Implémentation d'une méta-heuristique embarquée pour résoudre le problème d'ordonnancement dans un atelier flexible de production

Maroua Nouiri

République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
*****
Université de Carthage
École Polytechnique de Tunisie

THÈSE

Présentée à

L'École Polytechnique de Tunisie

En vue de l'obtention du

DOCTORAT

en Electronique et Technologie de l'Information et de la Communication

Par

Maroua NOUIRI

Implémentation d'une méta-heuristique embarquée pour résoudre le problème d'ordonnancement dans un atelier flexible de production

---

Soutenue le 03 Juillet 2017, devant le jury composé de :

Mr. Riadh ROBBANA (Professeur - INSAT)	**Président**
Mr. Mohamed ABID (Professeur - ENIS)	**Rapporteur**
Mr. Maher BEN JEMAA (Professeur - ENIS)	**Rapporteur**
Mr. Mohamed KHALGUI (Maître de conférences - INSAT)	**Examinateur**
Mr. Abderrazak JEMAI (Maître de conférences - INSAT)	**Directeur de Thèse**

Dédicaces

♠ Je dédie cette thèse ♠

À mes chers parents Hédi et Zohra,

En ce jour merveilleux, je voudrai vous témoigner l'amour infini et l'affection que je vous porte, veuillez percevoir à travers ce travail ma reconnaissance éternelle pour votre inépuisable tendresse et les efforts consentis pour mon instruction et mon bien-être.

À mes frères et mes soeurs

Vous m'avez comblé par votre solidarité et votre affection, cette thèse est la votre.

Mon amour est indéfectible.

À mes beaux frères

Ma gratitude pour votre aide, votre affection et vos encouragements.

À mes belles soeurs

Votre soutien m'a été d'un grand réconfort.

À mes oncles et mes tantes

Mes remerciements pour vos encouragements continus

À mes nièces, À mes neveux,

À tous mes proches,

À mes amis,

♠ Que Dieu le tout puissant vous bénisse tous. ♠

Remerciements

" *Soyons reconnaissants aux personnes qui nous donnent du bonheur ; elles sont les charmants jardiniers par qui nos âmes sont fleuries* ".

Marcel Proust

En premier lieu, je tiens à remercier mon Directeur de thèse, monsieur **Abderrazak JEMAI**, qui a cru en mes capacités et a accepté de diriger ma thèse. Je voudrai le remercier d'avoir enrichi mes connaissances. Son soutien et ses encouragements ainsi que sa bienveillance fraternelle m'ont comblé. Merci pour le temps consenti et la patience tout au long de ces années. Aucun mot ne saurait exprimer la gratitude, le respect et la reconnaissance que je vous dois.

Je tiens également à remercier **Mr. Ahmed Chiheb AMMARI** pour son aimable disponibilité et la promptitude de ses courriers. Ses remarques et ses précieuses suggestions m'ont beaucoup aidé à la finalisation de ce travail. Toute mon estime et mes profonds respects.

Ce travail a été réalisé en collaboration avec le laboratoire LAMIH de l'université de Valenciennes et du Hainaut-Cambrésis. Je tiens tout particulièrement à exprimer toute ma gratitude et mes vifs remerciements à **Mr. Abdelghani BEKRAR**, Maître de conférences à l'ISTV, et à nos professeurs **Mr. Smail NIAR** et **Mr. Damien TRENTESAUX**. Je voudrai les remercier d'avoir enrichi mes connaissances et d'avoir guidé mes stages au laboratoire LAMIH. L'ambiance de travail agréable au sein de leur équipe, les qualités pédagogiques et humaines ainsi que l'intérêt à l'égard de ma recherche m'ont permis au fil des ans de progresser au perfectionnement de ce travail. Toute ma gratitude et mes profonds respects.

Mes remerceiements s'adressent aussi à notre honorable professeur à l'INSAT, **Mr. Riadh ROBBANA** pour avoir accepté de présider le jury de cette thèse.

Je remercie nos honorables professeurs à l'ENIS **Mr. Mohamed ABID** et **Mr.Maher BEN JEMAA** d'avoir accepté la lourde charge de rapporteurs.

Je tiens à remercier également **Mr. Mohamed KHALGUI**, Maître de conférences à l'INSAT, d'avoir accepté de faire partie du jury de cette thèse et d'en être l'examinateur.

Table des matières

Liste des algorithmes

Table des figures

Liste des tableaux

Liste des abréviations

2s-PSO	two stage **P**article **S**warm **O**ptimization
ACO	**A**nt **C**olony **O**ptimization
AOR	**A**ffected **O**peration **R**escheduling
FJSSP	**F**lexible **J**ob **S**hop **S**cheduling **P**roblem
FMS	**F**lexible **M**anufacturing **S**ystem
FSSP	**F**low **S**hop **S**cheduling **P**roblem
GA	**G**enetic **A**lgorithm
HFS	**H**ybrid **F**low **S**hop
JSSP	**J**ob **S**hop **S**cheduling **P**roblem
mAOR	**m**odified **A**ffected **O**peration **R**escheduling
MAPSO1	**M**ulti **A**gent **P**article **S**warm **O**ptimization 1
MAPSO2	**M**ulti **A**gent **P**article **S**warm **O**ptimization 2
MAPSO2+	**M**ulti **A**gent **P**article **S**warm **O**ptimization 2+
MAPSO2++	**M**ulti **A**gent **P**article **S**warm **O**ptimization 2++
MAS	**M**ulti **A**gent **S**ystem
NN	**N**eural **N**etworks
PSO	**P**article **S**warm **O**ptimization
RSR	**R**ight **S**hift **R**escheduling
SA	**S**imulated **A**nnealing

Résumé

Les problèmes d'ordonnancement dans le secteur industriel sont en tête des problèmes d'optimisation les plus étudiés. Cependant, les approches de résolution ne sont pas adaptées à un environnement dynamique et de plus en plus incertain. Les travaux présentés dans cette thèse se situent dans le cadre de résolution du problème d'ordonnancement d'ateliers de type job shop flexible (FJSSP) sans et avec incertitudes. Nous avons proposé un algorithme d'optimisation par essaims particulaires (PSO) ainsi que deux architectures multi agent à base de PSO nommées MAPSO1 et MAPSO2 pour résoudre le problème FJSSP sans incertitudes. Après avoir testé nos algorithmes sur différents benchmarks , les résultats obtenues ont montré leurs efficacités. Nous avons proposé d'utiliser des systèmes embarqués pour le déploiement d'un processus d'optimisation intelligent et coopératif comme le PSO. Ainsi l'architecture MAPSO2 a été testée sur un système physiquement distribué comportant deux systèmes embarqués. Plusieurs améliorations ont été proposées donnant naissance à MAPSO2 + et MAPSO2++. Nous avons proposé aussi une approche prédictive basée sur PSO nommée 2s-PSO pour résoudre le problème FJSSP avec la prise en compte des pannes de machines. Les expérimentations ont montré que l'algorithme 2s-PSO augmente la robustesse et la stabilité de la solution. Ainsi, et comme dernière contribution, nous avons proposé une heuristique de ré-ordonnancement pour réparer l'ordonnancement en cas d'inefficacité sous incertitudes.

Mots Clés : FJSSP, PSO, Système Multi Agent, Système Embarqué, Panne, Incertitude, Ré-ordonnancement, Makespan, Robustesse, Stabilité.

Abstract

Scheduling problems in the industrial sector are the most studied optimization problems. However, resolution approaches are not suited to a dynamic and increasingly uncertain environment. The work presented in this thesis is aiming for solving the Flexible Job Shop Scheduling Problem (FJSSP) without and with uncertainties. We propose a Particle Swarm Optimization (PSO) algorithm as well as two multi-agent architectures based on PSO called MAPSO1 and MAPSO2 to solve the FJSSP problem without uncertainty. Testing our algorithms on different benchmarks has shown the effectiveness of the obtained results. We are also proposing to use embedded systems for the deployment of an intelligent and cooperative optimization system based on PSO. Thus the MAPSO2 architecture has been distributed on a physically distributed system comprising two embedded systems. Several improvements have been proposed giving rise to MAPSO2 + and MAPSO2 ++. A predictive approach based on PSO called 2s-PSO is also developed to solve the FJSSP problem with consideration of machine failures. Experimental results confirmed that the 2s-PSO algorithm increases the robustness and stability of the solution. As a final contribution, we proposed a rescheduling heuristic to repair schedules in the event of failure under uncertainties.

Key-words : FJSSP, PSO, Multi Agent System, Embedded System, Machine Breakdown, Uncertainty, Rescheduling, Makespan, Robustness, Stability.

Introduction Générale

Les problèmes d'optimisation occupent à notre époque une place prépondérante ; l'optimisation étant nécessaire dans différents domaines de production, du transport, de la logistique, etc.

Comprendre, analyser et formuler un problème d'optimisation nécessite en premier lieu une définition des paramètres, des variables, de l'espace de recherche ainsi que des fonctions à optimiser. Une fois le problème modélisé et formulé, une méthode adaptée pour la résolution du problème posé sera choisie.

A ce niveau, différentes approches peuvent être choisies et utilisées pour obtenir des solutions optimales en utilisant des heuristiques et des méta-heuristiques telles que la recherche tabou, l'algorithme génétique et l'algorithme d'optimisation par essaims particulaires.

Le chercheur est confronté à des difficultés concernant le choix d'une méthode efficace capable de produire une solution de bonne qualité en un temps de calcul raisonnable et le réglage optimal des divers paramètres d'une méta-heuristique en raison de l'inexistence de théorèmes de convergence applicables aux différents types de problèmes. Ces approches ont toutefois des limites, elles ne sont pas adaptées à des situations dynamiques. L'adaptation des méthodes de recherche opérationnelle pour les systèmes dynamiques soulèvent beaucoup des problèmes. C'est le contexte général de nos travaux de recherche.

Les problèmes d'ordonnancement dans le secteur industriel, sont en tête des problèmes d'optimisation les plus étudiés. Améliorer leur productivité et renforcer leur place sur le marché est un souci permanent pour les industriels. Cette thèse s'articule autour du problème d'ordonnancement dans les ateliers flexibles de production (Flexible Job Shop SchedulingProblem FJSSP).

De très nombreuses études se sont intéressées à la résolution de problèmes déterministes. Les auteurs considéraient dans ce cas que les données du problème sont parfaitement connues, et proposent des méthodes pour trouver un ordonnancement statique. Néanmoins, dans un atelier de production plusieurs sortes d'incertitudes peuvent surgir comme la variabilité du temps de traitement d'une opération, la survenue d'une panne machine, l'arrivée d'un nou-

veau job, etc. Ceci rend l'ordonnancement initialement conçu inexécutable et irréalisable. Par conséquent le caractère incertain ne doit pas être négligé lors de la résolution du problème d'ordonnancement.

L'objectif est de proposer un système souple, capable de prendre les décisions d'ordonnancement et de faire face promptement aux imprévus et aux incertitudes internes liées aux pannes. Nous proposons de distribuer le processus d'optimisation en utilisant les systèmes embarqués. Ainsi la solution finale sera déléguée par des entités décentralisées intelligentes et communicantes.

Comme première contribution, nous proposons un algorithme PSO (Particle Swarm Optimization) pour traiter le problème FJSSP sans incertitudes puis deux architectures multi agent à base de PSO nommé Multi Agent Particle Swarm Optimization 1 (MAPSO1) et Multi Agent Particle Swarm Optimization 2 (MAPSO2) pour décentraliser la décision. Néanmoins en utilisant le système multi agent la distribution reste virtuelle car les agents sont déployés sur la même machine. Nous proposons donc de la distribuer sur un système physiquement distribué comportant des systèmes embarqués. D'autres nouvelles améliorations d'architecture sont proposées à savoir MAPSO2+ et MAPSO2++. Les entités doivent communiquer et coopérer entre elles afin que ces solutions soient cohérentes et faisables. Nous discutons aussi l'ajout de capteurs pour ajuster et régler la communication entre les agents distants.

Cette thèse s'intéresse aux processus d'ordonnancement et de ré-ordonnancement dans un système flexible de production avec présence d'incertitudes internes, liées aux pannes de machines. Nous proposons une méthode préventive à base de PSO nommée two stage Particle Swarm Optimization (2s-PSO).

En raison de l'existence de machines identiques, les pièces à usiner peuvent avoir des routages alternatifs (Routing flexibilité). Nous utilisons cette caractéristique pour la création d'une nouvelle heuristique de ré-ordonnancement. Nous utilisons le même codage de particule avec une flexibilité de taille.

La thèse s'articule autour de quatre chapitres :

Le premier passe en revue, les problèmes d'ordonnancement, leurs principales caractéristiques, les types d'ateliers de production, ainsi que les différentes méthodes, exactes et approchées, pouvant être utilisées pour résoudre ce type de problème d'optimisation. On décrit par la suite le contexte de notre recherche et les notions nécessaires à sa compréhension en passant en revue les différentes méthodes existantes pour résoudre le problème d'ordonnancement de type job shop flexible sans incertitude. Nous proposons de structurer cet état de l'art selon le type d'architecture utilisée centralisée ou bien distribuée. Une revue de la littérature

est faite en premier lieu sur l'ordonnancement sous incertitude, les différents types d'incertitudes ainsi que les techniques de leur modélisation et en deuxième lieu, sur les approches de l'ordonnancement dans un milieu incertain. Nous structurons l'état de connaissances selon la classification de [CCAT14] qui distingue trois classes : les approches proactives, les approches réactives et les approches hybrides. Nous décrivons par la suite les techniques de ré-ordonnancement utilisées pour réparer un ordonnancement sous incertitudes.

Au deuxième chapitre, nous exposons la modélisation mathématique du problème de job shop flexible, décrivant les variables, les paramètres, les contraintes ainsi que la fonction objective puis nous nous focalisons sur la description de l'algorithme PSO proposé. Nous détaillerons par la suite le codage de la particule utilisée, les méthodes d'initialisation, y compris celle proposées, les topologies de voisinage ainsi que l'algorithme proprement dit.

Nous présenterons les deux nouvelles architectures multi agent proposées basées sur l'algorithme PSO nommé MAPSO1 et MAPSO2. Une description détaillée de leurs structures, du nombre d'agents utilisés, le rôle de chaque agent créé et les connaissances propres de chaque agent(les agents qu'il connaît et avec lesquels il peut communiquer). Nous décrirons dans une dernière partie les résultats de la phase expérimentale en procédant à la vérification de l'algorithme PSO proposé pour l'ordonnancement déterministe sur des benchmarks existants dans la littérature et nous effectuerons une comparaison des deux architectures MAPSO1 et MAPSO2.

