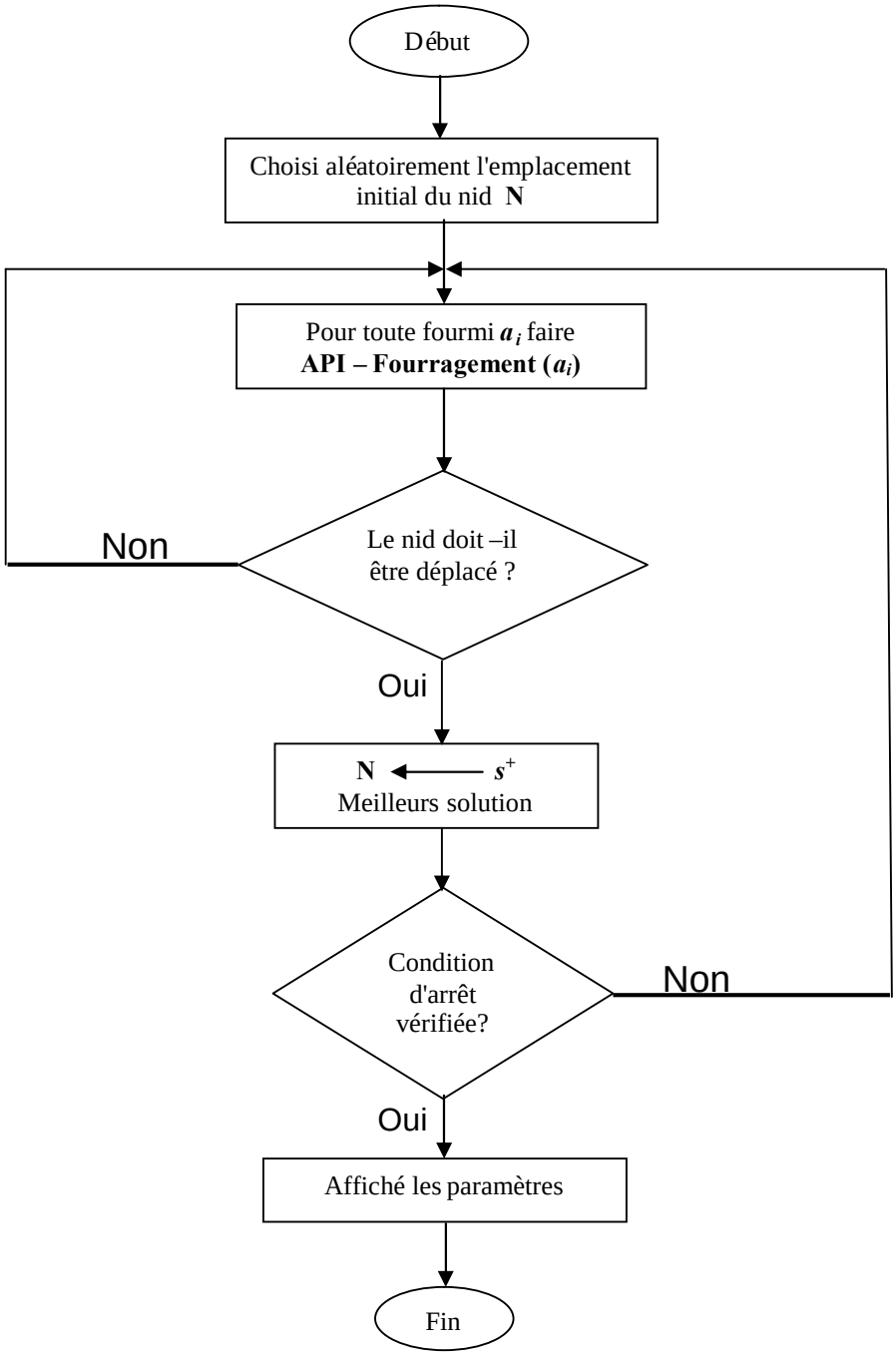


Figure 3.5 : ***Exploration globale et déplacement du nid.*** En (a), le nid (carré) est placé aléatoirement dans l'espace de recherche. En (b) sont représentés les sites de chasse (cercles) créés autour du nid. En (c), à cause de l'exploration locale, les sites de chasse se déplacent vers des zones plus intéressantes de l'espace de recherche (ici les pointillés les plus au centre). En (d) le nid est déplacé sur la position du meilleur site de chasse, les sites seront ensuite générés à partir de cette nouvelle position, comme en (b), et ainsi de suite.

Enfin, à chaque déplacement du nid, la mémoire des fourmis est vidée et elles doivent reconstruire leurs ***p*** sites de chasse. Du point de vue de l'optimisation, cela permet d'éviter des minima locaux dans lesquels les fourmis resteraient enfermées. Cela permet aussi de rassembler les fourmis autour du meilleur point trouvé et ainsi de concentrer les recherches. On pourrait cependant procéder d'une manière plus douce : il suffirait de placer le nid à la position du minimum global trouvé par la colonie à chaque fois que celui-ci est amélioré sans réinitialiser toutes les fourmis. Ainsi, quand une fourmi crée un nouveau site de chasse, elle le ferait dans le voisinage de l'optimum global. On se débarrasse alors du choix de la patience du nid.

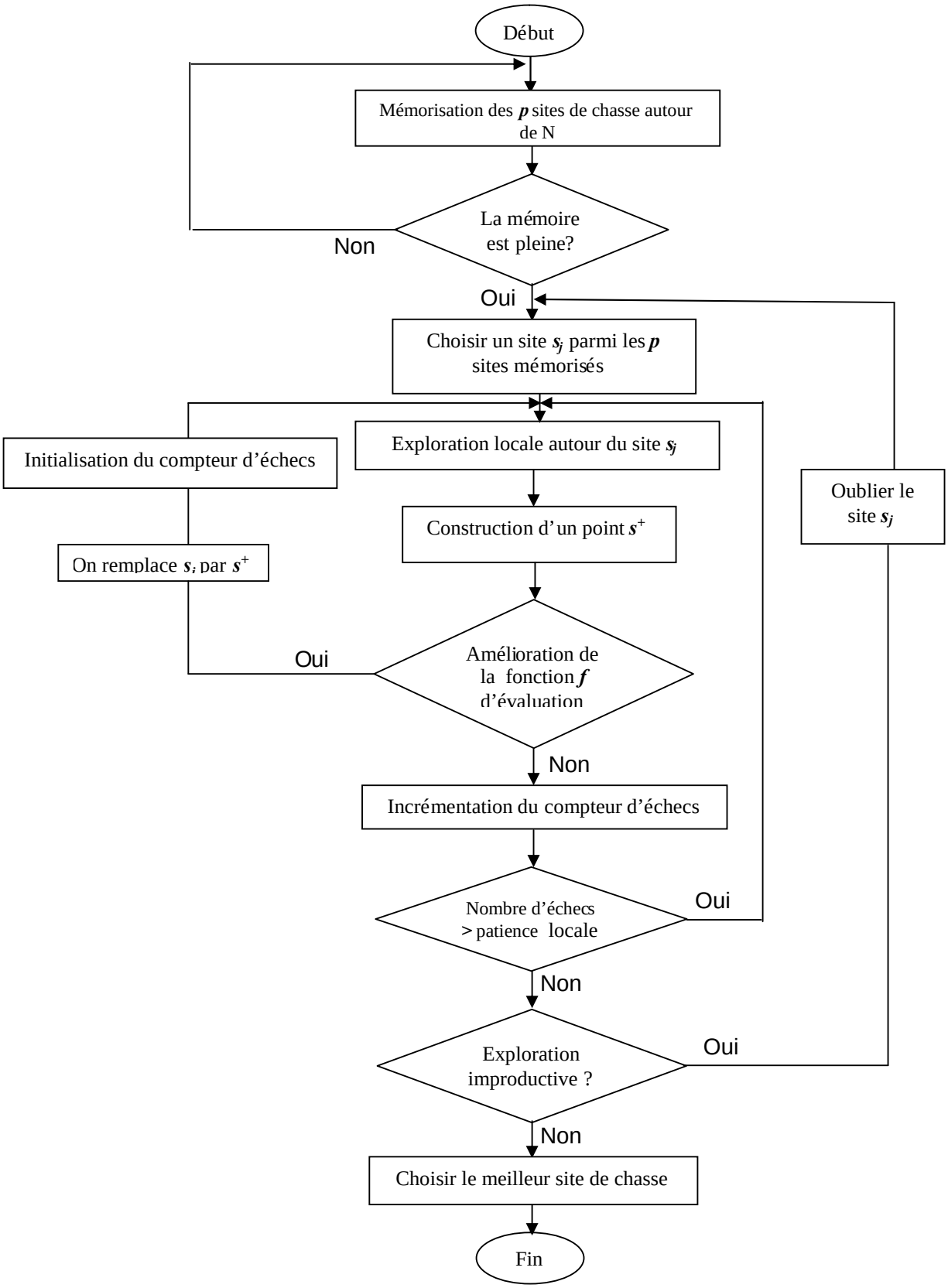
3.3.4. Algorithmes

Les principales étapes de la simulation de la colonie de fourmis *Pachycondyla Apicalis*, sont données par l'algorithme 3.1 suivant :



Algorithme 3.1 : « **API »** Simulation d'une colonie de fourmis *Pachycondyla apicalis*.

Le comportement local de fourragement d'une fourmi a_i est donné par l'algorithme 3.2 suivant :



Algorithme 3.2 : « **API – Fourragement (a_i) » ;** Comportement local d'une fourmi a_i.

3.4. Extensions de l'algorithme API

Cette section présente quelques une des inspirations biologiques supplémentaires qui ont été exploitées pour enrichir l'algorithme API.

3.4.1. Recrutement

Nous avons vu que *Pachycondyla apicalis* ne faisait pas de recrutement pendant le fourragement, cependant, il est possible d'introduire le recrutement en *tandem running* (marche en tandem) dans API [40]. Ainsi, une fourmi qui parvient à améliorer un site, tente de recruter une autre fourmi en lui transmettant la position de son site de chasse. Si elle y parvient, le nombre de recherches locales pour ce site va augmenter. Un paramètre $\boldsymbol{P}_{\mathbf{recrut}}$ représentant la probabilité pour qu'une fourmi tentant un recrutement y parvienne, sera introduit et nous noterons, par la suite, l'algorithme API avec recrutement, l'algorithme **API***r*.

3.4.1.1. Variantes du mode de recrutement

Le recrutement peut être mis en oeuvre de plusieurs façons différentes. Du point de vue de l'optimisation, une fourmi doit recruter une fourmi moins performante [19]. Notons par $\boldsymbol{R}_{xy}$ les différents types de recrutements, ***" x "*** détermine la règle de décision utilisée par une fourmi pour tenter un recrutement et ***" y "*** détermine le choix de la fourmi à recruter. Par exemple, pour ***x*****,** une fourmi peut recruter une autre fourmi, dans les cas suivants :

- si elle améliore son optimum (***x* = 1**) ;
- si elle améliore un site de chasse (***x* = 2**).

Le premier cas suppose que la fourmi possède une mémoire supplémentaire qui lui permette de conserver la position et la valeur du meilleur point trouvé depuis le début de la recherche. La fourmi recrutée peut être choisie de différentes manières suivant la valeur allouée à ***y*** , on prendra, par exemple :

- recrutement au hasard (***y*** = 1) ;
- recrutement de la fourmi ayant le plus mauvais optimum (***y*** = 2) ;
- recrutement de la fourmi ayant le plus mauvais site de chasse (***y*** = 3).

L'avantage de prendre ***y*** =1, est qu'il n'est pas nécessaire de consulter la mémoire de toutes les fourmis pour choisir la candidate au recrutement. Cela représente donc un coût calculatoire moindre.

3.4.2. Population hétérogène

La difficulté de choisir les paramètres de l'algorithme API ainsi définit nous a amenés à envisager deux propositions :

- **Variation des paramètres au cours du temps**. On peut noter que dans la nature, les fourmis n'ont pas la même stratégie de chasse en fonction de leur âge : plus elles sont âgées, plus elles sortent fréquemment du nid et plus elle vont chasser loin de celui-ci, [22]. Une autre façon de voir les choses serait d'utiliser l'influence de l'environnement, par exemple à travers les variations de température. En effet, l'activité des ouvrières est liée à la température extérieure. Par exemple, les *Pachycondyla apicalis* chassent le plus aux heures chaudes de la journée [40]. Dans les deux cas (variation de l'age ou de la température), du point de vue de la modélisation, cela se traduit par une variation des paramètres de chaque fourmi au cours du temps ;
- **Hétérogénéité des paramètres**. Dans la nature, les fourmis ont des caractéristiques différentes les unes des autres, elles sont parfois même regroupées en castes du fait de leurs différences morphologiques. Des modèles de division du travail ont été élaborés pour rendre compte de la capacité dynamique des fourmis à effectuer certaines tâches, [19]. Cette plasticité nous suggère de constituer une population de fourmis ayant des caractéristiques variées.

Nous retenons la dernière proposition. Nous utiliserons cependant un paramétrage statique car la variation d'un paramètre au cours du temps est délicate surtout si elle est adaptative. La population de fourmis sera donc constituée d'individus ayant des paramètres différents. Par la suite, les populations hétérogènes utilisées dans API, que nous noterons API$_h$, se composeront de fourmis dont seules les amplitudes $\boldsymbol{A}_{\mathbf{sit}}$ et $\boldsymbol{A}_{\mathbf{local}}$ varieront d'une fourmi à une autre et seront fixées dans le temps. On notera, cependant qu'une supervision par un algorithme évolutionnaire est très envisageable et peut être utilisée pour fixer les amplitudes $\boldsymbol{A}_{\mathbf{sit}}$ et $\boldsymbol{A}_{\mathbf{local}}$ de chaque fourmi. Nous fixerons les valeurs des deux amplitudes de façon automatique afin de couvrir le plus de combinaisons possibles :

$$\boldsymbol{A}_{sit}(a_1) = \mathrm{A}.\varepsilon^{0}, \ldots\ldots, \boldsymbol{A}_{sit}(a_i) = \mathrm{A}.\varepsilon^{i-1}, \ldots\ldots, \boldsymbol{A}_{sit}(a_n) = \mathrm{A}.\varepsilon^{n-1} \qquad (3.2)$$

Où $\varepsilon = \left(\frac{1}{0.01}\right)^{\frac{1}{n-1}}$ et A est l'amplitude de recherche relative à la première fourmi.

Le paramètre $A_{local}(a_i)$ est fixé pour toutes les fourmis à $0.1.A_{sit}(a_i)$.

La figure 3.6 donne en exemple les valeurs des paramètres A_{sit} et A_{local} pour 10 fourmis.

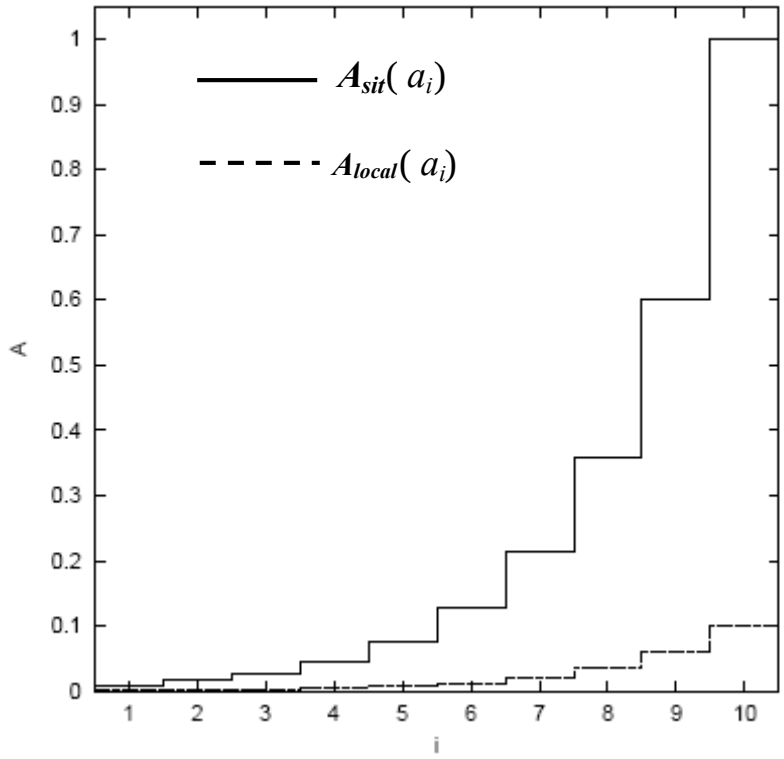


Figure 3.6 : Valeurs des paramètres $A_{sit}(a_i)$ et $A_{local}(a_i)$ pour 10 fourmis.

3.4.3. Prise en compte de la décision de sortir du nid

Un certain nombre de choix ont été arbitrairement faits pour la modélisation des *Pachycondyla apicalis*. Nous n'avons en effet pas pris en compte certaines caractéristiques de cette espèce qui peuvent cependant se révéler intéressantes du point de vue de notre problème d'optimisation. Le comportement prédateur d'une fourmi évolue par exemple en fonction de son âge [22]. Les jeunes fourrageuses (naïves) ont tendance à rester près du nid sans présenter de spécialisation spatiale alors que les fourrageuses expérimentées s'éloignent beaucoup plus du nid tout en montrant une fidélité à une zone de chasse importante. En ce qui concerne la recherche d'un optimum, cela consiste à augmenter l'effort de recherche dans le voisinage du point central. Bien que nous n'avons pas utilisé explicitement de fourmis naïves, la diversité statique introduite dans les paramètres, dans le

cas d'une population hétérogène, reproduit en quelque sorte cette diversification basée sur l'âge.