Le chapitre suivant sera consacré à l'implémentation embarquée de l'algorithme d'optimisation par essaims particulaires. Le déploiement de l'architecture MAPSO2 sur un système physiquement distribué est réalisé afin de comparer les performances par rapport MAPSO2 centralisé. Les limites observées lors de la comparaison nous ont mené à proposer une version améliorée nommé MAPSO2+ pour augmenter la flexibilité et l'adaptabilité de l'architecture à des situations dynamiques. L'intégration de cette architecture sur le système physiquement distribué est aussi présentée dans ce chapitre. Nous détaillerons une autre nouvelle amélioration de l'architecture, nommée MAPSO2++, qui discute à la fois l'intégration de système embarqué et l'ajout des capteurs de températures/humidité dans le déploiement d'un processus d'optimisation flexible et intelligent (PSO dans notre cas). Une contribution dans ce contexte sera présentée consistant à la réduction du temps d'exécution de l'architecture physiquement distribuée tout en jouant sur la configuration de la taille du sous essaim délégué devant être traité par chaque agent exécutant intégré dans un système embarqué.

Dans le dernier chapitre, nous proposerons une nouvelle méthode d'ordonnancement prédictif sous incertitudes basée sur l'algorithme PSO. L'objectif est d'obtenir une solution robuste

de bonne performance peu sensible à ces incertitudes. Nous détaillerons les différentes étapes de l'algorithme proposé nommé 2s-PSO. Un exemple illustratif sera présenté. Puis nous validerons l'algorithme proposé pour l'ordonnancement robuste par la simulation, pour juger la qualité de la robustesse face à plusieurs types d'incertitudes. Nous détaillerons par la suite la nouvelle heuristique de ré-ordonnancement proposée à base de PSO. Afin de valider notre méthode, nous avons implémenté la technique RSR et accompli une étude comparative entre le deux résultats en les simulant sur différents types de pannes. En conclusion nous présenterons un bilan final des résultats, les limites et les perspectives de recherches à l'issue de cette thèse.

Chapitre 1

Ordonnancement du job shop flexible

1.1 Généralités sur l'ordonnancement

1.1.1 Introduction

Ce chapitre est introductif à notre thème de recherche : le problème d'ordonnancement dans les ateliers de production. Nous présentons en premier lieu les points essentiels concernant l'ordonnancement, les critères d'optimisation ainsi que les différents types des ateliers de production (Flow shop, job shop, etc).
Dans un deuxième temps, une description des principales méthodes d'optimisation utilisées dans la littérature est donnée.
Enfin nous développons largement le problème d'ordonnancement de type job shop flexible en faisant un tour d'horizon sur les méthodes de résolution existantes dans la littérature.
Notre état de l'art est divisé en deux parties : ordonnancement déterministe et ordonnancement dynamique sous incertitudes.

1.1.2 L'ordonnancement

1.1.2.1 Définition

Au cours des dernières années, les problèmes d'ordonnancement ont fait l'objet de nombreuses études de recherche. En effet, les champs d'application de la théorie d'ordonnancement sont diverses, notamment dans les systèmes informatiques, dans la gestion des projets, l'organisation des activités de services, le transport, l'administration ainsi que dans les systèmes de production.
En informatique le processeur exécute les tâches selon un ordre bien déterminé qui dépend du type d'ordonnancement sous le pilotge de l'ordonnanceur. En gestion de projet, l'ordonnancement consiste à organiser dans le temps et à déterminer les dates d'exécution des activités constituant le projet.

Ordonnancer le fonctionnement d'un système industriel de production consiste à gérer l'allocation des ressources au cours du temps, à déterminer les séquences d'opérations en fixant leurs dates d'exécution tout en optimisant au mieux un ensemble de critères.
Plusieurs définitions de l'ordonnancement sont données :

— " *Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution* " [CC88].

— " *Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, ...) et de contraintes portant sur l'utilisation et la disponibilité de ressources requises* " [EL99].

D'une manière plus simple, l'ordonnancement est la programmation dans le temps de l'exécution d'une série de tâches (activités ou opérations) sur un ensemble de ressources physiques (machines), cherchant à optimiser un ou plusieurs objectifs (coûts, délais, qualité, etc), tout en respectant des contraintes.
Résoudre un problème d'ordonnancement, consiste à choisir pour chaque opération une date de début, de telle sorte que les contraintes du problème soient respectées (la solution est alors dite admissible ou réalisable) et qu'un ou plusieurs critères donnés soient optimisés.
Le mot ordonnancement, ou "schedule" dans la terminologie anglaise, désigne une solution du problème d'ordonnancement. Cette solution précise principalement ces trois importantes caractéristiques :

— l'affectation des tâches aux ressources nécessaires,

— le séquencement indiquant l'ordre de passage des tâches sur les ressources,

— le datage, qui précise les temps de début et de fin d'exécution de chaque tâche sur les ressources.

Quel que soit le contexte d'ordonnancement, la résolution nécessite la détermination des tâches, des ressources, des contraintes et des critères à prendre en considération lors du processus d'optimisation. Nous présenterons dans les paragraphes suivants une définition détaillée de chacun de ces éléments.

1.1.2.2 Les tâches

Une tâche est une entité élémentaire de travail (job). Elle fait référence à une opération caractérisée dans le temps, par une date de début et/ou de fin, et dont la réalisation nécessite une durée opératoire préalablement définie. Un job est constitué d'un ensemble d'opérations.

La réalisation ou l'exécution d'une tâche requiert l'utilisation d'une ou plusieurs ressources. Ils existent des tâches morcelables (préemptibles) pouvant être exécutées par morceaux, et d'autres non morcelables devant être exécutées en une seule fois et sans interruption.

1.1.2.3 Les ressources

Une ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche spécifique ou de plusieurs tâches. Il existe deux familles de ressources :

- **Les ressources renouvelables** : ce sont les ressources réutilisables après la fin d'exécution des opérations. En d'autres termes, ce sont les ressources qui redeviennent disponibles après leur utilisation comme les machines, les opérateurs, les outils de production, etc. Dans cette famille, on distingue deux catégories : les ressources disjonctives qui ne peuvent exécuter qu'une seule tâche à la fois et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité. Quelle que soit la nature de la ressource renouvelable, sa disponibilité peut varier au cours du temps.
- **Les ressources consommables** : comme leur nom l'indique ce sont les ressources, qui après avoir été allouées à une tâche, ne sont plus disponibles, c'est le cas des matières premières, de l'argent, de l'énergie, etc.

1.1.2.4 Les contraintes

La notion de contrainte est relative à un ensemble de variables de décision. Selon [EL99], les contraintes expriment les restrictions que peuvent prendre conjointement une ou plusieurs variables de décision. Elles représentent les limites imposées par l'environnement.
Il existe plusieurs contraintes qui diffèrent d'un problème à un autre :

- **Les contraintes temporelles**, ou de localisation temporelle, représentent des restrictions sur les valeurs que peuvent prendre les variables de décision temporelles d'ordonnancement. Ces variables concernent les décisions de localisation des tâches dans le temps. On peut citer comme exemple :
  - La contrainte de précédence, appelée encore contrainte de succession, qui lie le début d'une activité à la fin d'une autre. Certaines tâches doivent être achevées avant une date préalablement fixée. Ces contraintes sont imposées généralement par la cohérence technologique (les gammes opératoires dans le cas d'ateliers) décrivant le positionnement devant être respecté entre les tâches.

— Les contraintes de dates butoirs, qui exigent le respect d'un échéancier préalablement fixé. Dans le plan de production par exemple, le respect d'une date de livraison est primordial. Dans certain cas, d'autres contraintes sont imposées comme celle de synchronisation, de simultanéité [EL99].

- **Les contraintes de ressources** concernent l'utilisation de ces ressources et leurs disponibilités. La nature des ressources différentes engendre également des contraintes d'utilisation différentes. En effet, les ressources disjonctives induisent une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource alors que les ressources cumulatives impliquent la limitation du nombre de tâches à réaliser en parallèle. La contrainte de disponibilité détermine la quantité de ressources disponibles au cours du temps.

1.1.2.5 Les critères d'optimisation

Les critères d'optimisation constituent les objectifs qu'on souhaite optimiser. Dans le processus de résolution du problème d'ordonnancement, chaque critère est souvent représenté par une fonction objective à optimiser (minimisation ou maximisation). Ces exigences qualitatives et quantitatives à satisfaire, permettent d'évaluer la qualité de l'ordonnancement établi. Evaluer la qualité de l'ordonnancement revient à mesurer sa fonction objective. Le choix de la solution la plus satisfaisante dépend de la satisfaction des exigences du problème étudié.
Les critères dépendant d'une application donnée sont très nombreux. On trouve par exemple :

- **Les critères liés à l'utilisation des ressources** :
  - — Maximiser l'utilisation de la machine la moins gourmande en énergie.
  - — Minimiser la charge d'une machine ou le temps d'inactivité de l'ensemble des machines.
  - — Minimiser l'ensemble de ressources nécessaires pour réaliser un ensemble de tâches.
  - — Maximiser l'utilisation moyenne de ressources. Ce critère met en évidence les périodes creuses et les périodes d'utilisation à plein régime du système de production et nous renseignera sur les taux d'utilisation des ressources.
- **Les critères liés aux temps** :
  - — La minimisation des dates d'achèvement des activités.
  - — La minimisation des retards sur les dates d'achèvement des tâches.

— La minimisation du temps de transport dans les ateliers de production.

En fonction du nombre d'objectifs on parle d'optimisation mono-objective (l'optimisation d'un seul critère) et d'optimisation multi-objective (l'optimisation de plusieurs critères). En optimisation multi-objective, ces critères peuvent être combinés ensemble pour former une fonction multi-objective à optimiser. Par contre, la satisfaction de plusieurs critères à la fois est souvent délicate et peut mener à des situations contradictoires [BD93].

Nous nous intéressons aux problèmes d'ordonnancement dans les ateliers flexibles de productions. Dans ce qui suit, nous présenterons la notion d'atelier de production ainsi que les types d'ateliers.

1.1.3 Les systèmes flexibles de production

1.1.3.1 Définition

L'objectif majeur des entreprises est de conserver ou d'améliorer leur position dans le marché. Pour ce faire, les industriels doivent augmenter la productivité, essentiellement en quantité et en qualité de production, tout en réduisant les coûts au maximum.

Le développement de nouvelles technologies dans le monde manufacturier a donné naissance à l'émergence d'une nouvelle classe de systèmes de production : les systèmes flexibles de production (SFP) ou Flexible Manufacturing system (FMS) selon la terminologie anglaise.

La littérature propose différentes définitions des SFP, on peut citer :

- Celle de [KHL86]

" *Un SFP est un système de production hautement automatisé capable de produire une grande variété de types de produits en utilisant le même équipement et le même système de contrôle* ".

- Celle de [BOP88]

" *Un SFP combine un ensemble de machines de production à commande numérique interreliées par un système automatisé de transport/manutention (T/M) et une infrastructure informatique permettant la gestion et le contrôle du système* ".

- Celle de [ML93]

" *Un SFP est un système de production capable de produire différents types de pièces, il est composé d'un ensemble de machines à commande numérique interreliées par un système automatisé de T/M. La gestion et le contrôle de ce système sont informatisés*". Cette définition intègre l'aspect fonctionnel et structurel.

En conclusion et selon ces différentes définitions, tout système flexible de production exige la

présence des trois sous-systèmes suivants : un système de fabrication composé de machines, un système automatisé de transport/manutention (tapis roulants avec embranchements, robots dédiés au déplacement des pièces, etc) et un système informatique de contrôle assurant des fonctions de planification, suivi de la production afin de surveiller et de piloter le déroulement de la production.
Les systèmes de production sont très variés. Le nombre de machines mises en jeu ainsi que leurs dispositions détermine un type d'atelier différent.
On distingue : type single machine shop, type flow-shop, type job-shop, type open-shop, etc.

1.1.3.2 Les types des systèmes flexibles de production

1.1.3.2.a Single Machine Shop

C'est l'atelier de production le plus simple constitué d'une seule machine pour exécuter toutes les opérations, la machine fonctionne en continu nécessitant à chaque fois une nouvelle configuration pour la fabrication d'un produit différent. Par conséquent, le fonctionnement d'une seule machine est dicté par la séquence dans laquelle les produits passent par la machine.

1.1.3.2.b Flow-Shop

Ce sont des ateliers où la ligne de fabrication est constituée de plusieurs machines en série ; toutes les opérations passent par les machines dans le même ordre. Ils sont appelés également ateliers à cheminement unique.
Une gamme opératoire impose un ordre de passage des produits sur les machines, et donc un ordre des opérations associées à chaque travail.
Dans ce type d'atelier, chaque travail (job) a une gamme opératoire identique.
Il existe le type flow-shop hybride où une machine peut exister en plusieurs exemplaires identiques fonctionnant parallèlement.

1.1.3.2.c Job-Shop

Appelés également ateliers à cheminement multiple, ce sont des ateliers où les opérations sont réalisées selon un ordre bien déterminé, variant selon la tâche à exécuter. Chaque job a sa gamme opératoire propre en utilisant cet atelier.
Une extension du modèle job-shop classique est le job-shop flexible ; sa particularité réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble

d'opérations. Nous détaillerons le problème d'ordonnancement dans le job shop flexible dans le chapitre suivant.

1.1.3.2.d Open-Shop

Ce sont des ateliers à cheminement libre. Ce type d'atelier est moins contraignant car l'ordre des opérations n'est pas fixé a priori. L'ordre de passage sur les machines est libre pour tous les travaux, i.e., il n'existe aucune relation de précédence entre les opérations d'un même travail.

1.1.3.3 Les types de flexibilité dans les systèmes de production flexibles

Le concept de flexibilité est capital et intimement lié aux systèmes de production. En effet, plusieurs sortes de flexibilités sont présentes. On peut citer :

- La flexibilité du système de fabrication ou encore la flexibilité des machines qui peuvent exécuter différents types d'opérations avec un temps de changement raisonnable.
- La flexibilité du réseau des mouvements de produits entre les unités de production ou encore flexibilité de manutention des matériaux à savoir la capacité d'offrir des itinéraires de transfert alternatif à différents types de produits.
- La flexibilité des opérations, la possibilité de fabriquer le même produit en changeant la séquence d'opérations.
- La flexibilité de séquence de machine (Machine-sequence flexibility) qui est la capacité de produire un produit avec une séquence de machines différentes. Ce type de flexibilité est connu aussi sous le nom de flexibilité de routage "Routing Flexibility" à ne pas confondre avec la flexibilité de transport ou le routage entre les machines.
- La flexibilité liée aux incertitudes qui apparaissent aussi bien dans des activités à long terme comme les prévisions des ventes, que dans des aspects plus opérationnels comme la gestion des pannes ou la mise en place de politiques de maintenance préventive.
- La possibilité d'introduire de nouveaux types de produits dans la production.