Un deuxième aspect n'a pas été pris en compte : l'intervalle de temps qui sépare deux sorties est plus court si la fourmi a rencontré un succès à sa première sortie. S'il est probable que ce comportement permette une certaine économie d'énergie au niveau de la colonie en n'allouant des essais qu'aux fourmis les plus chanceuses/efficaces, il n'est pas évident que dans la transposition au problème de l'optimisation cela améliore les performances de l'ensemble. Comme nous considérons que l'évaluation de la fonction objectif représente une contrainte de ressource, nous pouvons proposer une amélioration qui va dans le sens de relier la décision de sortir du nid d'une fourmi a_i à une probabilité $P_s(a_i)$ qui sera modifiée, à chaque itération, de la manière suivante :

$$P_s(a_i) \leftarrow (1-\alpha).P_s(a_i) + \alpha.\delta \qquad (3.3)$$

Où α est un paramètre réel compris dans l'intervalle [0, 1] et commun à toutes les fourmis. δ est une valeur binaire qui vaut 1 si la dernière sortie a été fructueuse et 0 sinon. Au départ, et à chaque déplacement de nid, on fixera P_s à 1 et δ à 0.

L'amélioration de l'algorithme API ainsi envisagée, ne sera pas traitée de ce travail, car on pense qu'il n'est pas évident que dans la transposition aux problèmes de l'optimisation, cette variante produise une amélioration des résultats.

3.5. Les paramètres d'API.

Avant toute application de l'algorithme API, on se doit de fixer les valeurs des différents paramètres qui entrent dans la composition de l'algorithme. Le tableau 3.1 nous donne une liste récapitulative de ces paramètres.

Paramètres	Descriptions
n	nombre de fourmis
A_{sit}	amplitude maximale de création d'un site
A_{local}	amplitude d'exploration locale
P_{local}	patience locale d'une fourmi
P_N	patience du nid
p	nombre de sites mémorisés par une fourmi
P_{recrut}	probabilité de recrutement
O_{rand}	opérateur générant aléatoirement un point s de l'espace
O_{explo}	opérateur générant un point s' au voisinage de s
R_{xy}	type de recrutement

Tableau 3.1 : Tableau récapitulatif des paramètres d'API.

Les paramètres A_{sit} et A_{local} ne sont pas utilises dans la version hétérogène (API_h) où ils sont déterminés par la relation 3.2. Les paramètres R_{xy} et P_{recrut} ne sont utilises que si on autorise le recrutement (API_r). Enfin le paramètre P_N est déterminé automatiquement par la formule 3.1.

3.5.1. Influence de certains paramètres

Il n'existe pas de règle générale qui permet l'allocation d'une valeur à chacun de ces paramètres d'une manière systématique. En littérature, on rencontre plusieurs articles sur l'étude de l'influence des paramètres d'API sur les résultats obtenus [19], mais tous ces travaux sont spécifique à une application donnée et c'est par des tests « essais - résultats » que les paramètres optimaux sont déterminés.
Néanmoins, à la lecture de tous ces travaux, on peut tirer les conclusions suivantes :

- **Nombre de fourmis** n. Dans le cas de problèmes d'optimisation difficiles on peut être tenté de choisir un nombre de fourmis n très grand, cela dans une perspective d'élargir la recherche mais, le nombre d'explorations va augmenter au détriment du nombre d'exploitations. En plus si la population de fourmis est homogène (algorithme API), on risque plus d'avoir un jeu de paramètres mal adapté au problème, ce qui signifie qu'en augmentant le nombre de fourmis n, la capacité de recherche ne sera pas plus étendue.

  Dans le cas d'une population de fourmis hétérogène (algorithme API_h), si le nombre de fourmis est trop restreint, les variations des paramètres dans la population sont

trop rapides et ne permettent pas de couvrir suffisamment de cas. Par contre, similairement à la version homogène, s'il y a beaucoup de fourmis, le nombre d'explorations est trop important par rapport au nombre d'exploitations.

- **Nombre de sites de chasses *p* mémorisé par une fourmi**. Dans les deux cas (API et API$_h$) il semble évident qu'il est inutile d'utiliser trop de sites de chasse. Cela peut s'expliquer par le fait que les fourmis passent trop de temps en exploration (création de sites de chasse) relativement à l'effort d'exploitation (recherche de proie sur un site) quand le nombre de sites de chasse par fourmi est trop grand.
- **Patience locale.** Dans les deux cas (API et API$_h$) il semble qu'une patience locale trop élevée ne soit pas bénéfique, pourquoi explorer un site sans succès un grand nombre de fois ? il serait meilleur de l'oublier et de le remplacer par un nouveau site dans une nouvelle itération.

3.5.2. Opérateurs spécifiques à l'optimisation numérique

Nous avons défini d'une manière générale les opérateurs $\boldsymbol{O}_{\mathbf{rand}}$ et $\boldsymbol{O}_{\mathbf{explo}}$ dans la section 3.3 pour pouvoir les adapter à différents problèmes. Il convient maintenant de les préciser un peu plus.

Les fonctions étudiées sont définies sur un produit d'intervalles de R, chacun délimité par deux bornes $\boldsymbol{b_i}$ et $\boldsymbol{B_i}$. L'opérateur $\boldsymbol{O}_{\mathbf{rand}}$ doit donc générer de façon aléatoire un point de l'espace $\boldsymbol{S} = [\boldsymbol{b_1}, \boldsymbol{B_1}] \times \ldots \times [\boldsymbol{b_l}, \boldsymbol{B_l}]$ où l représente la dimension de la fonction à minimiser. Nous choisissons de générer un point $\boldsymbol{s} = (\boldsymbol{s_i})_{i=1\ldots l} \in (\boldsymbol{S})$ de façon uniforme :

$$s_i = b_i + U[0,1].(B_i - b_i) \tag{3.4}$$

Où U [0, 1] est un nombre aléatoire uniformément tiré dans l'intervalle [0, 1].
L'opérateur d'exploration $\boldsymbol{O}_{\mathbf{explo}}$ tient compte d'une amplitude A :

$$s_i' = s_i + A.U[-0.5,0.5].(B_i - b_i) \qquad \forall i \in \{1,....,l\} \tag{3.5}$$

Où U [-0.5, 0.5] est un nombre aléatoire uniformément tiré dans l'intervalle [-0.5, 0.5]. Evidemment, il faut s'assurer que $s_i' \in$ [b_i, B_i]. Cet opérateur a lui aussi un comportement uniforme dans l'intervalle [$s_i - 0.5.A.(B_i - b_i)$, $s_i + 0.5.A(B_i - b_i)$]. Nous verrons par la suite qu'une distribution normale est une alternative intéressante à cette uniformité.

3.6. Comparaison avec un algorithme génétique

D'une manière générale, l'algorithme API peut être comparé avec les algorithmes manipulant une population de solutions. Les Algorithmes Génétiques (AG) en font partie. Le tableau 7.8 présente une comparaison des AG avec API.

API	AG
n fourmis, ***p*** sites	***n*x*p*** individus
création d'un site	immigration
recherche locale	opérateur de mutation
patience locale	pression de sélection
déplacement du nid	redémarrage « restart »
recrutement	opérateur de sélection

Tableau 3.2 : Comparaison de API avec les techniques issues des Algorithmes Génétiques

Voici points par points les similarités et différences entre API et les techniques issues des algorithmes génétiques (AG) :

- les n fourmis manipulent p sites de chasse. La colonie de fourmis toute entière manipule donc $n \times p$ solutions ;
- la création d'un site correspond à l'arrivée d'une nouvelle solution dans la population. Pour un AG on parle d'immigration ;
- la recherche locale (opérateur $\boldsymbol{O}_{\mathbf{explo}}$) revient à faire une mutation de la solution représentée par le site exploré ;
- la patience locale ($\boldsymbol{P}_{\mathbf{local}}$) indique le nombre de tentatives d'amélioration d'un site. Cela revient à indiquer le niveau de confiance dans l'amélioration d'un site qu'une fourmi peut avoir. Si $\boldsymbol{P}_{\mathbf{local}}$ a une valeur faible, cela signifie que la fourmi sélectionne les meilleures solutions avec peu de tentatives sur chacune d'elles. A l'opposé, une grande valeur de $\boldsymbol{P}_{\mathbf{local}}$ indique que la fourmi dispose d'un plus grand nombre de tentatives pour améliorer une solution. La patience locale permet donc de régler le comportement de la fourmi en fonction des résultats qu'elle obtient. C'est en cela que l'on peut comparer la patience locale à la pression de sélection des algorithmes

génétiques qui permet de favoriser plus ou moins les solutions les plus adaptées. Il faut cependant noter que la patience locale est un paramètre individuel alors que la pression de sélection agit sur l'ensemble de la population

- le déplacement du nid permet de relancer la recherche et éviter de cette façon les optima locaux. Les techniques de « *restart* » visent le même objectif ;
- le recrutement a pour effet de dupliquer les meilleures solutions manipulées par les fourmis ce qui est très similaire à la fonction de l'opérateur de sélection des AG;
- on peut noter l'absence d'un équivalent chez API de l'opérateur de croisement des algorithmes génétiques.

Dans les AG manipulant une population finie de solutions, l'effet cumule de la sélection et du croisement a l'inconvénient de faire disparaître la diversité nécessaire à l'exploration.

En quelque sorte, on peut considérer que dans un AG les solutions sont au départ uniformément distribuées dans l'espace de recherche et qu'après un certain nombre d'itérations, la population se concentre sur les points les plus sélectionnés. Seule la mutation introduit alors une certaine diversité. Dans API, le processus est inversé : les fourmis explorent à partir d'un point central, le nid, et tendent à s'en éloigner au fur et à mesure des itérations.

3.7. Optimisation combinatoire : Problème du voyageur de commerce

Le problème du voyageur de commerce (PVC, *Traveler Salesman Problem* : TSP) est un problème très classique en optimisation combinatoire. Il est de plus en plus improbable de présenter une méthode surpassant, en résultats, toutes celles existantes (voir par exemple, [10] pour un aperçu des différentes approches). Cependant il est intéressant de voir comment API, en tant qu'algorithme d'optimisation générale, peut s'appliquer simplement à ce problème. Comme nous l'avons vu au chapitre 1 de nombreux algorithmes inspirés des fourmis sont nés de l'analogie entre ce problème et la recherche de nourriture par les fourmis [5,6,7]. Nous avons notamment constaté que ces heuristiques adoptent une approche *constructive* des solutions.

L'énoncé image du PVC est le suivant : un voyageur de commerce doit visiter un certain nombre de villes (nœuds) en respectant les règles suivantes :

- il doit visiter chaque ville une seule fois,
- il doit minimiser la distance totale parcourue,

– il revient à sa ville de départ en terminant la tournée.

Ce problème sera représenté donc, par un graphe (ensemble fini de nœuds et un ensemble fini d'arêtes) et le but sera de déterminer le plus court cycle hamiltonien, en supposant que l'on dispose d'une fonction distance, qui à deux nœuds donnés fait correspondre un réel.

La résolution (exacte ou approchée) de ce problème a des répercussions industrielles importantes. En effet, l'optimisation de l'emplacement des composants dans des circuits imprimés, le routage des réseaux de télécommunications en fibres optiques, ou encore l'optimisation du déplacement d'une tête perceuse lors de la fabrication d'un circuit imprimé, figures 3.7, sont autant d'applications qui sont étroitement reliées à la résolution du PVC.

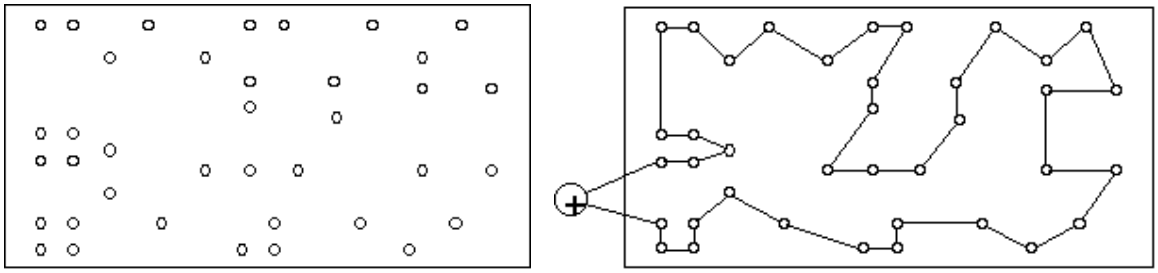


Figure 3.7 : l'optimisation du déplacement d'une tête perceuse lors de la fabrication d'un circuit imprimé

On peut représenter le cycle parcouru par le voyageur de commerce selon un arrangement, en fixant la première des n villes (nœuds) du problème.

L'algorithme consistera à tester les « (n-1)! » arrangements possibles, correspondants aux chemins (solutions) que peut emprunter le voyageur dans le graphe. Par exemple pour 31 villes, l'algorithme devra tester 30! = 265252859812191058636308480000000 solutions, c'est ce qu'on appelle « explosion combinatoire ».

L'adaptation la plus immédiate de API à la résolution d'un tel problème combinatoire est de procéder par la *modification successive d'un ensemble de solutions*, contrairement à une *approche constructive des solutions*. La position d'une fourmi dans l'espace de recherche est équivalente à un *chemin hamiltonien* dans le graphe des villes. Le déplacement d'une fourmi correspond alors à une modification de ce chemin.

Un certain nombre d'heuristiques sont envisageables pour accomplir cette modification. Nous en avons expérimenté quelques unes avec la contrainte qu'elles soient paramétrables par une amplitude et une valeur élevée de cette amplitude permet de modifier assez

fortement une solution (chemin) alors qu'une valeur faible ne doit pas trop perturber le chemin.

La plupart des heuristiques efficaces utilisent des heuristiques proches du problème permettant de modifier un chemin pour en trouver un plus court. L'algorithme ACS, par exemple (voir le chapitre 1), construit des solutions en utilisant la coopération des fourmis puis chaque chemin est amélioré (si possible) par l'heuristique 2-opt qui consiste à prendre deux villes, aléatoirement et on inverse le sens de parcours entre eux. La figure 3.8 illustre cette heuristique.

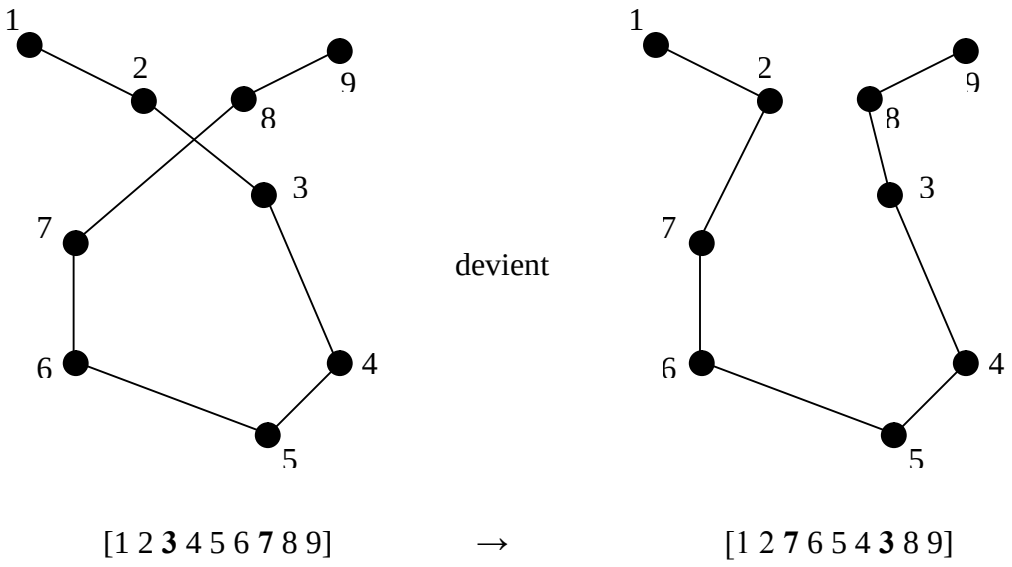


Figure 3.8 : L'heuristique 2-opt améliore le chemin (solution) initiale en permutant les villes 3 et 7.

Dans ce qui va suivre, on présente les techniques de modification d'un chemin, utilisées dans cette application et qui correspondant à l'opérateur O_{explo} de API:

- **la permutation simple** : deux villes sont échangées au hasard dans la séquence. L'amplitude correspond dans ce cas au nombre de permutations effectuées (une amplitude de 0.1 pour un problème à 100 villes occasionnera $0.1 \times 100 = 10$ permutations).
- **la permutation circulaire** : plusieurs villes sont permutées dans la séquence, le nombre de villes permutées représente l'amplitude du déplacement d'une fourmi.

- **l'insertion** : cette méthode consiste à choisir une ville au hasard et à l'insérer entre deux autres villes de la séquence. L'amplitude correspond alors au nombre d'insertions effectuées,
- **permutation probabiliste :** si la permutation de deux villes choisies au hasard dans la séquence augmente la longueur du chemin, la probabilité que cette permutation soit effectuée est inversement proportionnelle à l'augmentation occasionnée. Si la permutation diminue la longueur du chemin, elle est systématiquement effectuée.

Ces méthodes de perturbation d'une séquence ont le mérite d'être *simples* et *peu coûteuses* au niveau du temps de calcul. On peut cependant utiliser des méthodes améliorant localement une solution comme technique de déplacement d'une fourmi, par exemple on peut appliquer une méthode gloutonne sur une portion du chemin. On peut également envisager de différencier la création d'un site de chasse de l'exploration d'un site et associer à ces deux taches des techniques différentes de modification de la séquence.

Résultats.

Afin de tester cette version combinatoire d'API nous nous sommes basés sur un ensemble de jeux de tests très utilises dans le domaine. Les problèmes décrits dans le tableau 3.3 sont tirés de la liste de problèmes du voyageur de commerce TSP-LIB [10].
En ce qui concerne l'opérateur $\boldsymbol{O}_{\mathbf{explo}}$ il est apparu que les meilleurs résultats étaient obtenus en utilisant l'*insertion* comme technique de modification du chemin (à la fois pour la création de site que de l'exploration locale).

Problèmes	Nbre de villes
P16	16
EIL51	51
EIL76	76
KROA100	100
LIN318	318
RAT783	783

Tableau 3.3: Jeux de test pour l'évaluation de API sur le PVC.

En ce qui concerne l'opérateur $\boldsymbol{O}_{\mathbf{explo}}$ il est apparu que les meilleurs résultats étaient obtenus en utilisant l'*insertion* comme technique de modification du chemin (à la fois pour

la création de site que de l'exploration locale). Nous avons comparé API_h (version hétérogène d'API) à ACS afin d'en mesurer les performances en procédant de la maniére suivante :

- Comme ACS utilise une heuristique supplémentaire à chaque création d'un chemin, nous avons utilisé dans les deux cas, pour API et pour ACS, l'heuristique 2-Opt.
- Dans un objectif d'équité, les paramètres de chaque algorithme ont été ajustés pour que le même nombre d'appels à 2-Opt soit effectué.
- Les paramètres utilisés dans ACS sont : n= 20, N = 200, β = 2.0, q_0 = 0.98, ρ = 0.1 et τ_0 = 0.2.
- Les paramètres utilisés dans API_h sont : n = 20 et N = 200, p = 2, P_N = 44.

N étant le nombre d'évaluation de la fonction objectif *f*. Le tableau 3.4 résume les résultats obtenus.

Problèmes	ACS				API_h			
	s_{moy}	σ	s^+	durée	s_{moy}	σ	s^+	durée
P16	16.00	0.00	16	0.62	16.00	0.00	16	0.12
EIL51	434.57	8.75	426	8.37	439.35	9.51	419	0.52
EIL76	553.88	5.35	549	19.44	617.23	13.65	558	1.12
KROA100	43995.4	9598.6	29423	39.28	52238.3	1446.53	41897	2.03
LIN318	48143.1	296.82	46835	786.55	58796.7	3254.32	54876	19.34
RAT783	11323.2	75.72	10580	11724.5	15243.8	724.6	13564	321.4

Tableau 3.4 : Résultats obtenus par ACS et API pour le PVC. s_{moy} est la longueur moyenne du plus court chemin trouvé sur 20 essais et σ l'écart type correspondant. s^+ est la longueur du plus court chemin trouvé et la durée (en secondes) est une moyenne sur les 20 essais.

Les résultats obtenus par API_h sont inférieurs en qualité à ceux obtenus par ACS. Il faut cependant tenir compte des remarques suivantes :

- ACS a été développé spécifiquement pour le PVC et l'optimisation combinatoire ;
- ACS utilise l'information de distance entre les villes comme heuristique locale alors que API n'utilise pas cette information ;

- le temps de calcul de API est très inférieur au temps de calcul de ACS. Par exemple, pour un problème à 783 villes, API nécessite environ 321 secondes alors que ACS en nécessite 11724 pour le même nombre d'itérations ;

Pour accélérer la convergence de l'algorithme ACS pour les problèmes de grandes taille, on pourrait utiliser des listes candidates (élaborées par l'utilisateur) pour réduire la taille du voisinage pour un état donné et réduire ainsi le nombre de mouvement effectués par une fourmi, ce qui réduirait considérablement le temps de construction d'une séquence (solution). Pour améliorer le rendement d'API$_h$, il faudrait que définir l'opérateur $\boldsymbol{O}_{\mathbf{explo}}$ d'une manière plus réfléchi et plus perfectionné que la simple série d'insertions aléatoires qu'on a utilisé dans ce test.

3.8. Conclusion - perspectives

Dans ce chapitre, nous avons présentés l'algorithme de colonies de fourmis API déduit de la modélisation du comportement de fourragement de l'espèce *Pachycondyla apicalis* et son application au problème général de l'optimisation. Un des atouts de cet algorithme est que sa description est relativement simple et indépendante de l'espace de recherche considéré, contrairement aux autres algorithmes de fourmis. Ceci encourage l'utilisation d'API pour la résolution d'une multitude d'autres problèmes et dans divers domaines, on pense principalement aux problèmes d'optimisation continus difficile.

Malheureusement, dans la littérature, très peu de travaux ont été fait dans ce sens et les quelques applications qu'on rencontre se résument à de simples tests élaborés pour l'estimation les performances d'API ou de l'une de ses variantes. Dans le chapitre 4, on se propose de remédier à ce manque en appliquant l'algorithme API à un problème d'optimisation continu complexe « l'identification paramétrique d'une machine asynchrone triphasée ».

Pour les travaux futurs sur API, les idées sont nombreuses et peuvent concerner :

- des études expérimentales sur l'influence des paramètres d'**API** sur les résultats obtenus, ce qui nous emmènera à une formalisation de la détermination systématique des valeurs des paramètres de l'algorithme pour une application donnée.
- des hybridations d'**API** avec d'autres méthodes heuristiques d'optimisation, dans le but, d'accélérer la convergence de l'algorithme. On pourra, par exemple, changer la

règle de décision d'une fourmi lors de la capture d'une proie en utilisant une heuristique secondaire.

- L'utilisation de « modèle multi populations » pour pallier le problème de l'initialisation du nid. En effet les performances de l'algorithme sont fortement dépendantes de cette initialisation. L'idée permettra à l'algorithme de démarrer en différents points de l'espace de recherche S qui sera alors, bien quadrillé. La stratégie consistera à utiliser plusieurs colonies de fourmis, chacune avec un nid positionné initialement à une position différente dans S.
- On peur envisager aussi des adaptations d'API aux problèmes d'optimisation avec contraintes, à l'optimisation multicritère et l'optimisation dynamiques.

CHAPITRE 4
APPLICATION A L'IDENTIFICATION D'UNE MACHINE ASYNCHRONE TRIPHASEE.

4.1. Introduction

La machine asynchrone est la machine électrique la plus utilisée dans le domaine des puissances supérieures, sa robustesse et son coût d'achat et d'entretien lui ont permis de conquérir un espace de plus en plus grand au détriments des machines synchrones et à courant continu. Par ailleurs, les progrès réalisés en matière de commande et les développements technologiques, tant dans le domaine de l'électronique de puissance que celui de la micro électronique, ont rendu possible l'usage de commandes performantes faisant de la machine asynchrone un concurrent potentiel dans les domaines de la vitesse variable. La mise au point de commandes performantes requière une bonne modélisation et une bonne identification de la machine asynchrone. Dans les installations de grandes puissances où les essais directs sont onéreux et difficilement réalisables, la simulation s'impose comme une bonne alternative pour la prédiction des caractéristiques de la machine. Ces caractéristiques peuvent servir à un dimensionnement judicieux des différents éléments constituant le système globale ou à la mise en œuvre d'un système de diagnostique et de surveillance. Dans toutes ces situations, la modélisation et l'identification de la machine ont un impact non négligeable sur la précision des résultats obtenus.

Plusieurs travaux sur l'identification paramétrique des moteurs asynchrone ont été proposés en littérature [41,42,43,44,45,46]. Un grand nombre d'entre eux recourent au calcul, souvent problématique, des gradients de la fonction objectif [41,44,45], ce qui entraîne une amplification des erreurs de calcul et du bruit numériques des mesures. En outre, la plupart de ces méthodes estiment uniquement les paramètres électriques du modèle, les paramètres mécaniques étant déterminés séparément par d'autres techniques telle que la méthode du ralentissement de vitesse de la machine.

Dans ce chapitre, nous mettons en œuvre l'algorithme de colonie de fourmis **API** pour déterminer simultanément les paramètres électriques et mécaniques d'un moteur

asynchrone triphasé à partir du courant de démarrage et de la tension simple correspondante. Nous choisissons la méthode du modèle de référence (erreur de sortie) comme technique d'identification.

4.2. Modélisation de la machine asynchrone.

La représentation schématique de la machine asynchrone triphasée dans l'espace électrique est donnée sur la figure 4.1.

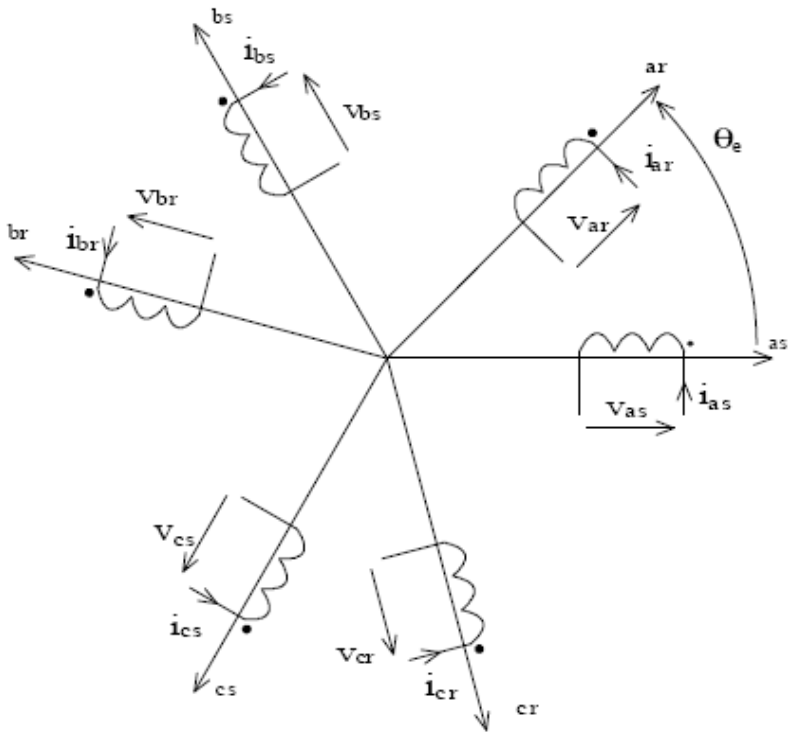


Figure 4.1 : Représentation schématique de la machine.

4.2.1. Hypothèses simplificatrices

Le modèle de la machine que nous adopterons repose sur les hypothèses simplificatrices habituelles suivantes :

- On suppose les circuits magnétiques non saturés et suffisamment feuilletés pour que les pertes de fer soient négligeables ; d'où proportionnalité des flux aux courants.
- Entrefer constant (pas d'effet d'encoches)
- On suppose que le bobinage est répartie de manière à donner une force magnétomotrice (f.m.m) sinusoïdale s'il est alimenté par des courants sinusoïdaux.

- Courants autres que dans les bobinages négligés.
- Symétrie ternaire de la machine.

4.2.2. Equations électrocinétiques

La loi de Faraday et la loi d'ohm permettent de relier les tensions sur les enroulements aux flux totalisés et aux courants dans ces bobinages.
Avec les conventions utilisées, les deux équations matricielles suivantes expriment les tensions sur les différents enroulements.

$$[v_s] = R_s.[i_s] + \frac{d[\phi_s]}{dt} \tag{4.1}$$

$$[v_r] = R_s.[i_r] + \frac{d[\phi_r]}{dt} \tag{4.2}$$

Les indices s et r désignent respectivement les grandeurs relatives au stator et au rotor.

$$[v_s] = \begin{bmatrix} v_{as} \\ v_{bs} \\ v_{cs} \end{bmatrix} \quad ; \quad [\Phi_s] = \begin{bmatrix} \varphi_{as} \\ \varphi_{bs} \\ \varphi_{cs} \end{bmatrix} \quad ; \quad [i_s] = \begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix} \quad . \tag{4.3}$$

$$[v_r] = \begin{bmatrix} v_{ar} \\ v_{br} \\ v_{cr} \end{bmatrix} \quad ; \quad [\Phi_r] = \begin{bmatrix} \varphi_{ar} \\ \varphi_{br} \\ \varphi_{cr} \end{bmatrix} \quad ; \quad [i_r] = \begin{bmatrix} i_{ar} \\ i_{br} \\ i_{cr} \end{bmatrix} \quad . \tag{4.4}$$

4.2.3. Expression des flux totalisés en fonction des courants

L'absence de saturation et la limitation au premier harmonique d'espace nous permettent d'écrire les expressions des flux statorique et rotorique comme suit :

$$\begin{pmatrix} [\Phi_s] \\ [\Phi_r] \end{pmatrix} = \begin{pmatrix} [L_s] & [M_{sr}] \\ [M_{rs}] & [L_r] \end{pmatrix} \begin{pmatrix} [i_s] \\ [i_r] \end{pmatrix} \tag{4.5}$$

Les quatre sous matrices d'inductance s'écrivent :

$$[M_{rs}] = [M_{sr}] = m_{sr} \cdot \begin{bmatrix} \cos\theta_e & \cos\left(\theta_e + \frac{2\pi}{3}\right) & \cos\left(\theta_e - \frac{2\pi}{3}\right) \\ \cos\left(\theta_e - \frac{2\pi}{3}\right) & \cos\theta_e & \cos\left(\theta_e + \frac{2\pi}{3}\right) \\ \cos\left(\theta_e + \frac{2\pi}{3}\right) & \cos\left(\theta_e - \frac{2\pi}{3}\right) & \cos\theta_e \end{bmatrix} \quad (4.6)$$

$$[L_s] = \begin{bmatrix} l_s & m_s & m_s \\ m_s & l_s & m_s \\ m_s & m_s & l_s \end{bmatrix} \qquad [L_r] = \begin{bmatrix} l_r & m_r & m_r \\ m_r & l_r & m_r \\ m_r & m_r & l_r \end{bmatrix} \quad (4.7)$$

Les différentes inductances désignent:

l_s : inductance propre d'une phase statorique,
l_r : inductance propre d'une phase rotorique,
m_s : inductance mutuelle entre phases statoriques,
m_r : inductance mutuelle entre phases rotoriques,
m_{sr} : inductance mutuelle entre stator et rotor,
θ_e : écart angulaire entre stator et rotor.

4.2.4. Equation du couple

L'expression du couple électromagnétique peut être obtenue à partir de la dérivée de la co-énergie magnétique par rapport à l'angle θ_e. La machine étant à structure lisse, le couple s'écrit :

$$C_{cm} = [i_s]^t . \frac{\partial [M_{sr}(\theta_e)]}{\partial \theta_e} . [i_r] \quad (4.8)$$

Où i_s et i_r désignent respectivement les courants statoriques et rotoriques.

4.2.5. Transformation triphasée – diphasée

Le but de l'utilisation de cette transformation est de passer d'un système triphasé *abc* vers un système diphasé *αβ*. On distingue deux méthodes de transformation, la méthode de Clarke et celle de Concordia. C'est la transformation de Concordia qui est utilisée dans notre modélisation.

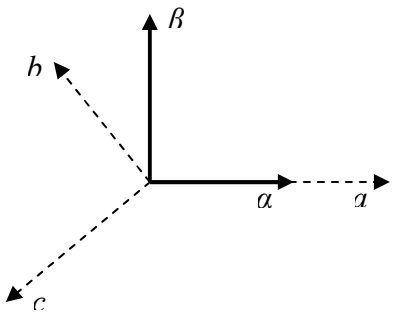
La méthode de Concordia est illustrée dans le tableau ci-dessous pour un passage d'un système triphasé *abc* vers un système diphasé *αβ* et inversement.

Transformation de Concordia

Passage d'un système triphasé *abc* vers un système diphasé *αβ*

$$\begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \xrightarrow{T_{23}} \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} \text{ c-à-d } \quad [x_{\alpha\beta}] = T_{23}.[x_{abc}]$$

Avec $T_{23} = \sqrt{\frac{2}{3}}.\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$



Passage d'un système diphasé *αβ* vers un système triphasé *abc*

$$\begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} \xrightarrow{T_{32}} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \text{ c-à-d } \quad [x_{abc}] = T_{32}.[x_{\alpha\beta}] \quad \text{avec } T_{23} = \sqrt{\frac{2}{3}}.\begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

Son application aux équations de la machine nous donne :

$$T_{23}[\mathrm{v_s}]_{abc} = T_{23}\left\{ \mathrm{R_s}.[\mathrm{i_s}]_{abc} + \frac{d[\Phi_s]_{abc}}{dt} \right\} = [\mathrm{v_s}]_{\alpha\beta} \tag{4.9}$$

$$[\mathrm{v_s}]_{\alpha\beta} = \left\{ \mathrm{R_s}.T_{23}.[\mathrm{i_s}]_{abc} + \frac{d\,T_{23}.[\Phi_s]_{abc}}{dt} \right\} = \mathrm{R_s}.[\mathrm{i_s}]_{\alpha\beta} + \frac{d}{dt}[\Phi_s]_{\alpha\beta} \tag{4.10}$$

Ainsi on aura réduit le système de 3 équations dans le repère *abc* à un système à 2 équations dans le repère *αβ*. De même pour le rotor,

$$[\mathrm{v_r}]_{\alpha\beta} = \left\{ \mathrm{R_r}.[\mathrm{i_r}]_{\alpha\beta} + \frac{d}{dt}[\Phi_r]_{\alpha\beta} \right\} \tag{4.11}$$

ainsi que pour l'écriture des flux en fonction des courants. L'intérêt pour le flux, c'est que les matrices 3x3 des inductances vont être réduites à des matrices 2x2. On aura alors l'apparition des inductances cycliques :

$$\begin{cases} L_s = l_s - m_s \\ L_r = l_r - m_r \\ L_m = \frac{3}{2}.m_{sr} \end{cases} \tag{4.12}$$

et .

$$\begin{bmatrix} [\Phi_s]_{\alpha\beta} \\ [\Phi_r]_{\alpha\beta} \end{bmatrix} = \left(\begin{array}{cc:cc} L_s & 0 & \multicolumn{2}{c}{L_m \,.\, P(\theta_e)} \\ 0 & L_s & \multicolumn{2}{c}{} \\ \hdashline \multicolumn{2}{c:}{L_m \,.\, P(-\theta_e)} & L_r & 0 \\ \multicolumn{2}{c:}{} & 0 & L_r \end{array} \right) \begin{bmatrix} [i_s] \\ [i_r] \end{bmatrix} \tag{4.13}$$

où la matrice $P(\theta_e)$ est une matrice de rotation :

$$P(\theta_e) = \begin{bmatrix} \cos\theta_e & -\sin\theta_e \\ \sin\theta_e & \cos\theta_e \end{bmatrix} \tag{4.14}$$

On dispose à présent d'une modélisation de la machine asynchrone dans 2 repères séparés, les grandeurs statoriques sont exprimées dans le repère $\alpha\beta$ stator et les grandeurs rotoriques le sont dans le repère $\alpha\beta$ rotor. Il faut exprimer toute la modélisation dans un repère commun. En effet si on examine de plus prés la matrice des inductances :

$$\left(\begin{array}{cc:cc} L_s & 0 & \multicolumn{2}{c}{L_m \,.\, P(\theta_e)} \\ 0 & L_s & \multicolumn{2}{c}{} \\ \hdashline \multicolumn{2}{c:}{L_m \,.\, P(-\theta_e)} & L_r & 0 \\ \multicolumn{2}{c:}{} & 0 & L_r \end{array} \right) \tag{4.15}$$

on voit que les grandeurs statoriques sont liées aux grandeurs rotoriques à travers θ_e.