Dans le sens le plus large, la flexibilité peut être définie comme " la capacité de production pour faire face aux changements internes et externes " [PC94]. Nous nous intéressons aux problèmes d'ordonnancement dans l'atelier de type job shop flexible connu en terminologie anglaise par "Flexible Job Shop Scheduling Problem FJSP".
Nous présenterons dans la section suivante les méthodes de résolution d'un problème d'optimisation et en particulier le problème d'ordonnancement.

1.1.4 Les méthodes de résolution

La difficulté de résolution des problèmes d'ordonnancement, émane du grand nombre d'entités à gérer, de la multiplicité des tâches, des contraintes citées précédemment ainsi que la présence des incertitudes. La majorité des problèmes d'ordonnancement sont en général des problèmes d'optimisations.
Pour faire face aux problèmes d'optimisation, des techniques de recherche opérationnelle sont utilisées. En présence d'un problème d'optimisation, le choix d'une méthode efficace capable de produire une solution optimale en un temps de calcul raisonnable, est une difficulté à laquelle tout décideur est confronté.
Nombreuses sont les méthodes proposées dans la littérature pour l'ordonnancement de la fabrication dans les ateliers.
Ces méthodes de résolution développées peuvent être classées en deux catégories : les méthodes exactes qui garantissent la complétude de la résolution et les méthodes approchées qui perdent la complétude pour gagner en efficacité.

1.1.4.1 Les méthodes exactes

Une méthode exacte est une méthode qui fournit une solution optimale pour un problème d'optimisation. Les méthodes exactes sont exigeantes en ressources, et elles requièrent des temps d'exécutions conséquents. En effet, une manière intuitive de résoudre les problèmes d'optimisation discrets consiste à énumérer une partie ou toutes les solutions admissibles. Néanmoins, pour un grand nombre de problèmes, cette énumération s'avère impossible à cause des temps de calcul prohibitifs entrainés par l'énumération.
L'utilisation de ce type de méthodes s'avère particulièrement intéressante dans les cas des problèmes de petites tailles.
Souvent quand il s'agit des problèmes de grande taille ou de taille critique, plusieurs chercheurs s'orientent vers l'utilisation des méthodes exactes en simplifiant ou éliminant certaines contraintes pour obtenir des bornes inférieures ou supérieures. Ces bornes constituent un point de comparaison et servent à évaluer la qualité des différentes méthodes approchées proposées pour la recherche de solutions admissibles.
La méthode par séparation et évaluation (branch and bound), la programmation linéaire ou la programmation dynamique (la méthode simplexe) sont des exemples de ce type de méthodes. A côté des algorithmes classiques exacts de résolution, des algorithmes approchés

ont été développés. Dans ce qui suit nous présentons les principales méthodes approchées utilisées en recherche opérationnelle.

1.1.4.2 Les méthodes approchées

L'espace de recherche de problème d'optimisation de taille critique est énorme et multidimensionnel menant à un processus de recherche laborieux. Plusieurs chercheurs se sont orientés vers l'utilisation de méthodes de résolution approchées, comme les heuristiques ou les métaheuristiques.
A l'inverse d'un algorithme exact permettant de garantir l'optimalité de la solution obtenue, un algorithme approché tente de s'en approcher le plus possible. Ce type de méthodes fournit des solutions de bonnes qualités en un temps de calcul raisonnable.

1.1.4.2.a Les heuristiques

Ce sont des méthodes très simples à mettre en oeuvre et qui dépendent du domaine d'utilisation. Le principe de base de ce type de méthode est de construire progressivement, en intégrant des stratégies de décision, une solution proche de celle optimale tout en cherchant à avoir un temps de calcul raisonnable [PB01].
Les stratégies de décisions sont souvent des critères d'optimisation locaux (une règle de construction donnée) menant à des solutions admissibles (réalisables) de qualité relativement bonne. Parmi ces stratégies, on peut citer FIFO (First In First Out) où la première tâche arrivée est la première à être ordonnancée, EDD (Earliest Due Date) où la tâche ayant l'échéance la plus imminente est prioritaire, SPT (Shortest Processing Time) où la tâche ayant le temps opératoire le plus court est traitée en premier, LPT (Longest Processing Time) où la tâche ayant le temps opératoire le plus important est traitée en premier.
On note que la qualité d'une heuristique est évaluée sur la base de plusieurs jeux d'instances de taille variable afin de déterminer la déviation moyenne du critère par rapport à sa valeur optimale et mesurer le temps de calcul en fonction de la taille du problème.

1.1.4.2.b Les métaheuristiques

Les méthodes dites, métaheuristiques sont des méthodes de recherche générales "génériques" indépendantes du problème étudié, dédiées à la résolution d'une large gamme de problèmes d'optimisation.
Le mot métaheuristique se compose de deux mots grecs :

— heuristique venant du verbe heuriskein et signifie "trouver" ;

— meta signifiant "au-delà" et indiquant un passage d'un niveau à un niveau "supérieur" pour étudier ou manipuler des informations de niveau inférieur.

Une métaheuristique n'est pas propre à un problème précis, mais adaptable et applicable à une large classe de problèmes. Elle consiste à explorer l'espace de solutions possibles afin de trouver une solution satisfaisante dans un temps de calcul raisonnable aussi bref que possible. Ces approches suscitent un intérêt croissant, de nombreuses métaheuristiques ont été conçues pour la résolution d'une famille de problèmes de plus en plus complexes dites NP-difficile tels que le problème d'ordonnancement.

Les métaheuristiques sont ainsi des méthodes de recherche générales, dédiées aux problèmes d'optimisation difficile. Elles sont, en général, présentées sous forme de concepts.

Parmi les métaheuristiques, on peut citer celles basées sur la recherche locale, tels que le recuit simulé et la recherche Tabou, celles basées sur les algorithmes évolutionnistes tels que les algorithmes génétiques et enfin les algorithmes basés sur la recherche globale tels que les algorithmes de colonies de fourmis et les algorithmes d'optimisation par essaim particulaires.

Dans ce qui suit, nous présentons brièvement les méthodes basées sur la recherche locale, la recherche tabou et l'algorithme génétique ; l'optimisation par essaim particulaires sera présentée en détail dans le chapitre 2.

a) **Les méthodes basées sur la recherche locale**

appelées aussi les méta-heuristiques à base de voisinage. Il s'agit d'une procédure de recherche itérative qui commence par une première solution réalisable, puis l'améliore progressivement vers une autre voisine en appliquant une série de modifications (ou mouvements), dans le but d'améliorer un critère bien déterminé.

Pour cela il faut introduire une structure de voisinage qui consiste à spécifier un voisinage pour chaque solution. Ainsi, à chaque itération, la recherche s'oriente vers une nouvelle solution réalisable qui diffère légèrement de la solution courante en remplaçant celle-ci par une meilleure située dans son voisinage. La recherche se termine dès qu'il n'existe plus de meilleure solution dans le voisinage. Dans ce cas on dit qu'un optimum local est rencontré.

L'inconvénient majeur de cette méthode est de rester bloquée dans des endroits (minimums locaux). En effet, la qualité de cet optimum local trouvé est souvent assez médiocre et dépend fortement de l'ensemble des transformations (mouvements) considérées à chaque itération.

Des méthodes de recherche locale plus sophistiquées ont été développées afin de contour-

ner ces minimums locaux. Ces approches autorisent de dégrader provisoirement le critère dans le but d'atteindre des endroits plus prometteurs menant à des solutions optimales dites minimums globaux.
Le recuit simulé [KGV83] et la recherche tabou [Glo89] ; [Glo90] sont des exemples de ces méthodes.

- **Le recuit simulé** : L'algorithme du recuit simulé a été proposé par [KGV83] et inspiré d'un processus utilisé en métallurgie (appelé le recuit). Afin d'améliorer la qualité d'un solide et atteindre des états de basse énergie, on chauffe à des températures de plus en plus élevées, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement. En se basant sur cette constatation, une température initiale sera attribuée au critère à optimiser. L'algorithme de récuit simulé cherche aléatoirement à chaque itération une solution voisine qui améliore le critère à optimiser. Une nouvelle solution voisine de coût supérieur à celui de la solution courante ne sera pas forcément rejetée. En effet, les mouvements non améliorés sont autorisés avec une certaine probabilité en tenant compte de la différence entre les coûts ainsi que du facteur température T associé au critère.

- **La recherche Tabou** : Il s'agit d'une méta-heuristique d'optimisation combinatoire présentée par Glover [Glo89] , [Glo90]. C'est une autre méthode se basant principalement sur la notion de voisinage et de mouvements interdits.
  Le voisinage d'une solution est un ensemble de solutions réalisables que l'on peut obtenir en effectuant un mouvement prédéfini (permutation de deux opérations par exemple). Partant d'une solution initiale, le processus itératif de l'algorithme, tente de converger vers la solution optimale en remplaçant à chaque itération la solution courante par la meilleure solution trouvée dans son voisinage même si ce choix entraîne une détérioration de la valeur de la solution courante.
  Les solutions explorées sont ajoutées dans une liste dite "la liste tabou" de taille fixée à l'avance. Cette liste tabou sert à court terme de mémoire et empêchera tout déplacement vers les dernières solutions visitées. Le mécanisme consiste à interdire, (d'où le nom de tabou) l'accès aux dernières solutions.
  Il est à noter que la configuration du nombre de voisinage, la taille de la liste tabou et les conditions d'arrêt peuvent être très différentes suivant le type de problème à traiter.

b) **L'algorithme Génétique**

Les algorithmes génétiques font partie de la famille des méta-heuristiques à base de population, connus sous l'appellation des algorithmes évolutionnaires. A l'inverse des méthodes de recherche locale qui font intervenir une solution unique, les méthodes évolutives manipulent un groupe de solutions admissibles à chacune des étapes du processus de recherche.

Parmi les méthodes évolutionnistes, nous citons les algorithmes génétiques.

L'algorithme génétique s'inspire de l'évolution naturelle des espèces en exploitant la simulation des mécanismes de variation et de sélection effectués dans les processus évolutifs naturels pour résoudre des problèmes [Hol92].

Dans cet algorithme, l'analogie entre la théorie de l'évolution consiste à considérer les solutions du problème à optimiser comme des chromosomes et des individus soumis à l'évolution.

Le principe d'un algorithme génétique est simple. On part d'une population initiale de solutions du problème (ensemble d'individus). Cette population évolue progressivement vers une population contenant les meilleurs chromosomes fils héritants des meilleures qualités des chromosomes parents à chaque génération.

L'algorithme s'appuie alors sur deux phases : la sélection (favorisant les individus ayant un meilleur fitness), et la recombinaison qui génère une nouvelle population d'individus enfants en conservant les "bonnes" caractéristiques de leurs parents.

Les nouveaux individus sont générés par le mécanisme de croisement et de mutation et évoluent par génération successive de telle sorte qu'ils soient plus performants que leurs prédécesseurs c'est-à-dire ils optimisent mieux la valeur de la fonction objectif (fonction fitness). La difficulté de la procédure réside dans la représentation ou le codage des solutions, la configuration de la taille de la population ainsi que le critère d'arrêt.

Une autre méthode à base de population est l'algorithme de colonies de fourmis.

c) **Les algorithmes de colonies de fourmis**

Cette approche inventée par Dorigo et ses collaborateurs [DG97] s'efforce 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.

Les algorithmes de colonies de fourmis sont nés d'une constatation simple : les insectes

sociaux, et en particulier les fourmis, communiquent entre eux de manière indirecte par le dépôt de substances chimiques sur le sol, appelées phéromones, pour atteindre depuis leur nid un objectif précis "une source de nourriture". Le phéromone marque ainsi leurs trajets et permet à leurs congénères de les suivre.

Grâce à ce moyen de communication, les fourmis arrivent à résoudre collectivement des problèmes trop complexes, notamment les problèmes de choix du plus court chemin entre une source de nourriture et leur nid.

d) **L'optimisation par essaim particulaires**

La méthode d'Optimisation par Essaim Particulaires (OEP) ou Particle Swarm Optimization (PSO) selon la terminologie anglaise, a été proposée en 1995 par James Kennedy et Russel Eberhart [KE95]. L'algorithme cherche à simuler la capacité des oiseaux à voler de façon synchrone et leur aptitude à changer brusquement de direction. Comme l'algorithme génétique, l'algorithme d'optimisation par essaim particulaire fonctionne à base de population d'individus. Chaque individu est appelé "particule" et l'ensemble est appelé essaim ou " swarm". Mais elle se distingue par le fait qu'elle n'utilise pas les opérateurs d'évolution (sélection, mutation, croisement).

L'OEP ou PSO est un algorithme itératif, à chaque itération les particules changent leur position (état) avec le temps en se basant sur leur expérience et des connaissances de ses meilleures voisines. Bien que chaque individu ait une intelligence limitée, le groupe possède une organisation globale, une dynamique de déplacement relativement complexe et peuvent trouver ensemble des solutions à un problème grâce à la collaboration des individus entre eux. L'intelligence globale de l'essaim résulte, donc, des interactions entre ses différentes particules. Cet algorithme sera détaillé dans le chapitre suivant.

Plusieurs méta-heurstiques sont développées afin de résoudre un problème d'optimisation. La difficulté réside alors dans le choix de la méthode la plus appropriée. La résolution des problèmes d'ordonnancement dans les ateliers de production, constitue une des principales préoccupations des industriels.

Dans ce qui suit nous présenterons un état d'art sur les méthodes développées pour résoudre le problème de job shop flexible FJSP.

1.2 Ordonnancement déterministe de FJSP

Le problème d'ordonnancement de job shop flexible FJSP est un problème de type "NP-difficile" ou "NP-hard" [CMM03]. Le mot "NP" vient de l'abréviation de "Non deterministic Polynomial time". Les problèmes de ce type ne peuvent à priori être résolus en un temps polynomial que par des méthodes approchées (heuristiques) ; au cours de leur exécution, ces algorithmes font des choix dont l'optimalité n'est pas démontrable.
Le problème du FJSP est plus difficile que le job shop classique car il introduit un niveau de décision supplémentaire à côté de l'ordonnancement, à savoir l'affectation des opérations, c'est à dire décider pour chaque opération quelle est la machine qui doit la traiter parmi celles disponibles. La difficulté consiste alors à choisir les meilleures affectations des machines aux opérations et le meilleur ordre afin d'optimiser un objectif [GJ09].
Plusieurs travaux ont été effectués afin d'aborder le problème de job shop flexible déterministe sans considération des incertitudes. Dans la section suivante nous présenterons un tour d'horizon des méthodes existantes dans la littérature. Notre état d'art est structuré selon l'architecture et le modèle utilisé lors du développement de la méthode de résolution. En effet, les approches précédemment présentées peuvent être soit centralisées dans une seule entité globale soit distribuées sur plusieurs entités décisionnelles.

1.2.1 Les approches centralisées

Globalement, les systèmes d'ordonnancement les plus répandus s'appuient sur une politique centralisée de résolution (en-ligne ou hors-ligne) des problèmes d'ordonnancement avec ou sans prise en compte des incertitudes. En effet, le processus de résolution des problèmes d'ordonnancement est classiquement assimilé à un problème de décision globale car la fonction d'ordonnancement gère l'organisation de la totalité des ressources, et requiert une connaissance globale des paramètres du système. Cette organisation centralisée des décisions suppose alors l'existence d'une entité qui supervise les activités des ressources, et prend des décisions globales, celles-ci devant être respectées par l'ensemble des ressources gérées.
Récemment, l'algorithme génétique (AG) a été adopté avec succès pour résoudre FJSP. Il s'appuie sur la théorie Darwinienne de la sélection naturelle des espèces : les individus qui ont hérité de caractères bien adaptés à leur milieu ont tendance à vivre assez longtemps pour se reproduire, alors que les plus faibles ont tendance à disparaître. C'est une méthode qui se base sur l'évolution des populations de génération en génération afin de trouver la meilleure solution du problème. Chaque population est composée par un ensemble d'individus appelés

chromosomes. Chaque chromosome représente une solution du problème. Cette population évolue durant une succession d'itérations, appelées générations. Au cours de chaque génération, une série d'opérateurs sont appliqués aux individus, pour créer la population de la génération suivante. Les chromosomes fils (enfants) issus du mécanisme de croisement et de mutation remplacent les chromosomes parents.

Récemment, plusieurs travaux dans la littérature ont tenté de développer des algorithmes génétiques de façon centralisée comme [KHB02] ;[PMC08] ; [ZGS11]. [KHB02]ont proposé un algorithme génétique pour résoudre FJSP mono-objectif et multi-objectif. Une approche de la localisation a été aussi proposée pour déterminer l'affectation des machines.

[PMC08] ont proposé un algorithme génétique avec plusieurs techniques de création de la population initiale telle que l'approche par localisation, l'approche de recherche du minimum global et la sélection aléatoire. Ils ont généré la population initiale par ces trois méthodes avec des pourcentages différents afin que la qualité de cette population soit bonne et ceci affecte la convergence de l'algorithme [PMC08].

[MlSlH10] ont proposé un AG hybride afin d'optimiser les différentes fonctions objectifs suivantes : minimiser le makespan qui est la date fin d'exécution de la dernière tâche dans l'atelier de production, réduire la charge totale du travail, tout en minimisant la charge du travail de la machine la plus chargée [MlSlH10].

[ZGS11] ont développé un algorithme génétique et ont proposé deux méthodes d'affectation : la méthode de sélection globale (GS) et la méthode de sélection locale (LS) pour générer une population initiale de haute qualité à l'étape d'initialisation et accélérer la vitesse de convergence de l'algorithme.

Tous ces travaux cités précédemments utilisent l'algorithme génétique pour résoudre FJSP de façon centralisée mais ils diffèrent l'un de l'autre dans le codage de la solution, la méthode utilisée pour générer la population initiale, la sélection des meilleurs chromosomes, etc.

En dehors des AG, un autre algorithme à base de population a été adopté avec succès pour résoudre FJSP : c'est l'optimisation par essaims particulaires.

La première étape de cet algorithme est la création de l'essaim c'est-à-dire l'initialisation des particules qui représentent des solutions potentielles au problème d'optimisation. Chacune de ces particules est dotée par un vecteur de position, une vitesse qui permet à la particule de se déplacer et un voisinage c'est-à-dire un ensemble de particules (Neighbours) qui interagissent directement sur la particule, en particulier celle qui a le meilleur critère. Ces solutions survolent dans l'espace de recherche, pour trouver l'optimum global.

Venter et Sobieszczanski ont introduit un algorithme PSO parallèle pour améliorer les per-

formances de l'approche [VSS06]. Les auteurs dans [JCT07] ont proposé une amélioration du PSO en ajoutant la méthode chaotique pour optimiser la meilleure solution globale.
Les auteurs dans [BGWT09] ont proposé un algorithme PSO pour résoudre FJSP avec scission de lot.
Récemment, l'hybridation du PSO avec d'autres méta-heuristiques pour mieux résoudre le FJSP a également été adoptée avec succès. De nombreux travaux ont été proposés pour ce sujet. [XW05] ont introduit une hybridation entre PSO et le recuit simulé afin de résoudre FJSP dans le but de minimiser le makespan [XW05]. [LPX+10] ont proposé une hybridation entre PSO et l'algorithme de la recherche Taboue avec l'utilisation de nouveaux opérateurs de mutation et de croisement de l'algorithme génétique. Tanga et al ont proposé un algorithme hybride qui combine PSO et GA [TZLZ11].
Nous remarquons que différentes métaheuristiques sont utilisées pour résoudre le problème de FJSP de façon centralisée. Elles diffèrent par la structure et le codage de la solution, la création de la population initiale ainsi que les mouvements utilisés pour passer d'une solution à autre, le critère à optimiser.
Nous remarquons aussi qu'un ensemble d'approches se basant sur l'hybridation entre deux méthodes d'optimisation afin d'améliorer la performance de l'algorithme. Cette hybridation va engendrer notamment une augmentation du temps de résolution en vue d'obtenir une meilleure solution.
Dans le cas d'une application embarquée, le temps d'exécution et l'énergie consommée sont deux facteurs très importants et critiques. Nous avons focalisé notre étude sur l'optimisation du facteur temps. La difficulté réside alors dans le choix de la bonne méthode et la bonne configuration pour trouver un compromis entre la qualité de la solution d'une part et le temps de la résolution d'autre part.
Pour se faire, nous avons fait une étude comparative des méthodes centralisées existantes. Notre objectif étant de résoudre le problème de FJSP tout en optimisant le makespan de la solution finale en se basant sur ce critère dans l'étude comparative.
Le tableau suivant résume les valeurs trouvées par les méthodes proposées par les chercheurs.

Comme le montre le tableau 1.1, l'utilisation de l'algorithme génétique et l'algorithme d'optimisation par essaim particulaire permet de trouver des solutions de meilleure qualité en comparaison aux autres métaheuristiques.
L'avantage d'une organisation centralisée réside dans la cohérence globale des décisions avec toutefois un inconvénient majeur : cette architecture est mal adaptée aux environnements

TABLE 1.1: Comparaison entre des métaheuristiques minimisant le Makespan

Travaux	Méthode	Instance	Makespan
[KHB02]	GA	4*5	18
		8*8	16
		10*10	7
[KHB02]	AL+GA	4*5	16
		8*8	15
		10*10	8
[XW05]	PSO+SA	4*5	-
		8*8	16
		10*10	7
[LPX+10]	TS+PSO	4*5	12
		8*8	-
		10*10	7
[GSG08]	hGA	4*5	-
		10*10	7
		8*8	14
[BGWT09]	LSPSO	10*10	5
		8*8	12

dynamiques et aux aléas qui peuvent surgir dans le système. En cas de panne, le système se bloque en attendant la décision issue du serveur exécutant l'algorithme centralisé. Les chercheurs s'orientent vers l'utilisation d'une architecture distribuée. Dans la section suivante nous présentons les travaux qui ont été réalisés dans ce sujet.

1.2.2 Les approches distribuées

Le progrès technologique en matière de systèmes de calculs et de réseaux de communications a permis de modifier le mode d'organisation classique centralisé des systèmes de gestion de production. Des approches de résolution distribuées sont développées. Ces méthodes s'occupent de la répartition et la distribution des calculs d'ordonnancement sur plusieurs acteurs autonomes. Les décisions d'ordonnancement étant réparties, ces approches requièrent toutefois des méthodes spécifiques de gestion afin d'assurer la cohérence globale des différents choix.

A l'inverse de la résolution parallèle, qui définit des algorithmes parallèles implémentés sur plusieurs processus pour une exécution simultanée, la résolution distribuée nécessite l'élaboration d'algorithmes évolués capables de favoriser les interactions. L'interaction dans les algorithmes distribués peut être considérée comme une coordination entre les différentes entités, ou une coopération pour atteindre un objectif donné ou encore une négociation.

La majorité des travaux proposent des approches basées sur les Systèmes Multi-Agents

(SMA). Un système multi-agent est un système artificiel composé d'une population d'agents autonomes qui coopèrent les uns avec les autres pour atteindre des objectifs communs, chaque agent poursuivant ses objectifs individuels [CLW04]. Ces agents se caractérisent par leur capacité de la mise en oeuvre en parallèle de la méta-heuristique. En outre, parmi les avantages importants de toute méthode multi-agent est de permettre à un agent de résoudre des sous-problèmes locaux de prendre des décisions locales tout en participant à la décision globale. La solution globale est issue suite aux interactions entre les différents agents.

Au cours des dernières années, de bons résultats ont été obtenus en utilisant des approches distribuées basées sur les systèmes multi-agents pour résoudre des problèmes d'ordonnancement comme le job shop flexible. En fait, la prochaine génération de systèmes de fabrication pointe vers un contrôle de plus en plus distribué.

Dans ce contexte, [CLW04] ont proposé un système multi agent basé sur l'algorithme génétique pour résoudre JSP. Dans cette approche, le traitement, la sélection, le croisement et la mutation sont contrôlés d'une manière intelligente. [AZ10] ont proposé un algorithme génétique parallèle à base d'agents pour le problème d'ordonnancement de job shop. Dans cette approche, la population initiale est créée par des agents en utilisant la méthode proposée dans [CLW04]. Ils définissent une politique de migration pour coordonner les échanges de migrants entre les différents agents.

Les auteurs de [EG04] proposent un modèle multi-agents basé sur la recherche tabou pour résoudre le FJSP.Les auteurs dans [AEJG12] proposent une nouvelle approche utilisant des systèmes multi-agents pour résoudre le FJSP. Le modèle proposé combine une approche d'optimisation locale basée sur Tabou (TS) et une approche globale d'optimisation basée sur un algorithme génétique. [WD12] proposent un nouveau système multi-agents d'ordonnancement MASS pour résoudre FJSP. Le système est inspiré par les stratégies de négociation du système immunitaire humain. [EG08] proposent une approche multi-agents basée sur une nouvelle technique locale de diversification pour résoudre FJSP. Parmi les approches évoquées durant ces dernières années, on note les approches distribuées basées sur PSO. [HE13] proposent un modèle multi-agents basée sur l'hybridation de TS et PSO, appelé Flexible Job Shop Multi-Agent Tabou Search Particules Swarm Optimization (FJS MATSPSO) qui est inspiré à partir de [EG08]. Le modèle proposé est composé d'agents responsables des ressources à un processus d'optimisation locale basée sur TS et un agent responsable de l'optimisation globale basée sur PSO.

La nature flexible de système production rend l'adoption de ce type d'organisation distribuée facile. Cette dernière s'occupe de la répartition des calculs sur des entités autonomes

de résolution. Ces métaheuristiques présentées précédemment utilisent le type d'architecture distribuée afin d'améliorer la qualité de la solution trouvée et donner plus de flexibilité. Mais leur limite est de ne pas prendre en considération les incertitudes qui peuvent surgir dans les ateliers de production.

En effet, les systèmes réels de production sont des environnements dynamiques caractérisés par des événements non planifiés imprévisibles. Plusieurs sortes d'aléas peuvent arriver en cours de la production et peuvent être de différentes origines [CCAT14]. Ceci rend les problèmes d'ordonnancement dans ces systèmes de plus en plus complexe et de plus en plus difficile à résoudre.

La prise en compte des incertitudes sur les données du problème a donné lieu à un nouveau champ de recherche appelé ordonnancement robuste ou ordonnancement dynamique. L'environnement étant dynamique et sensible aux changements, la solution produite doit résister aux perturbations de l'environnement tout en maintenant des bonnes performances.

Dans ce qui suit nous présenterons des notions sur les incertitudes, les différents types ainsi que les techniques de leurs modélisations. En deuxième lieu, nous présenterons les méthodes de résolution développées dans la littérature pour aborder ce problème.

1.3 Ordonnancement de FJSP sous incertitudes

1.3.1 Définition de l'ordonnancement sous incertitudes

La littérature scientifique a connu ces dernières années une émergence de la problématique d'ordonnancement avec prise en compte des perturbations. Dans un environnement industriel, la probabilité qu'un ordonnancement prévisionnel soit exécuté comme prévu est faible. En effet, l'environnement d'application de l'ordonnancement est naturellement dynamique et perturbé.

Des paramètres internes, propres au système assurant la réalisation d'un ordonnancement (capacité des ressources, durées opératoires, etc), ainsi que des paramètres externes, correspondants aux informations données par les fournisseurs (délais, volumes de production, etc) souvent supposés connus et constants alors qu'en réalité, ils sont incertains et susceptibles de varier dans le temps de façon plus ou moins prévisible et pouvant constituer une source d'incompatibilité possible du modèle avec le réel. Parmi ces imprévisions on peut citer :

— durées opératoires réelles des tâches pouvant différer de celles estimées.

— indisponibilité d'opérateur ou de matière première.

— indisponibilité des machines en raison d'une panne.

— une variation de la date d'échéance d'une opération, l'arrivée d'un job aléatoire, etc.

Ces aléas modifient et perturbent l'état d'un système de production.
Les données du problème d'ordonnancement sont assorties de valeurs numériques pour présenter l'état du système réel (temps d'exécution, vitesse d'une machine) susceptibles de changer .
Nous définissons, en premier lieu, les différents types d'incertitudes et nous présenterons un état de l'art sur les techniques d'ordonnancement sous incertitudes.

1.3.2 Les types d'incertitudes dans l'atelier de production

Ils existent plusieurs types de changements susceptibles d'intervenir, désignés souvent par les termes d'incertitudes, aléas et parfois de variations [BMZ95].