On choisi alors de transformer les grandeurs statoriques et les grandeurs rotoriques vers un repère commun dit "*dq*" et ceci à l'aide de deux transformations dans le plan qui sont des

rotations. Ce sont ces deux transformations auxquelles on ajoute la transformation de Concordia qui constitue la transformation de Park.

4.2.6. Transformation de Park

La transformation de Park est constituée d'une transformation triphasé – diphasé suivie d'une rotation. Elle permet de passer du repère *abc* vers le repère $\alpha\beta$ puis vers le repère *dq*. Le repère $\alpha\beta$ est toujours fixe par rapport au repère *abc*, par contre le repère *dq* est mobile. Il forme avec le repère fixe $\alpha\beta$ un angle appelé «angle de Park».

Si l'on note par θ_s et θ_r les angles de la transformation de Park des grandeurs statoriques et rotoriques, il existe une relation qui lie et simplifie les équations du modèle.
Les repères de la transformation de Park des grandeurs statoriques et rotoriques doivent coïncider pour une plus grande simplification des équations (figure 4.2). On prendra :

$$\theta_s = \theta_e + \theta_r \tag{4.16}$$

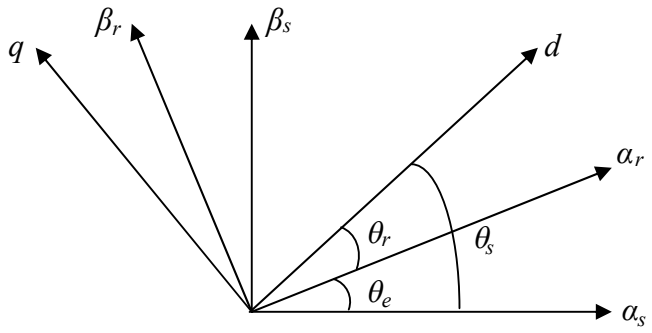


Figure 4.2 : Transformation de Park

Les grandeurs statoriques et rotoriques seront transformées de la manière suivante :

$$\begin{cases} [x_s]_{\alpha\beta} = P(\theta_s).[x_s]_{dq} \\ [x_r]_{\alpha\beta} = P(\theta_r).[x_r]_{dq} \end{cases} \tag{4.17}$$

D'où, les équations aux tensions :

$$\begin{cases} [v_s]_{dq} = R_s.[i_s]_{dq} + \dot{\theta}_s.P(\frac{\pi}{2}).[\Phi_s]_{dq} + \frac{d}{dt}[\Phi_s]_{dq} \\ [v_r]_{dq} = R_r.[i_r]_{dq} + \dot{\theta}_r.P(\frac{\pi}{2}).[\Phi_r]_{dq} + \frac{d}{dt}[\Phi_r]_{dq} \end{cases} \quad (4.18)$$

qu'on écrira désormais sous les formes :

$$\begin{cases} [v_{dqs}] = R_s.[i_{dqs}] + \dot{\theta}_s.P(\frac{\pi}{2}).[\Phi_{dqs}] + \frac{d}{dt}[\Phi_{dqs}] \\ [v_{dqr}] = R_r.[i_{dqr}] + \dot{\theta}_r.P(\frac{\pi}{2}).[\Phi_{dqr}] + \frac{d}{dt}[\Phi_{dqr}] \end{cases} \quad (4.19)$$

$\dot{\theta}_s$ et $\dot{\theta}_r$ sont respectivement les dérivées des angles des transformations de Park des grandeurs statoriques et rotoriques.

Cependant, c'est au niveau de l'écriture des flux que la méthode montre toute son efficacité :

$$\begin{bmatrix} [\Phi_s]_{dq} \\ [\Phi_r]_{dq} \end{bmatrix} = \begin{bmatrix} [\Phi_{dqs}] \\ [\Phi_{dqr}] \end{bmatrix} = \left(\begin{array}{cc:cc} L_s & 0 & L_m & 0 \\ 0 & L_s & 0 & L_m \\ \hdashline L_m & 0 & L_r & 0 \\ 0 & L_m & 0 & L_r \end{array}\right) \begin{bmatrix} [i_{dqs}] \\ [i_{dqr}] \end{bmatrix} \quad (4.20)$$

En effet, les sous matrices sont maintenant diagonales et ne dépendent plus de l'angle électrique θ_e entre le stator et le rotor.

Le système matriciel peut aussi être écrit sous la forme suivante :

$$\begin{cases} \Phi_{ds} = L_s.i_{ds} + L_m.i_{dr} \\ \Phi_{qs} = L_s.i_{qs} + L_m.i_{qr} \\ \Phi_{dr} = L_r.i_{dr} + L_m.i_{ds} \\ \Phi_{qr} = L_r.i_{qr} + L_m.i_{qs} \end{cases} \quad (4.21)$$

et les tensions, sous la forme :

$$\begin{cases} v_{ds} = R_s.i_{ds} - \omega_s.\Phi_{qs} + \dfrac{d\Phi_{ds}}{dt} \\ v_{qs} = R_s.i_{qs} + \omega_s.\Phi_{ds} + \dfrac{d\Phi_{qs}}{dt} \\ v_{dr} = R_r.i_{dr} - \omega_r.\Phi_{qr} + \dfrac{d\Phi_{dr}}{dt} \\ v_{qr} = R_r.i_{qr} + \omega_r.\Phi_{dr} + \dfrac{d\Phi_{qr}}{dt} \end{cases} \quad (4.22)$$

Où $\omega_s = \dot{\theta}_s$ et $\omega_r = \dot{\theta}_r$ sont appelés respectivement, *pulsation statorique* et *pulsation rotorique.* $\omega = \omega_s - \omega_r$ est la pulsation mécanique.

Dans le modèle de Park, l'expression du couple s'écrit de la manière suivante :

$$C_e = p.L_m.(i_{qs}.i_{dr} - i_{ds}.i_{qr}) \quad (4.23)$$

Selon l'application envisagée, le repère *dq* peut être lié au stator ($\omega_s = 0$), au rotor ($\omega_r = 0$) ou au champ tournant (champ crée par le bobinage statorique et qui tourne, en régime permanent, à la vitesse de synchronisme).

4.2.7. Modèle de la machine à cinq paramètres électrique

Dans un repère lié au stator, les équations électriques de la machine sont déduites des équations (21) et (22) dans lesquelles $\omega_s = 0$. En ajoutant l'équation mécanique, la machine sera entièrement décrite par le système suivant :

$$\begin{cases} v_{ds} = R_s.i_{ds} + L_s.\dfrac{di_{ds}}{dt} + L_m.\dfrac{di_{dr}}{dt} \\ v_{qs} = R_s.i_{qs} + L_s.\dfrac{di_{qs}}{dt} + L_m.\dfrac{di_{qr}}{dt} \\ v_{dr} = R_r.i_{dr} + L_r.\dfrac{di_{dr}}{dt} + L_m.\dfrac{di_{ds}}{dt} + p.\Omega.L_r.i_{qr} + p.\Omega.L_m.i_{qs} \\ v_{qr} = R_r.i_{qr} + L_r.\dfrac{di_{qr}}{dt} + L_m.\dfrac{di_{qs}}{dt} - p.\Omega.L_r.i_{dr} - p.\Omega.L_m.i_{ds} \\ J.\dfrac{d\Omega}{dt} = p.L_m.(i_{qs}.i_{dr} - i_{ds}.i_{qr}) - f_r.\Omega.C_s \end{cases} \quad (4.24)$$

En réarrangeant les équations précédentes de façon à faire apparaître le vecteur d'état $\left[\mathrm{i}_{ds} \quad \mathrm{i}_{qs} \quad \mathrm{i}_{dr} \quad \mathrm{i}_{qr} \quad \Omega\right]^t$, nous obtenons le système d'équations suivant :

$$\begin{cases} \dfrac{d\mathrm{i}_{ds}}{dt} = -\dfrac{L_r.\mathrm{R}_s}{A}.\mathrm{i}_{ds} + \dfrac{p.\Omega.L_m^2}{A}.\mathrm{i}_{qs} + \dfrac{L_m.\mathrm{R}_r}{A}.\mathrm{i}_{dr} + \dfrac{p.\Omega.L_m.L_r}{A}.\mathrm{i}_{qr} + \dfrac{L_r.\mathrm{v}_{ds}}{A} \\ \dfrac{d\mathrm{i}_{qs}}{dt} = -\dfrac{p.\Omega.L_m^2}{A}.\mathrm{i}_{ds} - \dfrac{L_r.\mathrm{R}_s}{A}.\mathrm{i}_{qs} - \dfrac{p.\Omega.L_m.L_r}{A}.\mathrm{i}_{dr} + \dfrac{L_m.\mathrm{R}_r}{A}.\mathrm{i}_{qr} + \dfrac{L_r.\mathrm{v}_{qs}}{A} \\ \dfrac{d\mathrm{i}_{dr}}{dt} = \dfrac{L_m.\mathrm{R}_s}{A}.\mathrm{i}_{ds} - \dfrac{p.\Omega.L_m.L_s}{A}.\mathrm{i}_{qs} - \dfrac{L_s.\mathrm{R}_r}{A}.\mathrm{i}_{dr} + \dfrac{p.\Omega.L_s.L_r}{A}.\mathrm{i}_{qr} + \dfrac{L_m.\mathrm{v}_{ds}}{A} \\ \dfrac{d\mathrm{i}_{qr}}{dt} = \dfrac{p.\Omega.L_m.L_s}{A}.\mathrm{i}_{ds} + \dfrac{L_m.\mathrm{R}_s}{A}.\mathrm{i}_{qs} + \dfrac{p.\Omega.L_s.L_r}{A}.\mathrm{i}_{dr} - \dfrac{L_s.\mathrm{R}_r}{A}.\mathrm{i}_{qr} - \dfrac{L_m.\mathrm{v}_{qs}}{A} \\ \dfrac{d\Omega}{dt} = \dfrac{p.L_m}{J}.(\mathrm{i}_{qs}.\mathrm{i}_{dr} - \mathrm{i}_{ds}.\mathrm{i}_{qr}) - \dfrac{f_r.\Omega.C_s}{J} \end{cases} \tag{4.25}$$

Où $A = L_s.L_r - L_m^2$

Ce système d'équation montre que le fonctionnement de la machine asynchrone dépend de cinq paramètres électriques [R_s R_r L_s L_r L_m] et de trois paramètres mécaniques [J f_r C_s].

Le rotor d'une machine asynchrone à cage n'est pas accessible directement aux mesures expérimentales, il est impossible de mesurer séparément les valeurs de R_r, L_r et L_m. Dans la section suivante, on va procéder à un changement de variables du vecteur d'état, pour réduire notre modèle de la machine et nous ramené à un modèle à quatre paramètres électrique au lieux des cinq précédent.

4.2.8. Modèle de la machine à quatre paramètres électrique

Les enroulements rotoriques étant en court-circuit, on prendra en fonctionnement normal de la machine, v_{dr} et v_{qr} nulles.

Procédons aux changements de variables suivants :

$$i_{dr} = \frac{L_m}{L_r}.i'_{dr} \qquad \text{et} \qquad i_{qr} = \frac{L_m}{L_r}.i'_{qr}$$

et introduisons :

- le coefficient de dispersion $\sigma = 1 - \frac{L_m^2}{L_s.L_r}$,
- les constantes de temps, rotorique $T_r = \frac{L_r}{R_r}$ et statorique $T_s = \frac{L_s}{R_s}$.

Les équations (24) deviennent :

$$\begin{cases} v_{ds} = R_s.i_{ds} + L_s.\frac{di_{ds}}{dt} + (1-\sigma).L_s.\frac{di'_{dr}}{dt} \\ v_{qs} = R_s.i_{qs} + L_s.\frac{di_{qs}}{dt} + (1-\sigma).L_s.\frac{di'_{qr}}{dt} \\ v_{dr} = \frac{1}{T_r}i'_{dr} + \frac{di'_{dr}}{dt} + \frac{di_{ds}}{dt} + p.\Omega.i'_{qr} + p.\Omega.i_{qs} \\ v_{qr} = \frac{1}{T_r}i'_{qr} + \frac{di'_{qr}}{dt} + \frac{di_{qs}}{dt} - p.\Omega.i'_{dr} - p.\Omega.i_{ds} \\ J.\frac{d\Omega}{dt} = p.(1-\sigma).L_s.(i_{qs}.i'_{dr} - i_{ds}.i'_{qr}) - f_r.\Omega. - C_s \end{cases} \quad (4.26)$$

Et, en faisant apparaître le vecteur d'état $\left[i_{ds} \quad i_{qs} \quad i_{dr} \quad i_{qr} \quad \Omega\right]^t$, nous obtenons le système d'équations suivant :

$$\begin{cases} \frac{di_{ds}}{dt} = -\frac{1}{\sigma.T_s}.i_{ds} + \frac{(1-\sigma).p.\Omega}{\sigma}.i_{qs} + \frac{1-\sigma}{\sigma.T_r}.i'_{dr} + \frac{(1-\sigma).p.\Omega}{\sigma}.i'_{qr} + \frac{v_{ds}}{\sigma.L_s} \\ \frac{di_{qs}}{dt} = -\frac{(1-\sigma).p.\Omega}{\sigma}.i_{ds} - \frac{1}{\sigma.T_s}.i_{qs} - \frac{(1-\sigma).p.\Omega}{\sigma}.i'_{dr} + \frac{1-\sigma}{\sigma.T_r}.i'_{qr} + \frac{v_{qs}}{\sigma.L_s} \\ \frac{di'_{dr}}{dt} = \frac{1}{\sigma.T_s}.i_{ds} - \frac{p.\Omega.}{\sigma}.i_{qs} - \frac{1}{\sigma.T_r}.i'_{dr} - \frac{p\Omega}{\sigma}.i'_{qr} - \frac{v_{ds}}{\sigma.L_s} \\ \frac{di'_{qr}}{dt} = \frac{p.\Omega}{\sigma}.i_{ds} + \frac{1}{\sigma.T_s}.i_{qs} + \frac{p.\Omega}{\sigma}.i'_{dr} - \frac{1}{\sigma.T_r}.i'_{qr} - \frac{v_{qs}}{\sigma.L_s} \\ \frac{d\Omega}{dt} = \frac{p}{J}.(1-\sigma).L_s.(i_{qs}.i'_{dr} - i_{ds}.i'_{qr}) - \frac{f_r.\Omega - C_s}{J} \end{cases} \quad (4.27)$$

C'est ce modèle de la machine asynchrone triphasé qui sera utilisé dans la suite de nos travaux. Ce modèle déterminé dans un repère lié au stator est, en effet bien adapté à

l'identification paramétrique, car il permet de s'affranchir de la mesure de la position du rotor et donc du capteur de position.

La machine est donc, décrite par un système non linéaire en Ω et elle est parfaitement caractérisée par un vecteur Π constitué de quatre paramètres électrique (σ, T_s, L_s et T_r) et de trois paramètres mécaniques (J, f_r et C_s).

$$\Pi = \begin{bmatrix} \sigma & T_s & L_s & T_r & J & f_r & C_s \end{bmatrix}$$

Avec :

σ : Coefficient de fuite.
T_s : Constante de temps statorique.
L_s : Self inductance statorique.
T_r : Constante de temps rotorique.
J : Masse d'inertie de rotor.
f_r : Coefficient de frottement.
C_s : Constante de moment de torsion.

4.3. Techniques d'identification

On définit l'identification ou modélisation expérimentale comme la procédure permettant la détermination de la représentation mathématique (ou modèle) d'un système à partir de résultats expérimentaux.

L'identification paramétrique repose donc sur la notion de modèle. On distingue deux classes de modèles :

- dans le cas de modèle par équations aux différences dont les paramètres et la structure peuvent avoir perdu tout lien avec la réalité physique, on parle communément de modèles du type boîte noire,
- à l'opposé, lorsque le modèle est de type continu par équation différentielle (ou système différentiel) dont les paramètres et la structure peuvent approcher la réalité physique, on parle dans ce cas de modèles de connaissance ou plus raisonnablement de modèles du type boîte grise. En effet, il est illusoire de vouloir décrire exhaustivement la réalité physique et on n'utilise en fait que des modèles approchés ou d'ordre réduit, d'où le terme de boîte grise qui correspond mieux à la modestie de nos ambitions.

Depuis la généralisation de la commande numérique des systèmes, l'automaticien s'est surtout intéressé à la modélisation et à l'identification des processus sous forme discrète. Cette représentation se prête bien à la commande numérique des systèmes. Cependant,

lorsqu'on s'intéresse à la réalité physique du système, comme c'est le cas en électrotechnique, pour la commande ou la surveillance des machines, l'utilisateur préfère utiliser un modèle proche de la réalité, dont les paramètres (résistances, inductances, ...) ont une signification tangible ; la représentation discrète ne peut donner, dans ce cas, une entière satisfaction. On retrouve la même attitude en génie des procédés ou en génie chimique (pour les constantes cinétiques de réaction chimique par exemple) ainsi qu'en robotique (pour des paramètres tels que masses et raideurs).