- Incertitude : Les connaissances sur un système réel sont souvent imparfaites, les valeurs de données d'un problème d'ordonnancement ne sont pas connues à priori, pouvant subir des modifications après le calcul de l'ordonnancement ou pendant son exécution. C'est le cas, par exemple, de la durée opératoire d'une tâche variant durant l'exécution de l'ordonnancement.
- Les aléas : désigne les types particuliers d'incertitudes "événements" qui modifient les données. La probabilité de la survenue d'un aléa peut ainsi être connue ou pas. On peut citer comme exemple d'aléas, le cas d'une machine tombant en panne ou le changement de priorité pour l'exécution d'une tâche donnée.

D'une manière générale nous parlerons d'incertitudes. Par ailleurs, de nombreux critères ont été proposés pour classifier de manière plus fine les différents types d'incertitudes. [MU99] a considéré deux classes d'incertitudes : celles liées aux travaux (ajout ou retrait d'un travail) et celles liées aux ressources (changement et/ou manque de ressources, etc).
Une autre classification a été présentée par [ABPR02] qui ont classifié les incertitudes en fonction de leur niveau de connaissance. Ainsi dans les environnements manufacturiers, trois types d'incertitudes sont identifiés :
-des incertitudes complètement inconnues, qui sont des événements imprévus pour lesquels aucune information n'est disponible à l'avance (par exemple, une grève ou un accident dans le lieu de travail ou une absence d'un ouvrier, etc),
-des suspicions du futur, issues de l'intuition et de l'expérience des décideurs,
-des incertitudes partiellement connues (pannes machines,...) pour lesquelles un modèle d'information est disponible.

[BT06] a classifié les incertitudes en incertitudes stratégiques, tactiques et opérationnelles. Les incertitudes stratégiques sont externes liées aux conditions environnementales et aux changements de technologie, ils affectent principalement la prise de décision à long terme. Les incertitudes tactiques peuvent modifier la prise de décision à moyen terme, les perturbations dans l'information et les flux des matières sont des exemples de ce type. L'absentéisme de l'opérateur et le changement des durées de traitement de l'opération sont des exemples des incertitudes opérationnelles qui affectent principalement la prise de décision à court terme.

1.3.3 Modélisation de l'incertitude

La prise en compte des incertitudes dans le problème d'ordonnancement rend la question de leur modélisation inévitable. Le choix du modèle pour présenter cette information incertaine, imprécise et incomplète et la rendre plus proche du modèle réel est un enjeu très intéressant. Le chercheur est confronté à des choix arbitraires, également valides, de simplification et d'agrégation. L'effort de modélisation implique la recherche d'un compromis entre une représentation proche de la réalité, et une représentation intelligible.
La littérature offre plusieurs modélisations. Dans [BMS05], quatre types de modèles sont distingués : les modèles stochastiques, les modèles flous, les modèles par intervalles et les modèles par scénarios. Nous présentons dans les sections suivantes les quatre modèles.

1.3.3.1 Modèle probabiliste

ou encore modèle stochastique : les données du problème d'ordonnancement sont associées à des variables aléatoires et à des distributions de probabilité en se basant sur des informations issues d'analyses statistiques appliquées à l'historique d'une organisation.

1.3.3.2 Modèle flou

La théorie des sous-ensembles flous se présente comme un outil privilégié pour la modélisation des situations présentants des imprécisions [Zad65].
Ce modèle permet d'exprimer des informations recueillies en langage naturel, ou des valeurs approximatives dues à des difficultés de mesure. Ces variations sont présentées par des fonctions d'appartenance.

1.3.3.3 Modèle par intervalles

Un modèle par intervalles peut être vu comme un cas particulier des modèles précédents puisque chaque paramètre peut prendre sa valeur, selon une densité de probabilité uniforme,

dans un intervalle de valeurs ou encore un ensemble continu de valeurs possibles. Les paramètres associés à ce modèle sont généralement les dates de disponibilité ainsi que les durées opératoires dates de début et de fin des tâches.

1.3.3.4 Modèle par scénarios

Cette modélisation nécessite la construction d'un ensemble de scénarios (appelés aussi instances ou jeux de données) contenant des valeurs numériques. Les paramètres du problème prennent une valeur dans un ensemble discret de valeurs possibles. Chaque ensemble correspond à un scénario, c'est-à-dire à un problème d'ordonnancement déterministe distinct.

1.3.4 Les mesures de performance de l'ordonnancement sous incertitudes

En intégrant les aléas et les incertitudes lors de sa résolution, le problème d'ordonnancement devient de plus en plus complexe et plus difficile à résoudre. Des nouvelles mesures de performance ont été proposées afin de satisfaire cet objectif en intégrant de nouveaux critères en terme de stabilité, robustesse et flexibilité.

1.3.4.1 La robustesse

Un nouveau critère de performance s'est imposé avec l'ordonnancement dynamique à savoir la robustesse.Il existe dans la littérature plusieurs définitions de la robustesse en ordonnancement, en voici quelques-unes :

— " *Un ordonnancement robuste est un ordonnancement capable de satisfaire des exigences de performance a priori dans un environnement incertain* " [LP91].

— " *Un ordonnancement robuste est un ordonnancement qui est insensible à des perturbations imprévues de l'atelier de production, étant donnée une politique de contrôle a priori* " [JLDWS94].

— " *La robustesse est la capacité d'un ordonnancement prédictif à faire face à des événements imprévus*" [DB00].

— " *Un ordonnancement robuste est un ordonnancement capable d'absorber certaine quantité d'événements inattendus sans avoir à être ré-ordonnancé*" [Davenport Beck 00].

— " *Un ordonnancement est dit robuste quand on prévoit sa bonne performance par rapport aux autres face à un certain ensemble de scénarios, et quand on utilise la technique de décalage à droite pour ré-ordonnancer* " [J^{+}01].

— " *Un ordonnancement est robuste si ses performances se dégradent un peu par les perturbations, c'est-à-dire que la performance d'un ordonnancement robuste est insensible aux perturbations* [LAR07].

— " *Un ordonnancement prédictif est dit robuste si la qualité de l'ordonnancement final exécuté (on line) est proche de la qualité du l'ordonnancement prédictif calculé (offline)*" [BVLB09].

Il est difficile de donner une définition type de la robustesse en ordonnancement de production, toutes les définitions citées précédemment sont différentes, mais complémentaires. Nous constatons que les définitions de la robustesse diffèrent de la capacité de faire face à des aléas, de résister à l'aproximation, de gérer le risque, etc.

Cette hétérogénéité implique également que la robustesse soit mesurée de façon différente. Plusieurs mesures génériques sont proposées [BMS05]. Ils diffèrent principalement selon les spécificités du domaine applicatif considéré [HL05].

Dans un contexte d'optimisation, il est important de mesurer la robustesse afin de pouvoir déterminer la solution la plus robuste.

Nous présentons dans le chapitre quatre quelques mesures de robustesse utilisées dans notre travail. De manière générale il s'agit de trouver une solution de bonne performance peu sensible aux incertitudes. Nous retenons la définition proposée par [BMS05], *un ordonnancement est robuste si sa performance est peu sensible à l'incertitude des données et aux aléas.*

1.3.4.2 La stabilité

La stabilité d'un ordonnancement est attribuée à la distance entre l'ordonnancement de référence et l'ordonnancement réalisé pour un scénario donné. La distance peut être mesurée par le nombre d'activités perturbées, la différence entre la date de début de l'activité planifiée et la date de début de l'activité réalisée, etc.

Selon [WSPC93], un ordonnancement est stable s'il présente une petite déviation soit dans le temps, soit dans la séquence entre l'ordonnancement prédictif et celui réalisé. Différentes mesures de la stabilité et de la robustesse ont été définies pour le problème de l'ordonnancement FJSP [J$^+$01], [AHE11].

1.3.5 Classification des méthodes de résolution sous incertitudes

Les approches de résolution proposées dans la littérature pour résoudre le problème FJSP supposent implicitement que l'ordonnancement statique peut être appliqué comme prévu.

Toutefois il est aujourd'hui irréaliste d'ignorer l'aspect dynamique et incertain de l'environnement d'application.

Plusieurs méthodes ont été développées pour prendre en compte l'incertitude lors de la résolution du problème. Dans la section suivante, nous présentons différentes approches d'ordonnancement sous incertitudes.

Ces approches de résolution proposées dans la littérature sont classées selon plusieurs critères.

Plusieurs classifications ont été proposées selon la nature, la façon et l'instant des décisions prises durant les étapes de résolution du problème, à savoir les décisions hors-ligne (par un algorithme statique) et les décisions en-ligne (moyennant un algorithme dynamique).

[MU99] classent les méthodes de résolution sous incertitudes en quatre catégories : l'approche réactive, l'approche prédictive-réactive, l'approche robuste et l'approche à base de connaissances.

La première catégorie est complètement réactive n'utilisant aucun ordonnancement de référence pour construire la nouvelle solution. La deuxième approche se basant sur l'exploitation d'un ordonnancement déterministe de référence lors de la phase réactive. L'approche robuste recouvre quant à elle les approches proactives qui tentent de prendre en compte l'incertain lors de la phase d'ordonnancement hors-ligne uniquement. La dernière catégorie "les approches à base de connaissance" peut être considérée comme une sous-catégorie particulière des approches réactives. En fait, l'objectif de ce type de méthode est de fournir un mécanisme pour la sélection dynamique d'une politique de ré-ordonnancement approprié, parmi un ensemble d'alternatives possibles.

Selon [DB00], il existe seulement deux types d'approches : les approches proactives et les approches réactives.

[La05] a proposé une autre classification et il distingue trois types d'approches : les approches réactives, les approches prédictives-réactives et les approches proactives-réactives. Dans cette classification, l'auteur a intégré les approches proactives dans les approches proactives-réactives.

D'autres classifications ont été élaborées selon la nature des outils utilisés pour modéliser l'incertitude. On peut citer par exemple le travail de [HL05] cinq types d'approches sont distingués : les approches réactives, les approches stochastiques, les approches floues, les approches proactives et les approches basées sur l'analyse de sensibilité. Cette classification se base plus particulièrement sur les problèmes d'ordonnancement de projet sous incertitudes.

Plusieurs types de modèles ne sont pas considérés dans certaines classifications, la classifi-

cation de [DB00] ignore l'approche proactive-réactive, ni l'approche prédictive-réactive celle de [La05] intégre les approches proactives dans les approches proactives-réactives.
De ce constat, [CCAT14] ont proposé une nouvelle classification globale qui couvre les différentes méthodes d'ordonnancement sous incertitudes dans plusieurs types de problèmes (ordonnancement de projet, ordonnancement des ateliers, etc). Elle comporte trois approches, à savoir les approches proactives, les approches réactives et les approches dites hybrides. Cette dernière regroupe les approches prédictives-réactives et les approches proactives-réactives.
Nous structurons notre état d'art sur l'ordonnancement de FJSP sous incertitudes selon cette récente classification de [CCAT14]. Nous distinguons principalement trois types : les approches proactives, les approches réactives, les approches hybrides (prédictive-réactive et proactive-réactives).

1.3.5.1 Les approches proactives

Les approches proactives appelées aussi "approches robustes" se basent sur la prise en compte des incertitudes uniquement lors de la phase d'ordonnancement hors-ligne. Il s'agit de produire un ordonnancement, ou un ensemble d'ordonnancements, relativement insensible aux incertitudes en jouant sur la flexibilité de l'atelier de production. L'ordonnancement est calculé hors ligne par anticipation en tenant compte a priori des connaissances sur les incertitudes probables.
Ce type d'approche est applicable aux incertitudes partiellement connues pouvant être modélisées. Si le degré d'incertitude est très élevé, ou si aucun modèle d'incertitudes n'est a priori disponible, alors une approche réactive sera plus appropriée.
Plusieurs approches proactives ont été proposées pour résoudre le problème FJSP en tenant compte de l'incertitude et des événements imprévus : [Jen03] propose un algorithme génétique pour trouver un ordonnancement robuste avec faible makespan (la date de fin d'exécution de la dernière tâche) pour le problème de job shop. Il a défini également une nouvelle mesure de robustesse. [FF10] développent une méthode multi-objectifs basée sur l'algorithme génétique pour résoudre le problème de job shop flexible tout en tenant compte de l'arrivée dynamique des jobs. [AHE12] proposent un algorithme génétique hybride pour obtenir un ordonnancement prédictif tout en tenant compte du fait que les temps de traitement des opérations soient incertains.
Les pannes des machines dans les ateliers de production constituent la source d'incertitudes la plus étudiée par les chercheurs [HSL13].
Plusieurs travaux ont abordé ce sujet comme [AHE11] qui ont proposé un algorithme géné-

tique hybride pour le problème de job shop flexible. Ils ont proposé aussi trois mesures de robustesse et de stabilité afin de trouver la meilleure qui permet d'obtenir un ordonnancement robuste, fiable et avec min Makespan. [XXC13] proposent un ordonnancement robuste pour le problème FJSP avec la prise en compte des pannes aléatoires des machines. Il s'agit d'une optimisation multi-objectif : min makespan et max robustesse. [DM12] se concentrent sur l'optimisation multi objectif FJSP en considérant les maintenances des machines. Ils proposent une hybridation entre l'algorithme génétique et le recuit simulé. [HSL13] appliquent Novel Clone Immune Algorithm pour résoudre FJSP tout en considérant les pannes des machines et proposent une nouvelle mesure de stabilité reflétant la stabilité de l'allocation des machines pour chaque opération.
Récemment, PSO a été utilisé pour résoudre le problème FJSP sous incertitudes. [PYY13] proposent un algorithme d'optimisation par essaim particulaires quantiques (QPSO) afin de résoudre le FJSP sous incertitude, principalement sur le temps de traitementet de livraison incertain. [SM16] proposent une méthode multi-objectifs basée sur QPSO pour générer un ordonnancement prédictif pouvant simultanément optimiser le makespan et la mesure de robustesse. [SLW+15]proposent un Algorithme Évolutionnaire Hybride avec le Réseau Bayésien (BN) pour résoudre le FJSP sous incertitudes temporelles. L'approche combine l'algorithme PSO et GA comme étant deux agorithmes évolutionnaires typiques.
Dans notre travail, nous limitons l'incertitude à des pannes de machines c'est-à-dire l'indisponibilité temporaire d'une machine.

1.3.5.2 Les approches réactives

A l'inverse des approches proactives, les approches réactives prennent en compte l'incertain lors de la phase d'ordonnancement dynamique (en-ligne). C'est-à-dire sans anticipation des incertitudes, il s'agit plutôt de réagir en temps réel, de façon opportune, lorsque des aléas surviennent.
Ces approches qualifiées aussi de "dynamiques" car les décisions sont prises dynamiquement et en temps réel au fur et à mesure de l'apparition des aléas, en privilégiant la rapidité des décisions sur leur qualité.
Le nouveau ordonnancement est élaboré sur la base de l'état courant du système d'activités (état des ressources et des tâches) et exploitera éventuellement des informations relatives à l'aléa considéré (durée d'une panne, estimation d'un retard). Un ordonnancement de référence, déterminé hors-ligne et de nature déterministe, est aussi parfois utilisé pour trouver la nouvelle solution.

Parmi les méthodes réactives fréquemment utilisées, nous citerons celle qui consiste à dispatcher les opérations aléatoirement vers les machines disponibles selon les règles à base de priorité connue sous le terme "priority dispatching rules".
Une autre alternative réactive est d'utiliser les systèmes multi agent. [LAR07] présentent une architecture multi-agents complète pour une planification dynamique des ateliers de type job shop tout en tenant compte de plusieurs types d'incertitudes. Néanmoins la distribution reste virtuelle car l'approche multi agent est intégrée dans la même machine.
[MCP10] proposent une nouvelle approche basée sur le réseau de neurones ou encore Neural Network (NN) afin de sélectionner en temps réel la règle de priorité la plus appropriée et la plus convenable à appliquer. Dans [ZPST12], les auteurs ont proposé une approche réactive pour résoudre le problème FJSP en utilisant des champs potentiels numériques. Les Champs potentiels sont utilisés pour allouer dynamiquement les jobs à des ressources sans prédiction. Le temps d'élaboration des décisions d'ordonnancement, nécessaires à la prise en compte d'un aléa, doit être suffisamment court relativement à la dynamique du système d'activités considéré.

1.3.5.3 Les approches hybrides

Plus récemment, des approches hybrides ont été developpées pour résoudre le problème de FJSP avec incertitude. Cette famille combine les deux approches précédentes et conduit soit à des approches prédictives-réactives, soit à des approches proactives-réactives [CCAT14].

1.3.5.3.a Les approches prédictives-réactives

Ces approches sont constituées de deux phases : la première phase consiste à construire un ordonnancement déterministe hors-ligne ne prenant en considération que les événements prévisibles. Par exemple, les jobs qui doivent être prévus sont tous disponibles dès le départ, les processing times sont déterminés, les machines et d'autres ressources sont disponibles durant l'horizon de l'ordonnancement. Durant la deuxième phase, cet ordonnancement est utilisé, en ligne, et adapté alors en temps réel, pour tenir compte des perturbations [COP04]. Comme exemple on peut citer le travail de [GST+15] qui a proposé quatre heuristiques pour l'ordonnancement du FJSP tout en considérant les insertions des nouveaux jobs. Les objectifs étant de minimiser le temps maximal d'achèvement (makespan), de minimiser le temps de retard, de minimiser la charge de la machine la plus surchargée ainsi que la charge totale des machines (Totalworkload). Ils utilisent les mêmes heuristiques durant la phase d'ordonnancement et la phase de ré-ordonnancement en tenant compte toujours des nouvelles insertions.

Le ré-ordonnancement consiste à modifier pendant une période de temps, la plus réduite possible, l'ordonnancement prédictif de référence perturbé, de sorte que cet ordonnancement soit réutilisable rapidement.

1.3.5.3.b Les approches proactives-réactives

L'approche proactive-réactive est aussi constituée de deux phases, une première hors ligne pour déterminer un ordonnancement robuste tenant compte de l'aspect incertain de certaines données et la deuxième phase consiste à adapter cet ordonnancement en-ligne en fonction de l'état du système au moment de son exécution, grâce à un algorithme dynamique.
Comme nous l'avons mentionné dans les sections précédentes, une approche d'ordonnancement proactive exploite une connaissance a priori des perturbations lors de la construction de l'ordonnancement prédictif initial. Toutefois, il est irréaliste de vouloir tenir compte proactivement de tous les événements aléatoires pouvant surgir au niveau de l'atelier. De ce fait, les approches proactives-réactives rajoutent une phase réactive afin de réagir aux événements incertains non pris en compte dans la phase proactive.
La différence entre les approches prédictives-réactives et les approches proactives-réactives est principalement due au fait que, dans les approches proactives-réactives, aucun réordonnancement n'est fait en-ligne ; une solution d'ordonnancement évaluée est choisie parmi un ensemble de solutions [CCAT14].
La plupart des méthodes de résolution des approches proactives-réactives permettent de construire un ensemble d'ordonnancements statiques de telle sorte qu'il soit facile de passer de l'un à l'autre, en cas de perturbation.
Les auteurs dans [WCCC09] ont proposé une approche qui construit un ordonnancement robuste minimisant la déviation entre l'ordonnancement initial de base (baseline) et l'ordonnancement prévu. Dans la phase réactive, lorsque des événements imprévus apparaissent, l'ordonnancement est rapidement révisé en temps réel en fonction des informations de fabrication. [EKBA17] proposent les "Dual-AntsColony"(DAC), une nouvelle approche hybride d'optimisation des colonies de fourmis (ACO) .
L'approche combine la recherche locale avec un ensemble de règles d'expédition (dispatching rules) pour résoudre les FJSP avec indisponibilité des machines due essentiellement aux activités de maintenance préventive (MP). L'objectif étant de minimiser le temps d'achèvement (makespan).
Comme c'est décrit précédemment, plusieurs approches ont été développées afin de résoudre le problème de FJSP sous incertitudes. Les différentes méthodes se distinguent par la méta-

heuristique utilisée, le codage de la solution, le type d'incertitude considéré, l'objectif à optimiser, la façon de réagir (hors-ligne/en-ligne), les mesures de robustesse et stabilité utilisées, etc.

Le choix de la méthode est toujours un grand défi à surmonter. La comparaison entre les méthodes d'ordonnancement sous incertitudes est difficile vue cette grande diversité.

L'étude comparative dans le cas statique ayant montré que l'algorithme génétique ainsi que l'algorithme d'optimisation par essaims particulaires sont les méta-heuristiques les plus performantes pour améliorer la qualité de la solution.

Selon nos connaissances, il n'existe à ce jour aucun travail de recherche ayant proposé une approche basée sur l'optimisation par essaim particulaire pour aborder le problème FJSP en tenant compte de ce type d'incertitude (panne des machines).

Nous proposons ainsi une approche basée sur le PSO afin de comparer entre deux approches prédictives (PSO ou GA) pour trouver la plus appropriée pour résoudre FJSP sous ce type d'incertitudes.

L'approche proposée étant prédictive avec deux objectifs : augmenter la stabilité et la robustesse de la solution tout en ayant un temps de makespan faible.

Le développement des systèmes capables de résister aux événements imprévisibles tout en maintenant des performances élevées est primordial. Le ré-ordonnancement est l'une des techniques de réparation souvent utilisée pour palier et corriger l'ordonnancement préventif en tenant compte de l'état du système. La section suivante sera consacrée pour un état d'art sur les différents types de ré-ordonnancement.

1.3.6 Les techniques de Ré-ordonnancement

Plusieurs méthodes de ré-ordonnancement ont été proposées pour le problème de job shop classique [KVR15]. Dans la section suivante nous décrivons les plus utilisées. Vieira et al. proposent une étude sur la plupart des méthodes de ré-ordonnancement dans les systèmes de production [VHL03].

Les techniques de ré-ordonnancement peuvent être classées en trois groupes :
ré-ordonnancement à droite, ré-ordonnancement total et ré-ordonnancement partiel [VHL03].

1.3.6.1 Méthode de décalage à droite

En anglais, elle s'appelle "Right Shift Rescheduling" (RSR). Cette règle, souvent utilisée dans le cas d'une panne de ressources. Il s'agit d'accommoder une panne par le déplacement global à droite de l'ordonnancement après réparation de la machine en question. En

fait, cette méthode consiste à retarder l'exécution des tâches directement et indirectement affectées par une panne en maintenant la même séquence de départ (l'ordre de passage des opérations sur les machines).
Les opérations directement affectées sont celles réalisées par la machine en question (en panne) et les opérations indirectement affectées sont celles liées par les contraintes de précédences. Bien que cette méthode permet de maintenir la stabilité de l'atelier, elle peut donner en contre partie des mauvaises performances [SOS93].
Cette méthode est appliquée soit d'une manière continue dans ce cas on parle du ré-ordonnancement continu qui consiste à calculer un ordonnancement à chaque fois qu'un nouvel événement surgit. Ceci peut mener toutefois à un temps de calcul prohibitif qui peut par conséquent retarder la production. En outre la solution est instable dans le temps, phénomène peu souhaitable en pratique.
A l'inverse, le ré-ordonnancement périodique appliqué à des intervalles réguliers requiert moins de calcul que le ré-ordonnancement continu. Toutefois, ce type de ré-ordonnancement donne de performances moins bonnes, surtout à la survenue d'une ou plusieurs perturbations importantes entre deux instants de ré-ordonnancement.
Si la panne interrompt le traitement d'une opération, alors l'opération interrompue est décalée à droite par la quantité de temps d'arrêt. Toutes les autres opérations sont décalées à droite par la quantité de temps d'arrêt [AS97].

1.3.6.2 Ré-ordonnancement total

c'est comme une méthode d'ordonnancement classique mais cette fois il s'agit d'ordonnancer un sous ensemble d'opérations. Cette technique a été proposée par [Sno01]. Elle consiste à décomposer le problème en sous-problème résolus en séquences. Elle est principalement proposée pour résoudre le problème de type job shop JSP avec l'arrivée des nouveaux jobs comme étant source d'incertitude. L'objectif considéré est la minimisation du retard moyen. Pour réduire la complexité du temps de calcul, les auteurs ont décomposé le problème en n sous-problèmes, qu'ils résolvent en séquences. Chaque sous-problème est résolu par un algorithme génétique.

1.3.6.3 Ré-ordonnancement Partiel

Parmi les techniques de ré-ordonnancement partiel existantes, on peut citer la méthode "Affected Operations Rescheduling" (AOR) proposée par [AS97]. Cette méthode est basée sur l'algorithme " binary branching algorithm", les opérations et les machines sont représen-

tées par un arbre binaire. Le principe fondamental de la technique AOR consiste à accommoder n'importe quelle incertitude en poussant le début d'opérations par quelques unités de temps en avant (en les retardant) par le temps minimum exigé. [SR03] proposent une méthode se basant sur AOR appelé "Modified Affected Operation Rescheduling" (mAOR). La méthode développée utilise le même concept principal que l'AOR tout en prenant compte des différents types d'incertitudes autres que l'indisponibilité de la machine telles que la variation du temps de traitement, l'arrivée d'un job inattendu, l'arrivée d'un job urgent.
[DJ12] proposent des heuristiques de ré-ordonnancement pour le job shop en considèrant la minimisation du retard du job comme un objectif principal et non seulement le makespan et / ou la stabilité comme les autres approches existantes.
Les méthodes précédemment décrites abordent le problème de ré-ordonnancement du job shop classique.
Quelques travaux seulement ont abordé ce problème pour le job shop flexible FJSP. [UG09] ont proposé une représentation adaptative pour sélectionner la politique de routage pour faire face aux échecs. [SSH13] ont proposé une méthode de ré-ordonnancement en temps réel pour gérer la flexibilité de routage. Ils utilisent une optimisation des colonies de fourmis, des algorithmes génétiques, un recuit simulé et la recherche taboue pour résoudre le problème de sélection de routage en temps réel afin de réduire la congestion dans le système.
Toutes ces différentes méthodes de ré-ordonnancement présentées varient les unes des autres selon le type d'incertitude à résoudre, le type de procédure de réparation totale ou partielle, la méthode heuristique utilisée pour l'optimisation, la fonction objective minimisant différents paramètres tels que makespan, stabilité, tardiveté moyenne ou optimisation multi-objectif combinant différentes métriques.
La caractéristique principale d'un processus de ré-ordonnancement est de réagir rapidement à des événements inattendus. Cependant, les techniques existantes n'accordent aucune importance au temps nécessaire pour calculer le nouvel ordonnancement réparé.
De ce constat, nous proposons une heuristique de ré-ordonnancement partiel qui minimise l'effet d'une panne machine sur le temps de fabrication tout en s'intéressant aux temps de calcul de l'approche.
Le problème de ré-ordonnancement peut être considéré comme un problème d'ordonnancement contraint, dont l'objectif est de minimiser l'écart et la déviation. Ce terme fait référence à la différence entre les horaires (heures de début et de fin) et dans les séquences d'opérations entre l'ordonnancement initial prédictif et l'ordonnancement réalisé. La différence d'horaires se réfère à l'écart de temps de départ tandis que la différence de séquence fait

référence à l'écart de séquence. Une variété de mesures de performances guide la méthode de ré-ordonnancement. Les plus utilisées sont des mesures de l'efficacité (par exemple le makespan) et des mesures de stabilité (c'est-à-dire la déviation par rapport à l'ordonnancement initial). Plus de détail est donné dans le dernier chapitre.

1.4 Conclusion

Dans ce chapitre nous avons présenté les problèmes d'ordonnancement, leurs principales caractéristiques, les types d'ateliers de production, ainsi que les différentes méthodes, exactes et approchées, pouvant être utilisées pour résoudre ce type de problème d'optimisation.
Nous avons présenté aussi un état d'art sur les méthodes développées pour résoudre le problème de type job shop flexible FJSP sans et avec considération des incertitudes.
Le chapitre suivant sera consacré à la présentation de l'algorithme d'optimisation par essaims particulaire développé ainsi que les architectures multi agent proposées.

Chapitre 2

PSO et Multi agent PSO pour résoudre FJSP

2.1 Introduction

L'utilisation des méthodes exactes pour résoudre le problème d'optimisation, comme le problème d'ordonnancement de job shop flexible, requiert un ensemble de calculs, dont le nombre évolue de façon exponentielle avec la taille du problème considéré. Il s'avère donc préférable d'utiliser les méthodes approchées. De plus, le caractère dynamique et incertain de l'environnement de production rend primordial la prise en compte de cet aspect lors de la résolution du problème. Nous nous intéressons tout d'abord à la résolution de FJSP statique déterministe en ignorant les incertitudes. Ceci a pour but de valider les performances de l'algorithme avant d'entamer l'optimisation sous incertitudes.
Nous proposons une approche basée sur l'optimisation par essaims particulaires. Ensuite, nous discuterons l'utilisation des systèmes multi agents pour la distribution de l'algorithme et nous présenterons deux architectures multi-agent, basées sur l'algorithme OEP, nommées MAPSO1 et MAPSO2.
Dans ce deuxième chapitre, nous introduisons le problème d'ordonnancement de type Job-Shop Flexible (FJSP en anglais) ainsi que leur formulation, à savoir l'algorithme d'optimisation par essaim particulaire, le codage de la solution, le paramétrage ainsi que les méthodes d'initialisations utilisées. Une description détaillée de deux architectures multi agent proposées sera donnée.