Aussi, ce travail est dédié à l'estimation paramétrique en génie électrique à partir de modèles à représentation continue, proches de la réalité physique. Les algorithmes d'identification qui en découleront pourront être utilisés pour l'observation d'état, la synthèse de lois de commande adaptative ou la surveillance des machines électriques, par suivi de l'estimation paramétrique.

Dans ce contexte, on va essentiellement s'intéresser à l'identification des systèmes à l'aide de modèles à représentation continue. Deux catégories d'algorithmes sont utilisables, que l'on classe suivant la nature des résidus en erreur d'équation ou en erreur de sortie.

Les algorithmes a erreur d'équation ne sont utilisables en pratique qu'avec des modèles du type équation différentielle à coefficients constants. Pour de tels modèles, de nombreuses techniques [47] ont été imaginées afin d'exprimer la sortie du système linéairement par rapport à ses paramètres (L.P). Cette propriété de linéarité permet alors d'utiliser la méthode des moindres carrés dont l'intérêt essentiel est de fournir une expression analytique de l'estimée des paramètres [48]. Malheureusement, on démontre que pour tout modèle L.P dont le régresseur dépend directement (ou indirectement par filtrage) des valeurs de la sortie, les résidus sont du type erreur d'équation et en conséquence l'estimateur est biaisé. Une solution à l'élimination de ce biais est d'utiliser une technique de variable instrumentale à modèle parallèle simulé [48], mais cela complique bien sûr l'algorithme, et la convergence peut dans certains cas poser un problème. Enfin, comme en général, les machines électriques ne se ramènent pas à des équations différentielles à coefficients constants mais plutôt à des systèmes différentiels non linéaires, on comprendra que cette méthodologie d'identification ne soit pas vraiment adaptée au problème envisagé. Toutefois, ces méthodes ne doivent pas être rejetées. En effet, bien que leurs estimations soient critiquables, elles sont néanmoins précieuses car elles peuvent servir à initialiser les méthodes à erreur de sortie. Le lecteur intéresse par ce sujet pourra se reporter à quelques présentations synthétiques et en particulier à une étude comparative de ces méthodes d'identification [47] afin de mieux apprécier un certain nombre de critères de choix. Ce

chapitre va donc être dédie à la présentation de la deuxième catégorie d'algorithmes, du type erreur de sortie ; qu'on appelle aussi méthode du modèle.

4.3.1. Algorithme d'identification du type erreur de sortie

Ces algorithmes sont connus sous l'appellation générique de méthode du modèle [49]. Ils se caractérisent fondamentalement par la simulation de la sortie à partir de la seule connaissance de l'excitation et du modèle (alors que les algorithmes du type erreur d'équation sont basés sur une prédiction de la sortie, à partir de la connaissance de l'excitation et des valeurs passées de la sortie). Grâce à cette procédure, la sortie simulée est indépendante de la perturbation affectant le système (s'il n'y a pas de bouclage) ; en conséquence, les résidus sont l'image de cette perturbation, d'où l'appellation d'erreur de sortie et d'intéressantes propriétés de convergence. Par contre, cette simulation complique le problème de minimisation du critère qui nécessite alors l'utilisation des techniques d'optimisation non linéaire.

4.3.1.1. Principe de la méthode à erreur de sortie

La méthodologie générale de mise en œuvre, communément appelée méthode du modèle, peut être symbolisée par la figure (4.3).

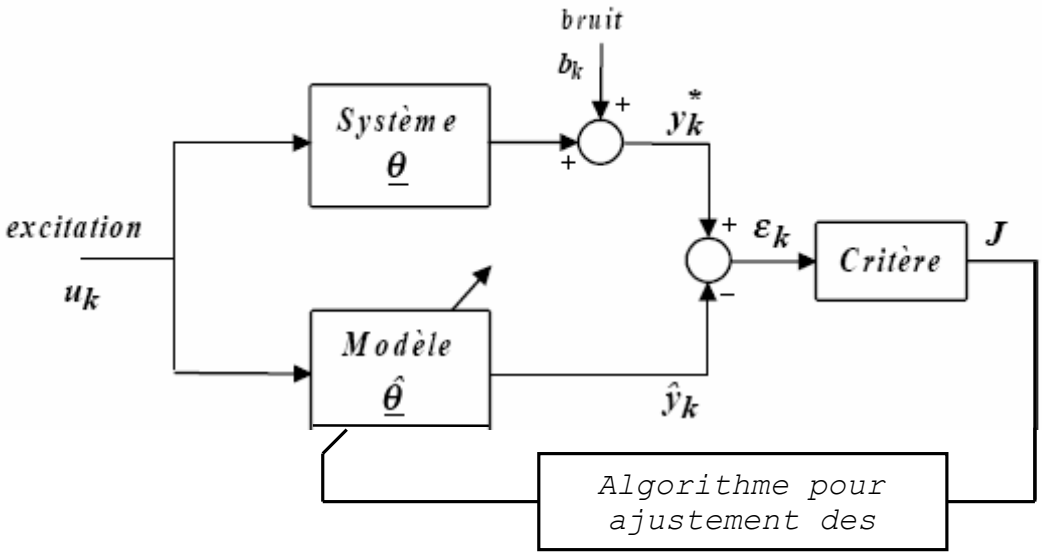


Figure 4.3 : Principe des méthodes à erreur de sortie

Considérons un système décrit par le modèle d'état général d'ordre *n* décrivant la réponse *y*(*t*) à l'excitation *u*(*t*), dépendant du vecteur paramètres $\underline{\theta}$:

$$\begin{cases} \dot{\underline{x}} = \underline{g}\,(\underline{x}, \underline{\theta}, u) \\ y = f\,(\underline{x}, \underline{\theta}, u) \end{cases} \qquad \text{avec} \qquad \begin{cases} dim(\underline{x}) = n \\ dim(\underline{\theta}) = N \end{cases}$$

Où *y*(*t*) et *u*(*t*) sont considérés monodimensionnels uniquement pour simplifier la présentation. On remarquera qu'aucune hypothèse de linéarité n'est nécessaire : $\underline{g}$ et f sont des lois issues d'un raisonnement physique, qui en général ne sont pas linéaires. On fera cependant l'hypothèse que le système est identifiable.

Considérons par ailleurs un ensemble de K données expérimentales $\{u_k, y_k^*\}$, acquises avec la période d'échantillonnage T_e, telle que t = K T_e ; le problème de l'identification est alors d'estimer le modèle qui explique au mieux ces données, donc de déterminer la valeur des paramètres du vecteur θ.

Soit $\hat{\underline{\theta}}$ une estimation de $\underline{\theta}$. Alors grâce à *u*(*t*), connue aux instants d'échantillonnage u_k, on obtient une simulation $\hat{y}_k$ de la sortie, soit :

$$\begin{cases} \dot{\hat{\underline{x}}} = \underline{g}\left(\hat{\underline{x}}, \hat{\underline{\theta}}, u\right) \\ \hat{y} = f\left(\hat{\underline{x}}, \hat{\underline{\theta}}, u\right) \end{cases} \tag{4.28}$$

On définit l'erreur de prédiction (résidu) notée ε_k entre la sortie réelle y_k^* et la sortie simulée $\hat{y}_k$

$$\varepsilon_k = y_k^* - \hat{y}(u_k, \hat{\underline{\theta}}) \tag{4.29}$$

Avec :

- $y_k^* = y_k + b_k$: mesure de la sortie y_k , perturbée par un bruit,
- y_k : valeur exacte de la sortie,
- b_k : perturbation aléatoire,
- ε_k : résidu.

4.3.1.2. Critère de similitude

L'estimation des paramètres de la machine asynchrone à l'aide de la méthode du modèle de référence (d'erreur de sortie), c'est-à-dire l'approximation des valeurs expérimentales

par un modèle mathématique, repose sur le choix d'un critère J objectif qualifiant l'approximation mesures/modèle. L'écart entres les mesures effectuées sur la machine et les valeurs calculées par simulation du modèle est ε_k (4.29). Le but final de l'estimation des paramètres étant bien sûr la minimisation de ce résidu ε_k.

Si nous choisissons comme critère J directement l'écart ε_k, nous nous apercevons que les erreurs positives compensent les erreurs négatives sur l'ensemble des K mesures et le critère ainsi choisi ne serait pas efficace. Pour remédier à ce problème, on avait le choix entre considérer la valeur absolue du résidu ε_k, considérer le carré de l'erreur, soit $(\varepsilon_k)^2$, ou choisir un critère J quadratique tel que définit dans (4.30).

La valeur optimale de $\underline{\theta}$ est obtenue, donc, par minimisation du critère quadratique suivant :

$$J = \sum_{k=1}^{K} \varepsilon_k^2 = \sum_{k=1}^{K} \left(y_k^* - \hat{f}_k \left(u, \underline{\hat{\theta}} \right) \right)^2 \qquad (4.30)$$

Comme $\hat{y}(t)$ n'est pas linéaire en $\hat{\theta}$, la minimisation de ce critère s'effectue par une méthode d'optimisation non linéaire [49]. La valeur optimale du vecteur paramètre notée $\underline{\theta}_{opt}$ est obtenue par un algorithme d'optimisation itératif.

4.4. Choix des signaux d'entrée-sortie

La figure 4.4 montre les variables d'entrées-sorties du système représentant la machine asynchrone triphasée.

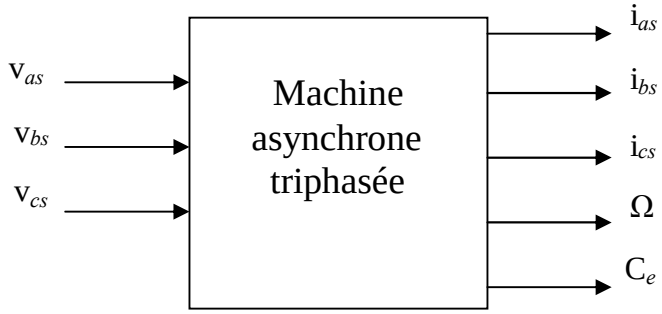


Figure 4.4 : Les variables entrées-sorties du modèle de la machine.

Plusieurs combinaisons d'entrées-sorties sont envisageables. Dans ce travail, nous n'utilisons que le courant démarrage et la tension simple correspondante pour déterminer simultanément les paramètres électriques et mécaniques de la machine.

4.5. Application d'API à l'identification paramétrique de la machine

Dans cette section, nous mettons en œuvre l'algorithme de colonie de fourmis **API** pour déterminer simultanément les paramètres électriques et mécaniques d'un moteur asynchrone triphasé à partir du courant de démarrage et de la tension simple correspondante. Nous choisissons la méthode du modèle de référence (erreur de sortie) comme technique d'identification.

Comme vu précédemment, en négligeant la saturation magnétique et l'effet de peau, et en ne considérant que le premier harmonique d'espace, le modèle de la machine asynchrone dans un repère lié au stator est le modèle (4.27) à quatre paramètres électrique (σ, T_s, L_s et T_r) et à trois paramètres mécaniques (J, f_r et C_s) suivant :

$$\begin{cases} \dfrac{d\mathrm{i}_{ds}}{dt} = -\dfrac{1}{\sigma.T_s}.\mathrm{i}_{ds} + \dfrac{(1-\sigma).p.\Omega}{\sigma}.\mathrm{i}_{qs} + \dfrac{1-\sigma}{\sigma.T_r}.\mathrm{i}'_{dr} + \dfrac{(1-\sigma).p.\Omega}{\sigma}.\mathrm{i}'_{qr} + \dfrac{\mathrm{v}_{ds}}{\sigma.L_s} \\ \dfrac{d\mathrm{i}_{qs}}{dt} = -\dfrac{(1-\sigma).p.\Omega}{\sigma}.\mathrm{i}_{ds} - \dfrac{1}{\sigma.T_s}.\mathrm{i}_{qs} - \dfrac{(1-\sigma).p.\Omega}{\sigma}.\mathrm{i}'_{dr} + \dfrac{1-\sigma}{\sigma.T_r}.\mathrm{i}'_{qr} + \dfrac{\mathrm{v}_{qs}}{\sigma.L_s} \\ \dfrac{d\mathrm{i}'_{dr}}{dt} = \dfrac{1}{\sigma.T_s}.\mathrm{i}_{ds} - \dfrac{p.\Omega.}{\sigma}.\mathrm{i}_{qs} - \dfrac{1}{\sigma.T_r}.\mathrm{i}'_{dr} - \dfrac{p.\Omega}{\sigma}.\mathrm{i}'_{qr} - \dfrac{\mathrm{v}_{ds}}{\sigma.L_s} \\ \dfrac{d\mathrm{i}'_{qr}}{dt} = \dfrac{p.\Omega}{\sigma}.\mathrm{i}_{ds} + \dfrac{1}{\sigma.T_s}.\mathrm{i}_{qs} + \dfrac{p.\Omega}{\sigma}.\mathrm{i}'_{dr} - \dfrac{1}{\sigma.T_r}.\mathrm{i}'_{qr} - \dfrac{\mathrm{v}_{qs}}{\sigma.L_s} \\ \dfrac{d\Omega}{dt} = \dfrac{p}{J}.(1-\sigma).L_s.(\mathrm{i}_{qs}.\mathrm{i}'_{dr} - \mathrm{i}_{ds}.\mathrm{i}'_{qr}) - \dfrac{f_r.\Omega - C_s}{J} \end{cases} \tag{4.31}$$

La machine est donc, décrite par un système non linéaire en Ω et elle est parfaitement caractérisée par le vecteur de paramètres Π tel que :

$$\Pi = \begin{bmatrix} \sigma & T_s & L_s & T_r & J & f_r & C_s \end{bmatrix}$$

Nous nous proposons d'utiliser la procédure **API** pour déterminer simultanément les sept paramètres du vecteur Π. Nous utiliserons d'abord des données simulées, puis des données expérimentales relevées sur un moteur.

4.5.1. Principe d'identification

Comme convenu (section 4.3.1), la méthode du modèle de référence, illustrée par la figure 4.5, est adoptée dans ce travail. On se propose d'identifier le vecteur de paramètres Π par minimisation de la fonction objective *E* représentant l'erreur quadratique entre les valeurs des courants connues, issues du système réel et les valeurs des courants calculées à partir du modèle adopté.

$$E = \sum_{i=1}^{K} (I_{kmi} - I_{kci})^2 \qquad (4.32)$$

K désigne la dimension du vecteur de courant mesuré et *k* l'indice de la variable d'état considérée. Le modèle et le système réel sont excités par la même tension U.

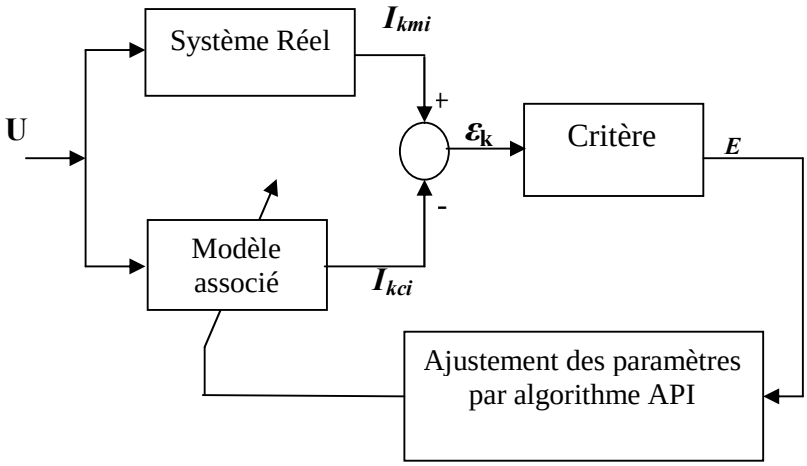


Figure 4.5 : *Principe d'identification*

La minimisation de la fonction *E* est effectuée par l'algorithme de fourmis **API**.

4.5.2. Adaptation de la technique **API** au problème

On présente dans cette section, l'adaptation, étape par étape, de l'algorithme **API** à notre application :

- D'un point de vue global, initialement, **API** place aléatoirement le nid à une position **N** de l'espace de recherche ***S***, ce qui, dans notre application, revient à générer aléatoirement un vecteur de paramètres initial Π_{ini}.
- Dans la seconde étape, l'exploration de l'espace au voisinage de **N** commence et cette exploration est déterminée par le comportement local des fourmis.

- A chaque pas de l'algorithme les ***n*** fourmis sont simulées en parallèle et chacune d'elles, quitte le nid pour formuler une nouvelle liste de ***p*** sites de chasse qu'elle mémorise. Un site de chasse est un point ***s*** de l'espace de recherche ***S*** construit par un opérateur $\boldsymbol{O}_{\mathbf{explo}}$ utilisant un rayon de recherche $\boldsymbol{A}_{sit}$ autour du nid **N**. Ceci revient, dans notre cas, à générer ***p*** vecteurs de paramètres (pour chaque fourmi) en partant du vecteur initial Π_{ini} . Les sept paramètres de chacun de ces ***p*** vecteurs seront calculés de la manière suivante :

$$\Pi_{ini\ j}^{'} = \Pi_{ini\ j} + A\boldsymbol{U}\left[-0.5,0.5\right]\left(B_j - b_j\right) \qquad \forall j \in \{1,....,7\} \qquad 4.32$$

  A est le rayon de recherche, U [-0.5, 0.5] est un nombre aléatoire uniformément tiré dans l'intervalle [-0.5, 0.5] et b_j et B_j délimitent le domaine d'existence de chaque paramètre du vecteur Π.

- Puis chaque fourmi choisi un site (vecteur de paramètres) au hasard parmi les ***p*** dont elle dispose et commence une exploration locale à son voisinage. Cette exploration locale consiste alors, à construire un point ***s'*** de l'espace de recherche ***S*** (un autre vecteur de paramètres Π') grâce à l'opérateur $\boldsymbol{O}_{\mathbf{explo}}$ avec un rayon de recherche $\boldsymbol{A}_{local}$. Dans notre application on utilisera la même formule (4.32) pour générer Π'. La fourrageuse capture une proie si l'exploration locale conduit à une meilleure valeur de la fonction objectif ***f*** (erreur quadratique), soit : $\boldsymbol{f(s') < f(s)}$ auquel cas le site ***s'*** sera mémorisé à la place de ***s*** et la prochaine exploration locale se fera dans le voisinage de ***s'***, donc du nouveau vecteur de paramètres.
- Si l'exploration locale est infructueuse, la fourmi choisira pour une exploration future au hasard, un autre vecteur de paramètres parmi les ***p*** qu'elle a en mémoire et quand un vecteur a été exploré plus de $\boldsymbol{P}_{local}$ fois sans succès, il est définitivement oublié et sera remplacé par un nouveau vecteur à la prochaine sortie (prochaine itération).
- Le nid est déplacé toutes les $\boldsymbol{P}_{\mathrm{N}}$ déplacements des ***n*** fourmis. Il est alors placé sur le meilleur point $\boldsymbol{s}^{+}$ trouvé depuis son dernier déplacement. A chaque déplacement du nid les fourmis reprennent leur exploration à partir de la nouvelle position du nid.

4.5.3. Réglage des paramètres et choix de la variante d'API

Pour la fixation des valeurs optimales des paramètres d'API, on procède par des tests « essais - résultats » en nous basant sur les aboutissements des travaux qui ont été fait sur

l'étude de l'influence de ces même paramètres en littérature (voir chapitre 3 section 3.5.1). En effet bien que ces travaux soient relatifs à des applications spécifiques, ils nous ont fournis quelques informations à priori qui nous a permit de réduire d'une manière sensible le coût de recherche des paramètres optimaux pour à notre application.

Les paramètres optimaux utilisés pour cette application d'**API** sont :

- Une population hétérogène constituée d'un nombre de fourmis : $\boldsymbol{n} = 6$.
- Les amplitudes de recherche $\boldsymbol{A_{sit}}$ et $\boldsymbol{A_{local}}$ varieront d'une fourmi à une autre mais seront fixées dans le temps, comme définie dans la relation 3.2 et illustrée dans la figure 3.6 du chapitre 3.
- En outre, ces amplitudes seront divisées par 10, pour toutes les fourmis dés que l'erreur quadratique descend sous un certain seuil.
- L'amplitude d'exploration locale $\boldsymbol{A_{local}(a_i)}$ est fixée pour toutes les fourmis à la valeur $0.1.\boldsymbol{A_{sit}(a_i)}$.
- Le nombre de sites mémorisés par une fourmi est $\boldsymbol{p} = 2$.
- La patience locale sur un site est $\mathbf{P}_{\boldsymbol{local}} = 3$.

4.5.4. Données simulées

Un programme représentant toutes les étapes de la l'algorithme **API** avec population de fourmis hétérogène, a été établi et exécuté pour déterminer simultanément les paramètres du vecteur Π d'une machine asynchrone triphasée en utilisant des données simulées. Ces données simulées sont obtenues par résolution numérique du système non linéaire (4.31) (décrivant le fonctionnement du moteur) par la méthode de Runge-Kutta en utilisant le vecteur de paramètres Π_c d'un moteur asynchrone triphasé connu, soit :

$$\Pi_c = [\ 0.09\ \ 0.054\ \ 0.159\ \ 0.123\ \ 0.038\ \ 0.002\ \ 0.5\ ]$$

Ces données sont illustrées par la figure 4.6 et représentent le courant de démarrage et la tension simple correspondante.

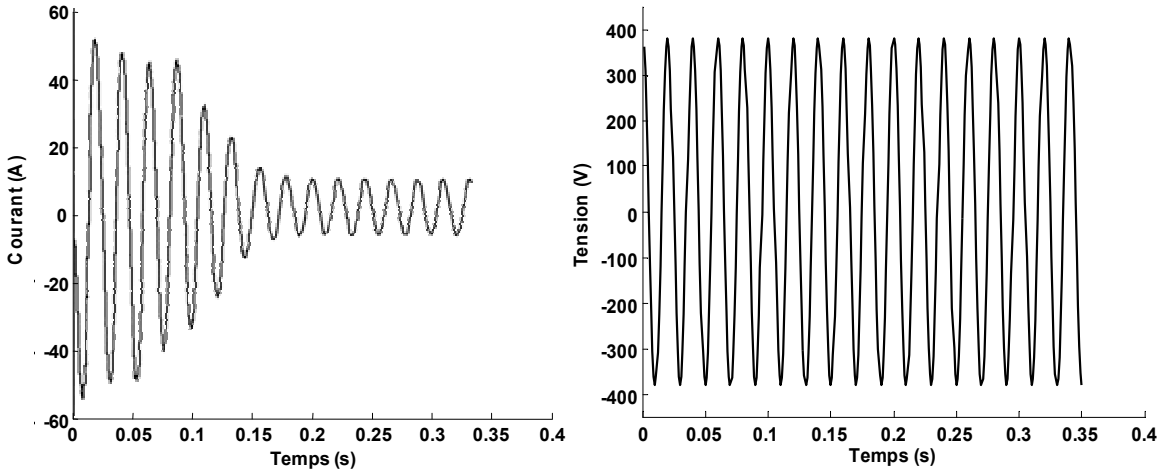


Figure 4.6 : Courant de démarrage et tension simple correspondante.

A l'aide de ces données simulées on se propose de faire une validation de l'algorithme **API** ainsi implémenté et étudier son efficacité vis-à-vis du problème.

D'autres part, on vérifiera que la connaissance du courant de démarrage et de la tension correspondante est suffisante pour l'estimation de tous les paramètres de la machine.

Les paramètres de la machine asynchrone triphasée ont été identifiés (calculées) par la procédure **API** ainsi définie en partant de plusieurs vecteurs initiaux différents Π_i générés aléatoirement sans aucune information à priori sur le système. Chaque vecteur initial Π_i correspond à la position initiale du nid dans l'espace de recherche ***S*** pour une exécution du programme.

Cependant, dans le but d'étudier l'efficacité de notre algorithme et pour pouvoir comparer nos résultats avec ceux obtenus à l'aide d'autres méthodes d'optimisation, pour la même application et la même machine (au sein de notre laboratoire SET) [41,50,51]. On choisi deux vecteurs initiaux Π_1 et Π_2 possédant chacun des composants identiques respectivement égales à **0.1** et **0.3** (comme cela a été fait dans les travaux énumérés).

Le tableau 4.1 donne les paramètres estimés par **API** en prenant Π_1 puis Π_2 comme vecteurs initiaux de paramètres.

On constate que les valeurs calculées par l'algorithme **API** sont très proches des valeurs des paramètres de la machine simulée, même si initialement, on démarre la recherche à partir de deux positions totalement différentes (vecteurs initiaux Π_1 et Π_2).

	σ	T_s (s)	L_s (H)	T_r (s)	J (Kg.m²)	f_r (N.m.s/rd)	C_s (N)
Paramèt. connus Π_c	0.09	0.054	0.159	0.123	0.038	0.002	0.5
Paramèt. Estim. 1	0.089987	0.054015	0.15902	0.12302	0.0380035	0.002126	0.488
Paramèt. Estim. 2	0.089986	0.054003	0.15904	0.12303	0.037986	0.002210	0.475

Tableau 4.1 : Comparaison entre les paramètres estimés par **API** et les paramètres connus (estimation à partir des 2 vecteurs initiaux Π_1 et Π_2).

Les figures 4.7 à 4.13 illustrent l'évolution des différents paramètres du vecteur Π en fonction du nombre d'itération lorsque les vecteurs initiaux sont Π_1 et Π_2. On observe la bonne concordance entre les paramètres estimés et les paramètres connus.

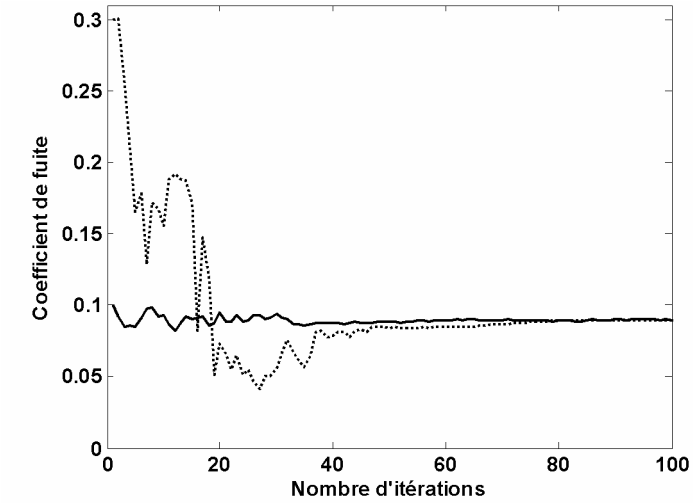


Figure 4.7 : Evolution du coefficient σ (*σ initial :* —— *0.1* ····· *0.3*)

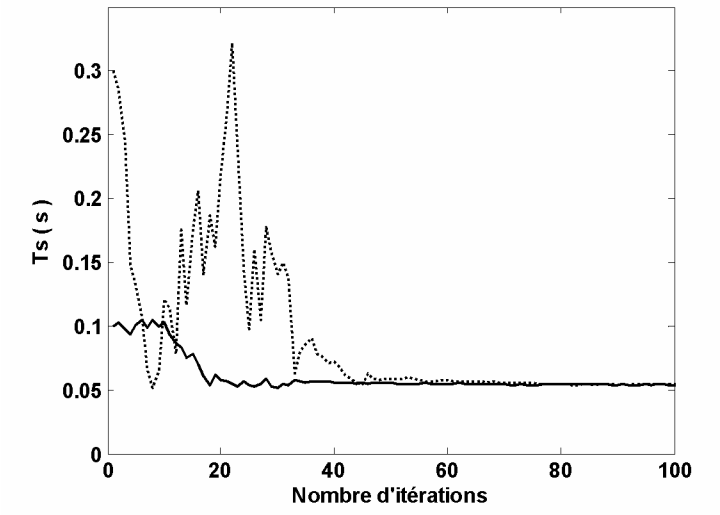


Figure 4.8 : Evolution de T_s (*T_s initial :* —— *0.1s* ····· *0.3s*)

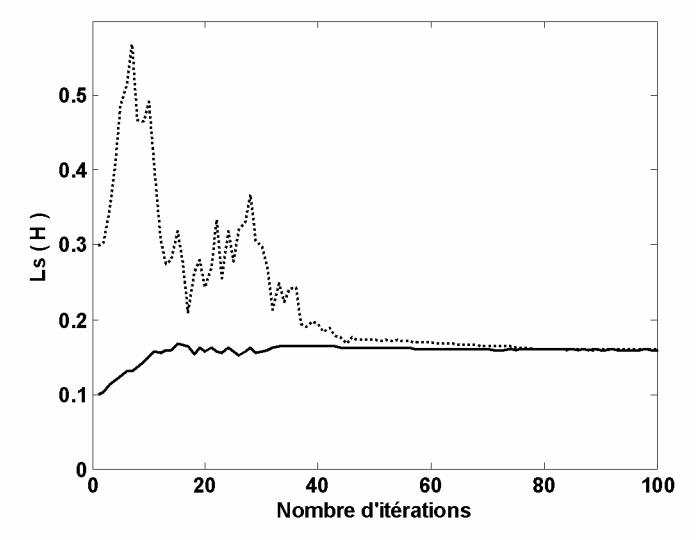


Figure 4.9 : Evolution de L_s (L_s *initial :* —— *0.1H* ····· *0.3H*)

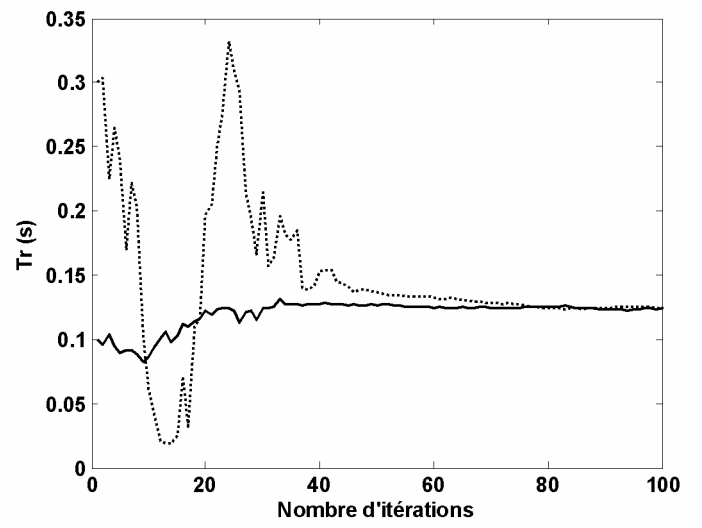


Figure 4.10 : Evolution de T_r (T_r *initial :* —— *0.1s* ····· *0.3s*)

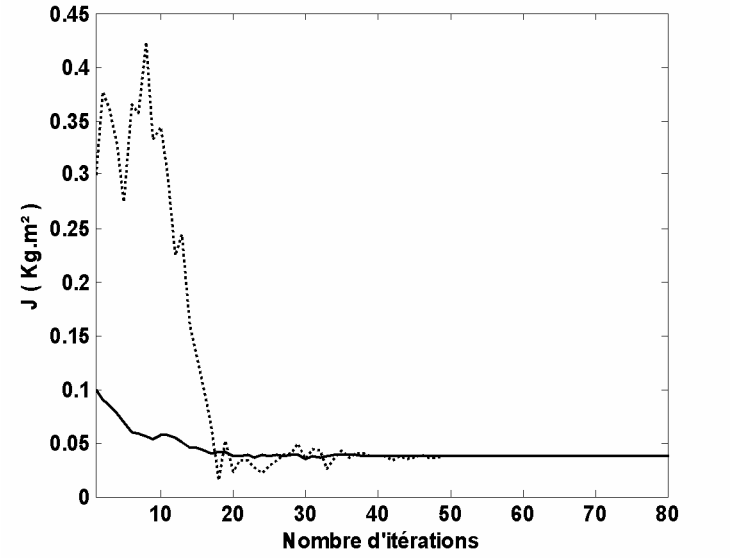


Figure 4.11 : Evolution de *J* (*J initial :* —— *0.1kg.m²* ····· *0.3kg.m²*)

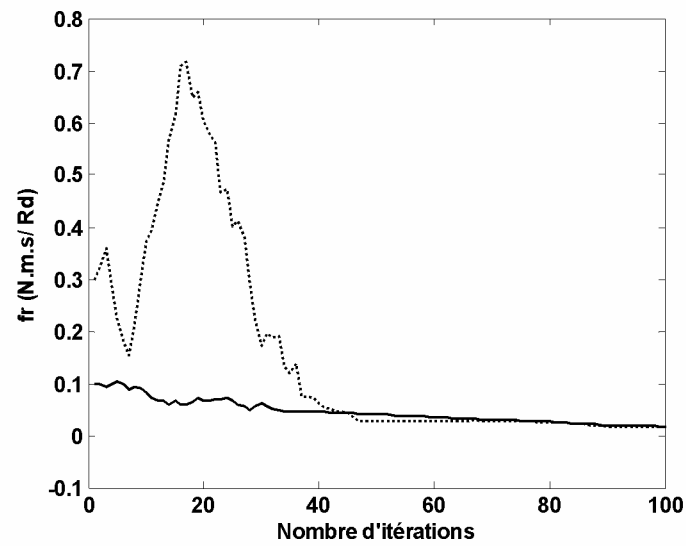


Figure 4.12 : Evolution f_r (f_r *initial :* —— *0.1m.s/rd* ····· *0.3m.s/rd*)

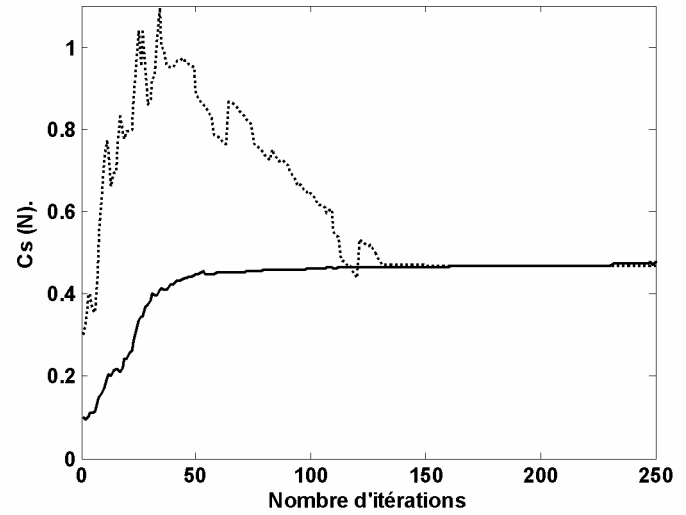


Figure 4.13 : Evolution de C_s (C_s *initial :* —— *0.1N* ····· *0.3N*)

Comme attendu, on constate que c'est le coefficient C_s qui converge le moins vite. Ce paramètre qui intervient principalement à faible vitesse, n'est en effet, pas suffisamment excité lors d'un simple essai unidirectionnel de démarrage.

On notera aussi, que l'évolution des paramètres n'est pas uniforme, ceci est sûrement dû au caractère aléatoire de la création de nouvelles solutions par API (nouvelles positions du nid).

4.5.5. Données expérimentales

Les essais sont effectués sur une machine asynchrone triphasée M, présentant les caractéristiques suivantes :

M : 2 pôles, 220/380 V, 3000 W.

Les courbes de la figure 4.14 représentent respectivement le courant de démarrage à vide et la tension simple correspondante mesurés pour le moteur. On observe que la source utilisée ne fournit pas une tension sinusoïdale pure ; cette dernière comporte un nombre important d'harmoniques.