2.2 La modélisation du problème FJSP

Le problème d'ordonnancement du job shop flexible est parmi les problèmes d'ordonnancement les plus difficiles à résoudre. Leur résolution de manière optimale s'avère très difficile

à cause de leur caractère fortement combinatoire et le grand nombre d'entités à gérer dans l'atelier.

C'est une extension du job shop classique. Il s'agit d'ordonnancer n jobs sur m machines. Chaque job étant constitué de plusieurs opérations.

A l'inverse du job shop classique où chaque opération est excutée par une machine spécifique, le job shop flexible offre plusieurs machines identiques, alternatives ou redondantes capables de réaliser les mêmes opérations. Dans ce cas, une opération donnée peut être réalisée par une ou plusieurs ressources, la durée de traitement dépendant de la ressource utilisée.

Il existe deux types du problème de job shop flexible : FJSP avec flexibilité totale ou Total FJSP en anglais (TFJSP) et FJSP avec flexibilité partielle ou encore Partial FJSP dans la terminologie anglaise (PFJSP).

Le Total FJSP (TFJSP) implique que chaque Job de l'ensemble peut être traité par n'importe quelle machine de l'ensemble de machines et le Partial FJSP (PFJSP) où un job donné ne peut être traité que par un sous-ensemble spécifique de l'ensemble des machines. Pour modéliser le problème de job shop flexible, nous introduisons dans les sections suivantes les paramètres, les variables et les contraintes.

2.2.1 Les paramètres

Les paramètres utilisés pour modéliser le problème d'ordonnacement FJSP sont comme suit :

- P ensemble de jobs, P= 1, 2,.. n.
- M ensemble des machines M= $M_1, M_2, ..., M_r$.
  I_j ensemble des opérations du job j, I_j= 1, 2,...,$|I_j|$, j$\in P$.
- O_{ij} est l'opération i du job j. L'opération i est l'opération numéro i du job j.
- q_i le nombre d'opérations du job i.
- M_{ij} est l'ensemble de machines de l'opération O_{ij}.
- p_{ijr} est le temps de traitement de l'opération O_{ij} sur la machine M_r.

2.2.2 Les Variables

Les variables utilisées pour modéliser le problème d'ordonnancement FJSP sont comme suit :

— t_{ij} le temps d'achèvement de l'opération O_{ij}(Completion time) ($i \in \mathrm{I}_j$), $t_{ij} \in \mathrm{N}$.
μ_{ijr} variable binaire égale à 1 si l'opération O_{ij} est traitée par la machine M_r ; 0, sinon.

— b_{ijkl} variable binaire qui est égale à 1 si l'opération O_{ij} est traitée avant l'opération O_{kl} ; 0, sinon.

— w_{ijr} temps d'attente de l'opération O_{ij} dans la file d'attente de la machine M_r.

— wv_{ijklr} variable binaire égale à 1 si l'opération O_{ij} attend l'opération O_{kl}dans la file d'attente de la machine M_r ; 0, sinon.

2.2.3 Les contraintes

Les contraintes sont les règles qui limitent l'affectation des opérations. Dans notre cas :

— **Les Contraintes disjonctives :** une machine ne peut traiter qu'une seule opération à la fois.

$$t_{ij} + p_{klr} * \mu_{klr} + BM * b_{ijkl} \leq t_{kl} + BM, \forall i, k \in I, \forall j, l \in P, \forall r \in M_r \tag{2.1}$$

avec :

-i représente l'indice de l'opération i du job j, O_{ij}.

-k représente l'indice de l'opération k of job l, O_{kl}.

-BM est un grand nombre.

$$b_{ijkl} + b_{klij} \leq 1, \forall i \in I_j, \forall k \in I_l, \forall j, l \in P \tag{2.2}$$

$$\sum_{r \in M_r} \mu_{ijr} = 1, \forall i \in I_j, \forall j \in P \tag{2.3}$$

— **Les Contraintes de précédence** : ou encore contrainte de succession entre les opérations du même job. Ces contraintes assurent la séquence de production du job.
Le temps d'achèvement de l'opération considère le temps d'achèvement de l'opération précédente du même job , le temps d'attente et le temps de transport si les deux opérations ne sont pas traitées par la même machine [TPB+13]. Dans notre travail, le temps de transport est négligable.
La contrainte est modélisée comme suit :

$$t_{(i+1)j} \geq t_{ij} + p_{(i+1)jr2} + w_{(i+1)jr2} \forall i \in I_j, \forall j \in P, \forall r_2 \in M_r \tag{2.4}$$

avec :

- $p_{(i+1)jr2}$ représente le temps de traitement de l'opération $O_{(i+1)j}$ par la machine M_{r2}

qui doit être executée après l'opération O_{ij} du job j.

- r_2 est la machine qui va traiter l'operation $O_{(i+1)j}$.

— Les jobs sont indépendants et sans aucune priorité.

— Tous les jobs sont disponibles à l'instant zéro.

— Pas de préemption : une opération entamée sur une machine ne peut pas être interrompue.

2.2.4 La Fonction objective

La résolution du problème de FJSP présente trois difficultés principales :

— La première est relative à l'assignation ou l'affectation de chaque opération O_{ij} de chaque job à une machine M_k.

— La seconde correspond au calcul des temps de début td_{ij} et des temps de fin t_{ij} de l'opération O_{ij}.

— La troisième est l'optimisation d'un critère bien déterminé tout en satisfaisant des contraintes.

Le critère à optimiser est souvent modélisé par une fonction objective de minimisation ou maximisation. Dans ce travail, notre premier objectif étant la minimisation du makespan (C_{max}), qui est la date de fin d'exécution de la dernière tâche. Il s'agit de trouver l'ordonnancement qui a le plus petit temps total d'achèvement. La fonction objective est la suivante :

$$C_{max} = max t_{ij} \tag{2.5}$$

avec t_{ij} est le temps d'achèvement de l'opération O_{ij}

2.3 Optimisation par Essaim Particulaire PSO

2.3.1 Description de la méthode

Comme nous l'avons mentionné, l'algorithme d'Optimisation par Essaim Particulaire (OEP) ou Particle Swarm Optimization (PSO), s'inspire du comportement social des animaux évoluant en essaim, tels que les nuées d'oiseaux et les bancs de poissons. C'est un algorithme itératif à base de population des particules.

L'ensemble des particules est nommé "essaim" ou "swarm" selon la terminologie anglaise. Chaque particule représente une solution potentielle au problème d'optimisation. Elle est dotée par :

- Un vecteur de position.

- Une vitesse qui permet à la particule de se déplacer.

- Un voisinage, c'est-à-dire un ensemble de particules (Neighbours) interagissant directement sur la particule, en particulier celle qui a le meilleur critère.

Le principe de l'OEP est de déplacer l'ensemble de particules pour trouver la solution optimale. L'évolution d'une particule à une autre est fondée sur la communication et l'interaction entre elles sans éliminer aucune solution quitte à dégrader la performance.

Le comportement collectif dans l'algorithme d'OEP se manifeste lors du mouvement et le déplacement des particules. En effet, à chaque itération la particule ajuste et modifie sa position en fonction de :

— sa position et sa vitesse actuelle,

— sa meilleure ancienne position,

— la meilleure position de son voisinage [CK02].

Ceci est formulé analytiquement par les relations suivantes :

$$v_i(t+1) = wv_i(t) + c_1[x_i^*(t) - x_i(t)] + c_2[x_g^*(t) - x_i(t)] \tag{2.6}$$

avec :

-$v_i(t+1)$ et $v_i(t)$ étant les vitesses de la particule aux itérations t et $t+1$

-x_i est la position courante de la particule i à l'itération t

-x_i^* est la meilleure ancienne position de la particule i.

-x_g^* est la meilleure position du voisinage de la particule i à l'itération t

-w facteur d'inertie fixé

-c_1, c_2 représentent des coefficients de confiance. Ce sont des valeurs aléatoires entre [0,1].

La mise à jour de la position de la particule se fait à travers l'équation suivante en se basant sur la vitesse que l'on vient de calculer.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.7}$$

2.3.2 Codage de la solution

Un aspect important de l'algorithme d'optimisation par essaim particulaire est la façon dont sont codées toutes les solutions. Une représentation appropriée des particules est d'une grande importance afin de réussir l'application du PSO pour résoudre le problème FJSP.

Les problèmes FJSP comportent deux phases de décision :

- Une phase d'affectation des opérations aux machines adéquates.

- Une phase de séquencement d'opérations (ordre d'exécution).

Par conséquent, une particule doit contenir ces deux informations.

Dans ce travail, nous utilisons le même codage que celui de [JCT07]. La position d'une particule ainsi que sa vitesse se composent en deux parties vectorielles : $Process[\ ]$ et $Machine[\ ]$.

- Un vecteur d'affectation $Machine[\ ]$ représente la machine sélectionnée correspondante pour chaque opération. Chaque composante dans le vecteur $Machine[\ ]$, est un nombre naturel m qui désigne le nombre d'une machine sélectionnée pour traiter l'opération correspondante dans $Process[\ ]$.

- Un vecteur de séquence d'opération $Process[\ ]$ qui indique la séquence d'opérations sur chaque machine. Chaque composante dans le vecteur $Process[\ ]$ désigne une opération du job j. Le nombre de répétitions du nombre j indique le nombre d'opérations du job j.

La taille de deux vecteurs est identique soit le nombre total d'opérations.

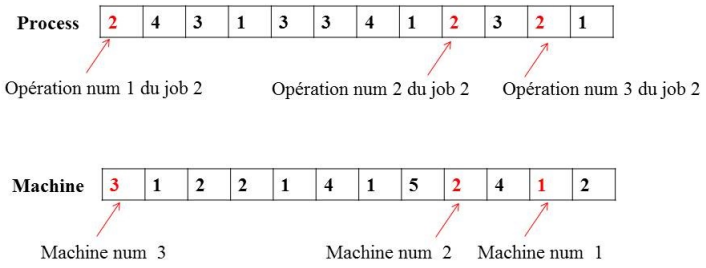

FIGURE 2.1: Codage de la particule

La figure 2.1 illustre un exemple d'une particule pour le problème job shop flexible composé de 4 jobs et 5 machines noté FJSP 4*5.

Le nombre total d'opérations est 12, alors la particule est codée sous forme de deux vecteurs de taille 12. Chaque nombre j dans le vecteur $Process[\ ]$ désigne une opération du job j et les mêmes chiffres du job dans des positions différentes du vecteur représentent l'ordre des opérations du job. Le nombre d'occurrence de chaque numéro correspond au nombre total d'opérations de ce job. Dans cet exemple job 1 a 3 opérations, Job 2 a 3 opérations, job 3 a 4 opérations et job 4 a seulement 2 opérations.

Ces particules se déplacent dans l'espace de recherche, afin de trouver l'optimum global. La section suivante décrit le processus de mouvement.

2.3.3 Configuration des paramètres

Dans l'algorithme d'OEP, les particules se déplacent à l'intérieur d'un espace de recherche. Pendant le déplacement, chaque particule ajuste sa position selon sa propre expérience et selon l'expérience des particules voisines, se servant de sa meilleure position produite et de celle de ses voisines. Le mouvement d'une particule à la recherche de l'optimum est donné par l'équation (2.6) et l'équation (2.7) données précédemment.

L'algorithme OEP comprend certains paramètres qui influencent la performance de l'algorithme. Malgré les efforts récents de recherche, la sélection des paramètres de l'algorithme reste dans une large mesure empirique. Plusieurs choix typiques des paramètres de l'algorithme sont rapportés dans [Tre03] ; [CK02]. [Tre03] proposent des directives de sélection de paramètres pour garantir la convergence optimale. Il existe plusieurs types d'ajustements de coefficients.

La combinaison des paramètres w, c_1 et c_2 permet de régler l'équilibre entre les phases de diversification et d'intensification du processus de recherche. [CK02] ont démontré qu'une bonne convergence peut être obtenue en rendant dépendants ces paramètres. L'utilisation d'un facteur de construction permet de prévenir la divergence de l'essaim. Après avoir essayé plusieurs configurations des coefficients de confiance afin de trouver la meilleure, nous optons pour l'utilisation du facteur de construction proposé par [Ken99] et présenté dans l'équation ci-dessous :

$$k = 1 - \frac{1}{\alpha} + \frac{\sqrt{|\alpha^2 - 4\alpha|}}{2} \tag{2.8}$$

avec $\alpha = c_1 + c_2 > 4$; $c_1 = 2$;$c_2 = 2,1$

Ainsi l'équation de vitesse devient comme suit :

$$v_i(t+1) = k(v_i(t) + c_1[x_i^*(t) - x_i(t)] + c_2[x_g^*(t) - x_i(t)]) \tag{2.9}$$

L'algorithme d'optimisation par essaim particulaire étant défini initialement dans le domaine continu, il s'avère évidemment nécessaire d'effectuer une conversion du domaine continu vers le domaine discret pour la résolution des problèmes d'ordonnancement.

Nous utilisons la même procédure que celle proposée par [JCT07]. Lors de la mise à jour d'une position, pour chaque composant du vecteur $Machine[\ ]$: si une composante x passe à une fraction décimale $x^{'}$, après avoir été calculée par l'équation (2.7), alors x prend la partie entière par excès $x = \lceil x^{'} \rceil$

En outre, m étant le nombre total de machines constant, chaque composant du vecteur $Machine[\ ]$ doit rester dans le même intervalle [1,m]. Si après la mise à jour une composante

sort de cet intervalle, on remplace la valeur par une valeur aléatoire dans cet intervalle.
Concernant le vecteur *Process*[ ], après avoir appliquer l'équation (2.7), l'algorithme ordonne d'abord l'ensemble des composantes dans l'ordre croissant.
Puis, selon l'ordre, il construit le nouveau vecteur selon l'ordre en correspondance avec la valeur initiale dans l'ancien vecteur *Process*[ ]. La figure 2.2 illustre un exemple bien détaillé de la procédure.