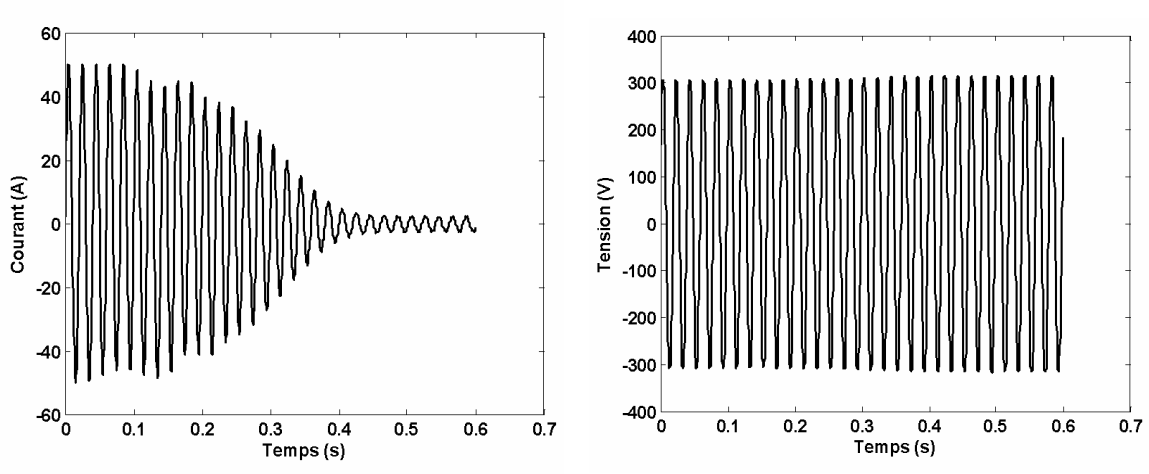


Figure 4.14 : Démarrage à vide : courant mesuré et tension correspondante

La procédure **API** conduit aux paramètres donnés par le tableau 4.2 :

σ	T_s **(s)**	L_s **(H)**	T_r **(s)**	J **(Kg.m²)**	f_r **(N.m.s/rd)**	C_s **(N)**
0.025283	**0.24144**	**0.54227**	**0.12789**	**0.091562**	**0.000135**	**0.23904**

Tableau 4.2 : Paramètres estimés par **API** à partir de l'essai de démarrage.

Afin de valider l'estimation paramétrique, on vérifie que le courant estimé est comparable au courant réel aussi bien pour le régime transitoire que pour le régime établi (figures 4.15 – 4.17).

En effet le vecteur de paramètre Π obtenu par minimisation de la fonction objectif sur tout le démarrage, est un vecteur de paramètres moyen, il convient pour simuler aussi bien le régime permanent que le régime transitoire. Ce vecteur tient compte des éventuelles variations de certains paramètres, comme la variation des inductances avec le phénomène de saturation magnétique.

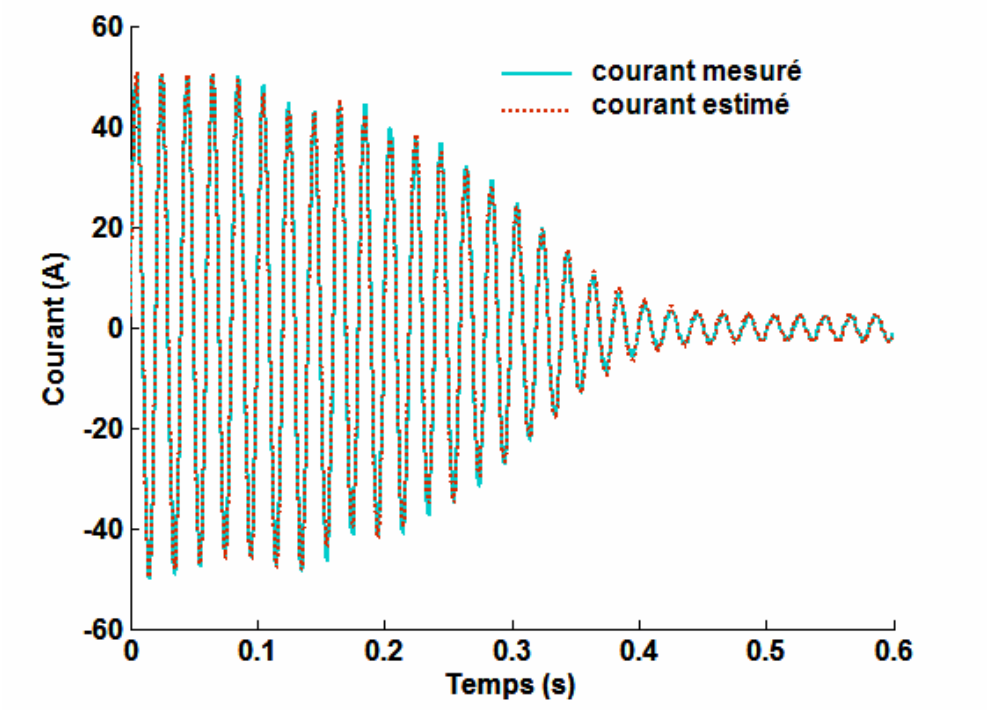


Figure 4.15 : Comparaison entre le courant mesuré et le courant calculé à l'aide des paramètres estimés.

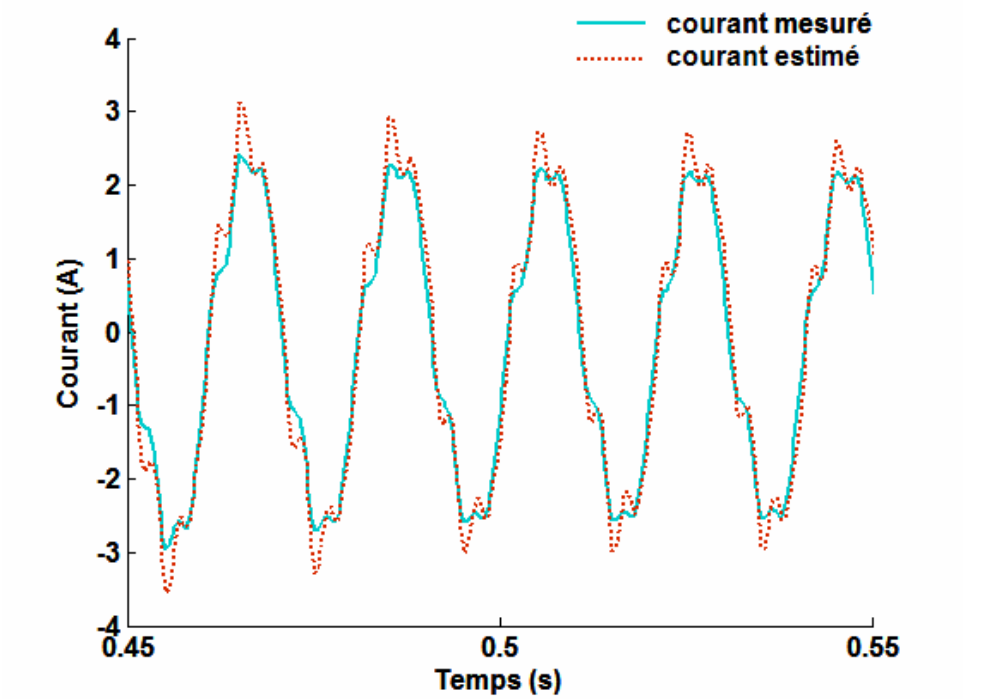


Figure 4.16 : Zoom sur la partie régime établi de la figure 4.15.

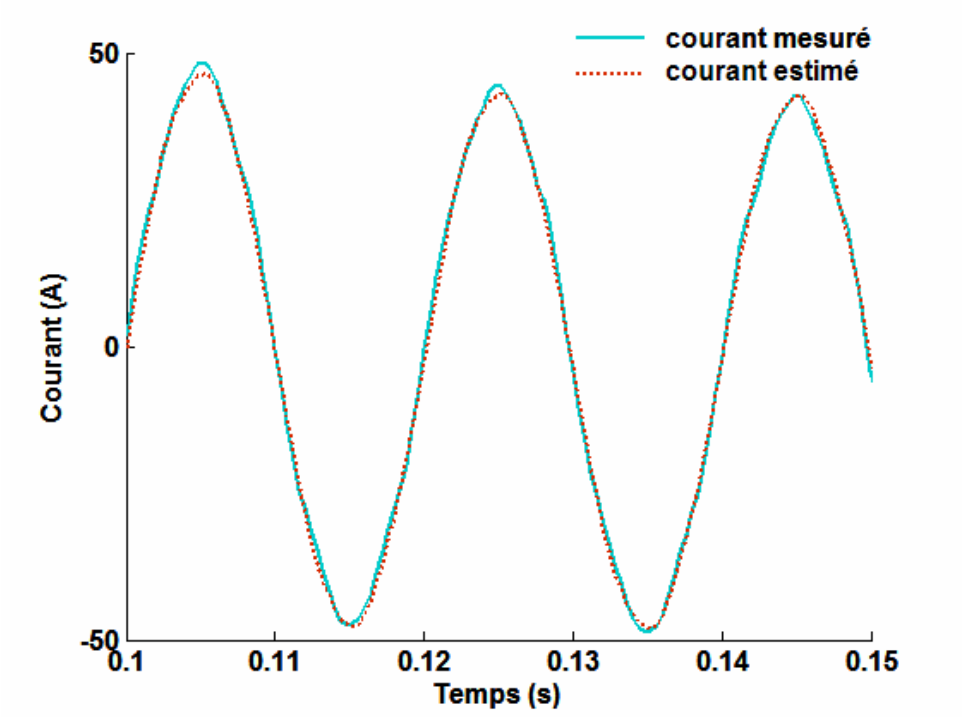


Figure 4.17 : Zoom sur la partie « régime transitoire » de la figure 4.15.

On constate que le courant estimé suit parfaitement le courant mesuré avec une erreur maximale de 7% (courant crête à crête) (figure 4.18).

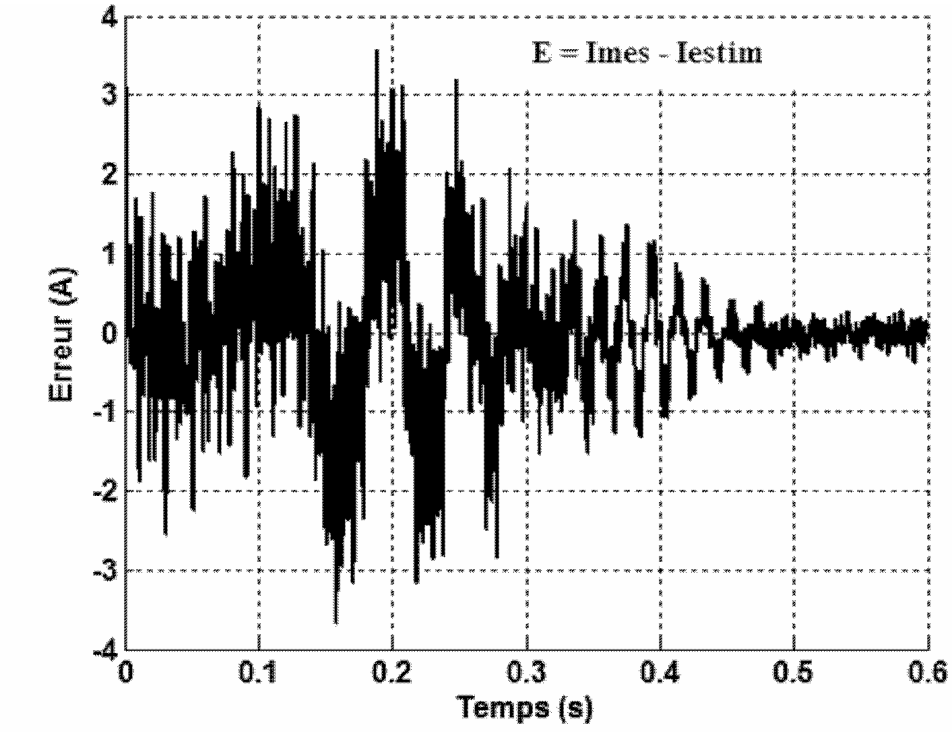


Figure 4.18 : L'erreur E entre le courant mesuré et le courant calculé avec les paramètres estimés.

L'évolution de l'erreur, comme observée dans la figure 4.18, peut être dû à plusieurs facteurs, on citera principalement :

- Le type de bruit affectant les mesures prises sur la machine,
- les erreurs de modélisation causées par la négligence de certains paramètres non linéaire tel que la saturation magnétique.

4.5.6. Comparaison avec d'autres méthodes

Il est important de comparer nos résultats avec ceux obtenus par l'utilisation d'autres techniques, sur la même application, la même machine et les mêmes conditions de travail.

Au sein du laboratoire de recherche S.E.T de notre département, plusieurs procédures de minimisation n'utilisant pas le calcul de dérivées, ont été utilisées pour déterminer simultanément les paramètres électriques et mécaniques de la machine asynchrone triphasée [41,42]. On citera :

1. la méthode de relaxation cyclique ,
2. la méthode de Hooke & Jeeves,
3. la méthode de Rosenbrock,
4. les algorithmes génétiques.

En pratique, on compare le rendement de deux techniques, en étudiant la qualité des solutions obtenues et en mesurant l'effort de calcul.

Dans un premier temps, on confronte les résultats obtenus par API avec ceux obtenus par la méthode de Rosenbrock, méthode qui a prouvé sa supériorité relativement aux deux premières techniques (relaxation cyclique et Hooke & Jeeves) [41].

Nos conclusions sont qu'après plusieurs exécutions des deux techniques (API, tel que défini précédemment et Rosenbrock), les qualités des solutions trouvées (vecteurs de paramètres) sont presque similaires, cependant API nécessite un effort de calcul moindre (nombre d'itérations ou temps de calcul), ce qui représente un avantage décisif.

Dans un second temps, on compare nos résultats avec ceux obtenus par un algorithme génétique [42]. Vu le caractère non déterministes des deux méthodes (deux exécutions successives ne donne pas la même solution), et vu que la complexité théorique d'une itération diffère d'une méthode à l'autre (une génération pour un AG et une solution construite par la colonie pour API), on procède de la manière suivantes :

- La mesure de l'effort de calcul est exprimée en seconde,

- On fixe le temps de calcul pour les deux méthodes puis on procède à un test statistique portant sur les moyennes des solutions obtenues.
- La répartition des solutions produites par les deux méthodes n'étant pas gaussienne, le nombre d'exécutions numériques est choisi très grand (plus d'une centaine d'exécution pour chaque méthode)

Nos constatations sont que la qualité de la valeur moyenne de la solution produite par API est très supérieure à celle produite par l'AG, ce qui nous fait dire que la méthode basée sur les algorithmes génétiques, nécessite un effort de calcul gigantesque en temps et en itérations.

4.5.7. Conclusion

Ce travail, qui se positionne comme une contribution à l'identification paramétrique de la machine asynchrone triphasée, a montré que l'algorithme de fourmis **API** constitue une alternative non négligeable dans le domaine de l'optimisation des processus complexes. Les résultats obtenus sont en effet très compétitifs avec ceux obtenus par d'autres méthodes d'identification plus classiques [41, 42, 43].

Un des atouts de cet algorithme est que sa description est relativement simple et indépendante de l'espace de recherche considéré, ce qui encourage son utilisation pour d'autres applications. En outre, cet algorithme est adapté aussi bien aux problèmes combinatoires qu'aux problèmes où la fonction objective est à variables continues.

Le succès de la métaheuristique ne doit pas masquer les difficultés du réglage des divers paramètres de la procédure. Ce réglage « optimal » tel que préconisé en théorie est inapplicable en pratique à cause du coût prohibitif en calcul.

Les principales évolutions d'**API** que l'on peut espérer sont ses adaptations aux problèmes d'optimisation avec contraintes, à l'optimisation multicritère et l'optimisation dynamiques.

Les perspectives sont nombreuses et concernent pour la plupart des études expérimentales sur l'influence des paramètres d'**API** sur les résultats obtenus, ainsi que des hybridations d'**API** avec d'autres méthodes heuristiques d'optimisation.

CONCLUSION

La métaheuristique de colonies de fourmis inclut une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthode classique efficace. Ces algorithmes sont généralement utilisés comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans la procédure employée et la majorité d'entre eux s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromone) et construisent ainsi une solution à un problème, en s'appuyant sur leur expérience collective.

Cette métaheuristique, d'origine discrète, a été élaborée par les informaticiens pour la résolution des problèmes d'optimisation combinatoires. Cependant, il existe, en ingénierie, une large gamme de problèmes où la fonction objectif est à variables continues et pour lesquelles, la métaheuristique peut être d'un grand secours (fonction non dérivable, multiples minimums locaux, non convexe, nombre de variables important,..). Dés lors, l'adaptation de ces algorithmes, aux cas continus devient une priorité et plusieurs tentatives "d'adaptation" sont apparues.

En plus des problèmes classiques liés au perfectionnement de l'adaptation d'une métaheuristique, les algorithmes de colonies de fourmis posent un certains nombre de problèmes reliés à la construction des solutions, ce qui nous fait dire que les algorithmes produits n'ont pas atteint leur pleine maturité.

Dans ce travail thèse, on s'est intéressé à l'algorithme de colonie de fourmis API, déduit de la modélisation du comportement de fourragement de l'espèce *Pachycondyla apicalis* et son application au problème général de l'optimisation (discret et continu). L'intérêt porté à cette espèce de fourmis dans le domaine de l'optimisation provient du fait que ceux ci utilisent des principes relativement simples à la fois d'un point de vue global et local pour rechercher leurs

proies. A partir de leur nid, ces fourmis couvrent globalement une surface donnée en la partitionnant en plusieurs sites de chasse individuels. Chaque fourmi effectue une exploration aléatoire locale de ses sites de chasse et le site choisi est sensible au succès précédemment rencontré sur les autres sites.

On a vu que cet algorithme, qui ne fait pas usage de la communication indirecte par pistes de phéromone, s'est avéré, au moins pour partie, stochastique et offre une robustesse en terme d'hypothèses sur la fonction objective, une simplicité d'implémentation et n'est pas limité par la dimensionnalité de l'espace de recherche. En outre, API, qui est aussi une méthode à démarrage multiple, améliore sont efficacité par un partitionnement adaptatif de l'espace de recherche.

Plusieurs directions ont été explorées et exploitées pour l'enrichissement de l'algorithme API et plusieurs variantes d'API ont été élaborées. Pour cela, on s'est, soit s'inspiré des caractéristiques naturelles des fourmis, soit, on a ajouté à API des étapes de recherches locales liées au problème traité.

L'algorithme, ainsi défini et ses variantes ont été ensuite, appliquées à deux types de problèmes :

- un problème combinatoire difficile : le problème du voyageur de commerce,
- un problème continu : l'identification paramétrique d'une machine asynchrone triphasée.

Dans la première application, on a testé et observé le comportement d'API vis-à-vis d'un algorithme de fourmis classique, type ACO, pour un problème d'optimisation combinatoire NP-difficile très connu dans le domaine.

La deuxième application nous a permit d'étudier la réaction d'API, à un problème à variables continues, relativement complexe. En effet, la détermination des paramètres de la machine asynchrone à l'aide de la technique du modèle de référence est obtenue par la minimisation de la fonction objectif représentée par l'erreur de sortie quadratique entre les valeurs des courants mesurés et ceux calculés. Sachant que la fonction objectif n'est pas explicite, l'avantage décisif de l'algorithme API par rapport aux autres approches utilisées pour la même application, est qu'il soit direct et ne recourt pas au calcul, souvent problématique, des gradients de la fonction objectif.

En outre, la plupart des méthodes utilisées jusqu'alors, estiment uniquement les paramètres électriques du modèle, les paramètres mécaniques étant déterminés séparément par des procédés auxiliaires qui demandent des prouesses techniques qui ne sont pas toujours évidentes. Dans cette thèse, les paramètres électriques et mécaniques sont estimés simultanément par API et les résultats obtenus sont très compétitifs, avec ceux obtenus par méthodes exactes classiques.

Les perspectives de ce travail sont relativement nombreuses, tant du point de vue théorique que pratique. On pourrait citer, par exemple :

- l'approfondissement du principe de méta-réglage, pour une formalisation de la détermination systématique des paramètres de l'algorithme API.
- Hybridations d'API avec d'autres méthodes heuristiques d'optimisation, dans le but, d'améliorer la convergence de l'algorithme. On pourra, par exemple, changer la règle de décision d'une fourmi lors de la capture d'une proie en utilisant une heuristique secondaire.
- L'utilisation de « modèle multi populations » pour pallier le problème de l'initialisation du nid. En effet les performances de l'algorithme sont fortement dépendantes de cette initialisation. L'idée permettra à l'algorithme de démarrer en différents points de l'espace de recherche S qui sera alors, bien quadrillé. La stratégie consistera à utiliser plusieurs colonies de fourmis, chacune avec un nid positionné initialement à une position différente dans S.
- On peur envisager aussi des adaptations d'API aux problèmes d'optimisation avec contraintes, à l'optimisation multicritère et l'optimisation dynamiques.

APPENDICE A

LISTE DES SYMBOLES ET DES ABREVIATIONS.

A_{local} : Amplitude d'exploration locale

A_{sit} : Amplitude maximale de création d'un site

C_s : Coefficient de frottements secs

d_{ij} : distance entre les nœuds i et j

E : Erreur quadratique entre les valeurs des courants connues et calculées

f_r : Coefficient de frottements

i_r : Courant rotorique

i_s : Courant statorique

J : Masse d'inertie du rotor

L^k : Longueur du chemin parcouru par la fourmi k

L_r : Self inductance rotorique

L_s : Self inductance statorique

n : nombre de fourmis

O_{rand} : Opérateur générant aléatoirement une solution s de l'espace S

O_{explo} : Opérateur générant aléatoirement une solution s' au voisinage de s

P_{ij}^k : Probabilité de transition du nœud i vers le nœud j par la fourmi k

p : nombre de sites mémorisés par une fourmi

P_{local} : Patience locale d'une fourmi

P_N : Patience du nid

P_{recrut} : Probabilité de recrutement d'une fourmi

R_{xy} : Type de recrutement d'une fourmi

T_r : Constante de temps rotorique

T_s : Constante de temps statorique

v_{ij} : visibilité entre les nœuds i et j

V_s : Tension statorique

V_r : Tension rotorique

Φ_s : Flux statorique

Φ_r : Flux rotorique

$\tau_{ij}(t)$: Intensité de la trace de phéromone sur le chemin (i,j) à l'instant t

$\Delta \tau_{ij}^{k}$: renforcement du chemin (i,j) en phéromone par la fourmi k

ρ : Coefficient d'évaporation des traces de phéromone

σ : Coefficient de fuite

Π_c : vecteur de paramètres de la machine asynchrone triphasé connue

Π_{ini} : vecteur de paramètres initiale

REFERENCES

1. Colorni, A., "Distributed optimization by ant colonies", Proc. 1st European Conf. on Artificial Life, Paris, Elsevier, (1991), 134-139.
2. Colorni, A., Dorigo, M. & Maniezzo, V., "An investigation of some properties of an ant algorithm." In (Männer and Manderick) ,(1992), 509–520.
3. Colorni, A., Dorigo, M. & Maniezzo, V., "New Results of an Ant System Approach Applied to the Asymmetric TSP." I.H.Osman and Kelly,J., editors, Metaheuristics International Conference, Hilton Breckenridge, Colorado. Kluwer Academic Publishers (1995), 356–360.
4. Colorni, A., Dorigo, M., Maniezzo, V. & Trubian, M., "Ant System for Job-shop Scheduling." JORBEL - Belgian Journal of Operations Research,Statistics and Computer Science, 34(1) ,(1994), 39–53.
5. Dorigo, M., Maniezzo, V. & Colorni, A.," The Ant System: optimization by a Colony of Cooperating Agents." IEEE Trans. Syst. Man Cybern, tome B, No 26,(1996), 29-41.
6. Dorigo, M., Gambardella, L.M., "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem", IEEE Trans. on Evolutionary Computation, Vol.1, No.1, (April 1997), 53-66.
7. Dorigo, M., Stützle, T., "Handbook of Metaheuristics", tome 57 de International series in operations research and management science, chapitre "The ant Colony Optimization Metaheuristics : Algorithms, Applications and Advances." Kluwer Academic Publishers, Boston Hardbound, (2003).
8. Dorigo, M., Di Caro, G.," The Ant Colony Optimization meta-heuristic", In: Corne, D., Dorigo, M., Glover, F., (Eds.), News Ideas in Optimization, McGraw-Hill, London, (1999), 11-32.
9. Gambardella, L.M., Dorigo, M., "Solving symmetric and asymmetric TSPs by ant colonies", Proc. IEEE Conf. on Evolut. Computa. ICEC, New Jersy, (1996), 622-627.
10. Reinelt, G. (1995)., "TSPLIB ."Universität Heidelberg, Institut fur Angewandte Mathematik, http ://www.iwr.uni-eidelberg.de/iwr/comopt/TSPLIB95/TSPLIB.html.
11. Maniezzo, V., Carbonaro, A., "An ANTS heuristic for the frequency assignment problem." *Future Generation Computer Systems*, 16(8), (2000), 927–935.
12. Maniezzo, V., Colorni, A., and Dorigo, M., "The Ant System Applied to the Quadratic Assignment Problem." Technical Report 94-28, IRIDIA, Université Libre de Bruxelles, Belgium, (1994).
13. Gambardella, L.M., Taillard, E.D., & Dorigo, M.,"Ant colonies for the quadratic assignment problem", Journal of the Operational Research Society, Vol. 50, No. 2, (Feb. 1999), 167-176.
14. Costa, D., Hertz, A.," Ants can color graphs", Journal of the Operational Research Society, Vol. 48, No. 3, (Mar. 1997), 295-305.

15. Bilchev, G., Parmee, I.C, “The ant colony metaphor for searching continuous design spaces” In Fogarty, T.C. (Ed), Proc. of the AISB Workshop on Evolut. Computa., Vol. 993 of LNCS, Springer-Verlag, Berlin, Germany, (1995), 25-39.

16. Mathur, M., Karale, S.B., Priye, S., Jyaraman, V.K. & Kulkarni, B.D., “Ant colony approach to continuous function optimization” Industrial and Engineering Chemistry Research, Vol.39, (2000), 3814-3822.

17. Ling, C., Jie, S., Ling, Q. & Hongjian, C.,” A method for solving Optimization Problems in Continuous Space Using Ant Colony Algorithm.” Proc. of the Third Internat. Workshop on Ant Algorithms (ANTS’2002), Lecture Notes in Computer Science, 2002, 288-289.

18. Socha, K.,” ACO for Continuous and Mixed-Variable Optimization.” In Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F., and Stützle, T., editors, Ant Colony Optimization and Swarm Intelligence, volume 3172 of Lecture Notes in Computer Science, Springer, (2004), 25-36.

19. Monmarché, N., “ Algorithmes de fourmis artificielles : application à la classification et à l’optimisation” Thèse de doctorat, Laboratoire d’informatique, Université de Tours, France, (Décembre 2000).

20. Monmarché, N., Venturini, G., Slimane, M., “On how Pachycondyla apicalis ants suggest a new search algorithm” Future Generation Computer Systems, Vol.16, No.8, (2000), 937-946.

21. Monmarché N., Desbarats L., Slimane M., Venturini G.,“ Etude d’un nouvel algorithme d’optimisation et d’apprentissage inspiré d’une colonie de fourmis *Pachycondyla apicalis*”, In colloque Intelligence Artificielle et Complexité, Paris, (1998), 114 – 119.

22. Venturini, G., Slimane, M., Monmarché, N. & Fresneau, D., “ Modélisation des fourmis *Pachycondyla apicalis* appliquée à l’optimisation numérique.” Rapport interne 194, Laboratoire d’Informatique de l’Université de Tours, E3i Tours, (1997), 15 pages.

23. Fresneau, D., “ Biologie et comportement social d’une fourmi ponérine néotropicale (Pachycondyla apicalis)” Thèse d’état, Université Paris XIII, France, (1994).

24. Berthiau, G., Siarry, P., ”Etat de l’art des méthodes d’optimisation globale.” RAIRO Operations Research, tome 35, No 3, (2001), 329-365.

25. Berro, A.,”Optimisation multiobjectif et stratégie d'évolution en environnement dynamique.” Thèse PhD, Université Paul Sabatier, Toulouse, (2001).

26. Talbi, E., “A taxonomy of hybrid metaheuristics.” Journal of heuristics, tome 8, (2002), 541 – 564.

27. Dreo, J., Siarry, P., “ Algorithmes à estimation de distribution et colonies de fourmis.” 11éme journée évolutionnaire (JET11).

28. Deneubourg, J. L., Goss, S., Franks, N., Detrain, C., Chretien, L.,” The dynamics of collective sorting: robot-like ant and ant-like robots.” In Meyer and Wilson, (1990), 356–365.

29. Stützle, T., Hoos, H., “Max-Min ant system”, Future Generation Computer Systems, Vol. 16, No. 8, (Jun 2000), 889-914.

30. Stützle, T., “Parallelisation Strategies for Ant Colony Optimization.” In (Eiben et al., 1998a).

31. Taillard, E., “FANT : Fast Ant System.” Technical Report 46-98, IDSIA, Lugano, (1998).
32. Gutjahr, W. J., “A graph-based Ant System and its convergence.” Future Generation Computer Systems, tome 16, No 8, (2000), 873-888.
33. Gutjahr, W. J., “ACO algorithms with guaranted convergence to the optimal solution.” Information Processing Letters, tome 82, No 3, 2002, 987-999.
34. Lumer E.D., Faieta B., “Diversity and Adaptation in Populations of Clustering ants” Proceedings of the Third International Conference on Simulation of Adaptative Behavior, (1994), 501-508.
35. Dreo, J., Siarry, P.,”Un algorithme de colonies de fourmis en variables continues hybridé avec un algorithme de recherche locale”, 5eme Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2003).
36. Dréo, J., Siarry, P., “Continuous interacting ant colony algorithm based on dense hierarchy” Future Generation Computer Systems, Vol. 20, No. 5, (2004), 841-856.
37. Dreo, J., Siarry, P.,” Un nouvel algorithme de colonie de fourmis exploitant la communication directe entre individus, application à l'optimisation en variables continues.” ,4eme Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2002).
38. Wilson, E., Hölldobler, B., “Dense Heterarchy and mass communication as the basis of organization in ant colonies.” Trend in Ecology and Evolution, tome 3, 1988, 65-68.
39. Jaisson, P., “ La fourmi et le sociobiologiste.” Odile Jacob, Paris, (1993).
40. Meslet, O., ”Extension de l’algorithme API : Multi population et Recrutement pour l’optimisation numérique.” Rapport de projet de fin d’études, Ecole d’Ingénieurs en Informatique pour l’Industrie (E3i), Université de Tours, France, (1998).
41. Bounekhla, M., Zaim M.E. & Rezzoug, A. “ Calcul du gradient à l’aide des fonctions de sensibilité. Application à l’identification de la machine asynchrone” Revue Internationale du Génie Electrique, Vol. 6, No.5, 6/ (2003), 525-545.
42. Bounekhla, M., Guessoum, A., Zaim, M.E. & Rezzoug, A., “ Induction machine identification by genetic algorithm” Proc. 36 th Universities Power Engineering Conference UPEC, Swansea, UK, (2001), 196-199.
43. Bounekhla, M., ”Contribution à l’identification paramétrique de la machine asynchrone”, Thèse de doctorat d’état és-Sciences, ENPA , Alger, (Mai 2004).
44. Bellaa-Mrabet, N., Jelassi, K., “Comparaison de méthodes d’identification des paramètres d’une machine asynchrone.” Eur. Phys. J. AP3, (1998), 71-80.
45. Steven, R.S., Steven B.L., “Identification of induction motor parameters from transient stator current measurements” IEEE Trans. Indust. Electron., Vol. 46, No.1, (Feb. 1999), 139-149.
46. Clerc, G., Chouiter, D. R., M’Rabet Bellaaj N. &. Retif J. M., “Comparative study of identification methods for induction machines”. EPE’97, Trondheim, Norway, 8-10 (Sept 1997), 1524-1528.
47. Mensler, M.,”Analyse et étude comparative de méthodes d'identification des systèmes à représentation continue. Développement d'une boîte à outil logicielle.” Thèse de doctorat, Université de Nancy I, France, (1999).

48. Ljung, L.," System identification: Theory for the user." Prentice Hall, New Jersey, USA, (1987).

49. Trigeassou, J. C. & Poinot, T.," Identification des systèmes.", Chapitre : Identification des systèmes à représentation continue - Application à l'estimation de paramètres physiques, 177-211, Hermès, Paris, (2001).

50. Bachir, S., "Contribution au diagnostique de la machine asynchrone par estimation paramétrique" Thèse de doctorat, Université de Poitiers, France, (2002).

51. Caron, J. P., Hautier, J. P.," Modélisation et commande de la machine asynchrone." Technip, (1995).

52. Monmarché, N., Venturini, G., Slimane, M., " Chaînes Markov Cachées par une population de fourmis *Pachycondyla apicalis*." Rapport interne 195, Laboratoire d'Informatique de l'Université de Tours, E3i Tours, 1999, 24 pages.

53. Laurin, S.," Extension de l'algorithme API au Problème du Voyageur de Commerce." Rapport de projet de fin d'études, Ecole d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France, (1998).

54. Dreo, J., Petrowski, A., Siarry, P. & Taillard, E. D.,"Metaheuristiques pour l'optimisation difficile." Eyrolles, 2003.

55. Dromel, G., "Comparaison de systèmes d'optimisation à base de population." Rapport de projet de fin d'études, Ecole d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

56. Culioli, J.,"Introduction à l'optimisation." Ellipse, seconde édition, (1994).

57. Corne, D., Dorigo, M. & Glover, F.," New Ideas in Optimisation." McGraw-Hill, London, UK, (1999).

58. Pelikan, M., Goldberg, D., and Lobo, F.,"A Survey of Optimization by Building and Using Probabilistic Models." IlliGAL Report 99018, Illinois Genetic Algorithms Laboratory, University of Illinois, (1999).

59. Tang Z. B., "Adaptative partitioned random search to global optimization", IEEE Trans. Autom. Contr., Vol.39, (1994), 2235 – 2244.

60. Boudjahen, D., Mansouri, N., "A new algorithm for the global optimum neighbourhood localization in global optimization" Proc. ALIO / EURO Conf. on combinatorial optimization, Paris, France, (2005), 36 - 45.

61. Demirhan, M., Ozdamar L., "A note on a partitioning algorithm for global optimization with reference to Tang's statistical promise measure" IEEE Trans. Autom. Contr., Vol. 45, No.3, (2000), 510-515.

62. Hadj Sadok, M., Bounekhla, M. & Guessoum, A., "Contribution à l'identification paramétrique de la machine asynchrone par l'algorithme de fourmis API", 5éme Conférence sur le Génie Electrique (CGE'05), Alger, (Avril 2007).

63. Hadj Sadok, M., Salhi, H., Guessoum, A. & Bounekhla, M., "Induction machine parametric identification by the ants *Pachycondyla apicalis* algorithm." The Mediterranean Journal of Measurement and Control, (Avril 2007), 49- 58.

64. Moreau, S., Trigeassou, J. C., Loron, L., "Estimation paramétrique de la machine asynchrone. Etude comparative de deux algorithmes." 3éme conférence internationale sur l'automatisation industrielle, Montréal Canada, (Juin 1999).

65. Holland, J., "Adaptation in natural and artificial systems." University of Michigan Press, Ann Arbor, (1975).

66. Goldberg, D., "Genetic Algorithms in Search, Optimization and Machine Learning." Addison-Wesley, (1989).