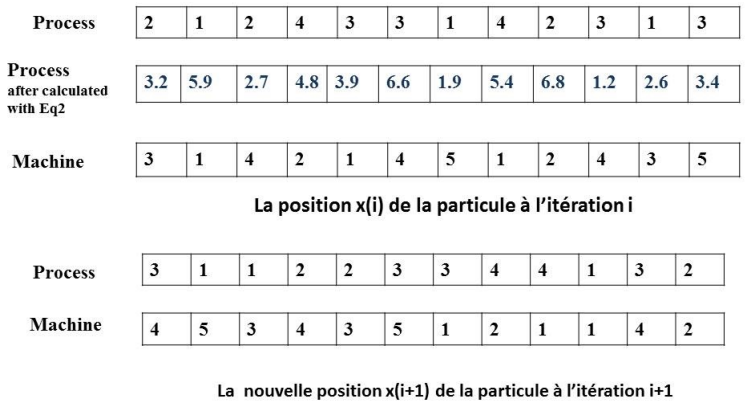

FIGURE 2.2: Exemple de mise à jour de la position d'une particule

Cette procédure nous permettra de construire une solution réalisable qui vérifie les contraintes de précédence. Nous remarquons que le vecteur machine a été mis à jour aussi selon l'ordre de vecteur Process [ ]. Ceci s'avère intéressant et utile s'il s'agit surtout d'un problème d'ordonnancement de type P-FJSP. Dans ce cas la particule ne passe pas par un processus de vérification de faisabilité de la solution car l'opération est bien attribuée à une machine parmi l'ensemble de machines associées à cette opération. S'il s'agit d'un problème de type T-FJSP, on peut diversifier les solutions en changeant les machines même aléatoirement grâce à la flexibilité des machines qui peuvent exécuter n'importe quelle opération.

FIGURE 2.3: Nouvelle position en changeant le vecteur *Machine*[ ]

La figure 2.3 illustre le même exemple mais en changeant l'affectation des machines ce qui nous amène à deux solutions différentes.
Pour diversifier et mieux explorer autant que possible l'espace de recherche, nous devons

vérifier qu'il existe une distance entre les particules.
Soit x_1 et x_2 deux positions. La distance entre ces deux positions est donnée par l'équation suivante :

$$d(x_1, x_2) = \|x_1 - x_2\| \tag{2.10}$$

Si la distance entre la position actuelle et la nouvelle position égale à 0, cela signifie que la particule ne bouge pas du tout ou qu'elle reste exactement à la même position.
Comme nous avons vu précédemment, la nouvelle position de la particule dépend de la position courante, la meilleure position visitée est la meilleure position de son espace de voisinage. La section suivante décrit les topologies de voisinage.

2.3.4 Topologie de voisinage

Une autre question importante qui mérite l'attention, est la topologie de la communication utilisée pour diffuser l'information à l'intérieur de l'essaim.
Le voisinage d'une particule est un ensemble de particules voisines (Neighbours) qui interagissent directement avec la particule, en particulier celle qui a le meilleur critère. La taille de voisinage est variable et peut être modifiée.
L'espace de voisinage à un rôle important dans l'intensification qui consiste à explorer en détail une région de l'espace de recherche jugée prometteuse.
Un élargissement temporaire du voisinage de la solution courante en augmentant le nombre de particules voisines peut aider l'algorithme à visiter un autre ensemble de solutions qui peut mener à la convergence.
De nombreuses topologies ont été proposées et améliorées de topologies statiques en topologies dynamiques.
Les deux méthodes les plus couramment utilisées sont connues sous le nom "gbest" et "lbest". Elle font partie de la famille de topologie statique.

- **Topologie de voisinage statique :**

  -Topologie en étoile ou Gbest : Dans la topologie en étoile "the star topology", les particules peuvent partager des informations à travers une structure entièrement connectée. Cette topologie utilise un mécanisme de voisinage global où la trajectoire de la recherche de chaque particule est influencée par la meilleure position trouvée par n'importe quel membre de la population entière [Ken99].

  -Topologie en anneau : basée sur un voisinage local, appelée aussi "the ring topology" ou "lbest" [MK03]. Dans cette approche, les particules ne partagent que des informa-

tions avec leurs n voisins directs définis (le nombre de solutions voisines n est variable). Quelques autres topologies ont été proposées pour équilibrer le comportement extrême des approches de gbest et de lbest, telles que la topologie de communication dynamique Dynamic Ring et Dynamic Clan.

- **Topologie de voisinage dynamique :**

  -Dynamic Ring[WX08] : A chaque génération, un nouveau espace de voisinage est fourni à la particule. Dans cette topologie, les particules sont disposées dans un anneau régulier, comme dans le modèle classique en anneau.

  -Dynamic Clan [BFCFdM09] : L'essaim est réparti en groupes ou clans de particules et chaque clan admet un leader. Chaque clan contient un nombre fixe de particules qui sont entièrement connectées. Dans cette topologie les auteurs ajoutent la possibilité de migration des particules d'un clan à un autre. A chaque itération, la particule qui a la meilleure position est choisie comme un leader.

Nous avons testé ces topologies citées ci-dessus et nous avons trouvé que les approches sont proches en termes de qualité de la solution trouvée. L'algorithme converge presque vers la même solution dans toutes les topologies, cependant le temps d'exécution de l'algorithme devient plus court en se servant de la topologie étoile Gbest. Ainsi dans l'équation (2.9) la solution voisine X_g^* est remplacée par la particule globale gbest.

2.3.5 La phase de création de la population initiale

Plusieurs recherches ont montré que la qualité de la population initiale affecte d'une certaine manière la qualité de la solution trouvée ou la vitesse de convergence de l'algorithme. Ainsi, une étape trés importante lors de la résolution de notre problème consiste à produire la meilleure qualité initiale de la population.

Le processus d'initialisation se décompose en deux phases : une phase d'affectation et une phase de séquencement. La première consiste à choisir la machine qui va traiter l'opération et la deuxième phase consiste à déterminer la séquence des opérations sur chaque machine. L'algorithme d'optimisation par essaim particulaires, originalement conçu, est basé sur un essaim des particules créé de façon aléatoire et se répètant ensuite en cycle d'évolution.

Afin de diversifier l'espace de recherche, nous avons proposé d'utiliser trois approches lors de la phase de création de la population initiale avec un pourcentage différent : une méthode aléatoire [LPX+10], l'approche de localisation proposée par [KHB02] et une nouvelle approche modifiée dénotée "MMkacem" [NJA+13].

- La méthode aléatoire :

  consiste à choisir aléatoirement un numéro du job j et puis sélectionner une machine aléatoire parmi l'ensemble de machines candidates, a noter M_i, pour traiter l'opération courante i du job j.

- L'approche de localisation :

  proposée par [KHB02]. Cette méthode prend en considération le temps de traitement ainsi que la charge du travail de la machine.

  En fait, pour chaque opération de chaque job, la machine ayant le plus petit temps de traitement sera sélectionnée. Ensuite une mise à jour de la charge de travail sera effectuée sur la machine concernée. C'est-à-dire ajouter le temps de traitement de l'opération choisie au temps de traitement des autres opérations sur la même machine.

  La figure 2.4 montre un exemple illustratif de cette méthode. Les valeurs en gras dans

	M1	M2	M3
O11	7	6	2
O12	8	2	1
O21	9	5	4
O22	2	5	1
O23	4	6	8
O31	3	2	1

	M1	M2	M3
O11	7	6	**2**
O12	8	2	**3**
O21	9	5	**6**
O22	2	5	**3**
O23	4	6	**10**
O31	3	2	**3**

	M1	M2	M3
O11	7	6	2
O12	8	2	3
O21	9	**7**	6
O22	2	**7**	3
O23	4	**8**	10
O31	3	**4**	3

.....

FIGURE 2.4: Approche de localisation

une colonne implique la mise à jour de charge de travail de cette machine.

On remarque que l'ordre des opérations dans la solution construite dépend de leur emplacement dans le tableau. [KHB02] ont proposé de permuter aléatoirement deux jobs dans le tableau pour diversifier les solutions. Nous proposons de modifier légèrement la méthode en ajoutant un aspect aléatoire global. Ainsi la première étape consiste à sélectionner aléatoirement le job puis appliquer l'approche de localisation de [KHB02]. La figure 2.5 illustre ceci sur le même exemple donné précédemment.

Dans cet exemple, la première opération seléctionnée aléatoirement est O_{31} puis une mise à jour est effectuée sur la machine choisie.

	M1	M2	M3
O11	7	6	2
O12	8	2	1
O21	9	5	4
O22	2	5	1
O23	4	6	8
O31	3	2	1

	M1	M2	M3
O11	7	6	**3**
O12	8	2	**2**
O21	9	5	**5**
O22	2	5	**2**
O23	4	6	**9**
O31	3	2	1

	M1	M2	M3
O11	7	**11**	3
O12	8	**7**	2
O21	9	5	5
O22	2	**10**	2
O23	4	**11**	9
O31	3	**7**	1

......

FIGURE 2.5: Approche de localisation aléatoire

- Approche modifiée "MMKacem" :

  nous proposons de sélectionner aléatoirement le numéro du job et choisir la machine ayant le plus petit temps de traitement sans effectuer la mise à jour de charge du travail. La figure suivante illustre la méthode sur le même exemple.

	M1	M2	M3
O11	7	6	2
O12	8	2	1
O21	9	5	4
O22	2	5	1
O23	4	6	8
O31	3	2	1

	M1	M2	M3
O11	7	6	2
O12	8	2	1
O21	9	5	4
O22	2	5	1
O23	4	6	8
O31	3	2	1

	M1	M2	M3
O11	7	6	2
O12	8	2	1
O21	9	5	4
O22	2	5	1
O23	4	6	8
O31	3	2	1

......

FIGURE 2.6: Approche modifiée

La création du swarm initial ou essaim initial est obtenue par une combinaison de différentes approches avec un pourcentage différent. Ainsi nous optons à initialiser la population initiale 60% par l'approche modifiée, 20% la méthode aléatoire et 20% par la méthode de localisation de [KHB02]. La section suivante illustre l'algorithme d'PSO proposé.

2.3.6 L'algorithme PSO proposé

Selon la description ci-dessus, nous donnons l'algorithme PSO pour résoudre le FJSP déterministe (voir l'algorithme 1) avec : la variable *swarmsize* étant la taille de la population ou l'essaim, la variable *generation* représente la génération courante, $Max_{iteration}$ représente le critère d'arrêt qui est le nombre maximum de générations ou itérations. *pBestParticule* étant la meilleure position visitée par la particule et *gBestParticule* est la meilleure solution

globale.

Algorithme 1 L'algorithme PSO proposé

1: **initialisation des paramètres** $Swarmsize$, $Max_{iteration}$, k, c_1, c_2
2: *generation* **0**
-Création de l'essaim initiale 60 % l'approche modifiée "MMkacem", 20% approche de localisation, 20% méthode aléatoire.
-Évaluer chaque particule en calculant la fonction objective Cmax donnée par l'équation (2.5).
-initialiser *pBestParticule* de chaque particule par une copie d'elle-même.
-initialiser *gBestParticule* comme la particule globale qui a la plus petite valeur de la fonction objective dans la population courante.
3: **while** (generation $< Max_{iteration}$)
-*generation*=*generation*+1
-Mise à jour de la vitesse et de la position de chaque particule par l'équation (2.7) et (2.9).
-Evaluer la fonction objective de chaque particule.
-Mise à jour de *pbestParticule* de chaque particule.
-Mise à jour de la solution globale *gbestParticule* .
End while
4: **Sortie :** la meilleure solution globale *gBestParticule* (l'ordre des différentes opérations des différents jobs, l'affectation des machines, la valeur de fitness).

L'algorithme PSO ainsi présenté a été développé premièrement sur une architecture centralisée afin de tester et valider ces performances avant qu'il soit distribué.
La section suivante décrit les deux architectures distribuées proposées.

2.4 Les approches Multi agents basées sur PSO

Plusieurs approches distribuées ont été proposées pour résoudre le problème d'ordonnancement de type job shop flexible. L'avantage d'une organisation distribuée réside principalement dans la décentralisation du processus d'optimisation. Le but principal de l'utilisation d'une architecture distribuée étant d'obtenir une solution hautement performante à savoir une bonne qualité (Min makespan) mais aussi une robustesse et une stabilité face aux changements et aux événements non planifiés.
Nous avons opté pour l'utilisation des systèmes multi agents comme outils de distribution de l'algorithme. Deux architectures multi agents basées sur l'algorithme OEP nommées MAPSO1 et MAPSO2, sont ainsi proposées pour distribuer le processus de résolution du problème FJSP [NBJ+15b].
Dans cette section, nous présentons les deux architectures avec description de leurs structures, le nombre d'agents utilisés, le rôle de chaque agent, etc.

2.4.1 Multi Agents PSO 1 (MAPSO 1)

Afin de comparer le temps d'exécution entre le PSO centralisé présenté précédemment et le PSO distribué, nous proposons tout d'abord une architecture multi agents simple. Cette architecture se compose de quatre agents : un Agent Sychroniseur (AS) et trois Agents Exécutants (AE). Dans ce modèle, chaque agent dispose de ses propres connaissances (les agents qu'il connaît et avec lesquels il peut communiquer), une mémoire locale et une boîte aux lettres dans laquelle il stocke les messages reçus des autres agents.
Nous pouvons les décrire comme suit :

1. **Agent Exécutant (AE)** : Un processus d'optimisation basé sur la méthode d'optimisation par essaim particulaire est situé dans l'agent exécutant. Cet agent crée sa sous-population dans la première étape avec une méthode d'initialisation spécifique, puis exécute l'algorithme de PSO. Dans la deuxième étape, il transmet la meilleure particule trouvée à l'agent de synchronisation (SA). Le nombre d'agents executeur est configuré dans cette architecture selon le nombre de méthodes d'initialisation. Ainsi trois AE sont définis et chaque AE crée sa propre population par la méthode d'initialisation spécifiée, puis exécute l'algorithme.
2. **Agent Synchroniseur (AS)** Les connaissances de l'agent de synchronisation sont tous les agents existants dans le système. Cet agent est responsable du déclenchement de la phase de collecte (réception de la meilleure particule de chaque agent Exécutant EA) et détermine la meilleure particule globale.

L'architecture de la méthode et les différentes couches sont présentées dans la figure 2.7. Dans cette architecture, chaque agent crée son essaim par une méthode d'initialisation spécifique et exécute le PSO indépendamment des autres agents. Ceci permet de suivre plusieurs optima locaux en parallèle et d'augmenter la probabilité d'en trouver de nouveaux. L'utilisation des systèmes multi agents est consacrée entièrement, dans ce cas, au traitement parallèle de la méta-heuristique sans toucher à la force des applications distribuées à savoir la communication et la coordination.
Quelques limites ont été observées dans la première architecture. D'une part, l'agent de synchronisation est absent tout au long la phase d'optimisation et n'est introduit que dans la phase de collecte (réception de la meilleure particule de chaque EA), et d'autre part, l'aspect intelligent et communicant des agents est faible. La section suivante décrit la deuxième architecture proposée.

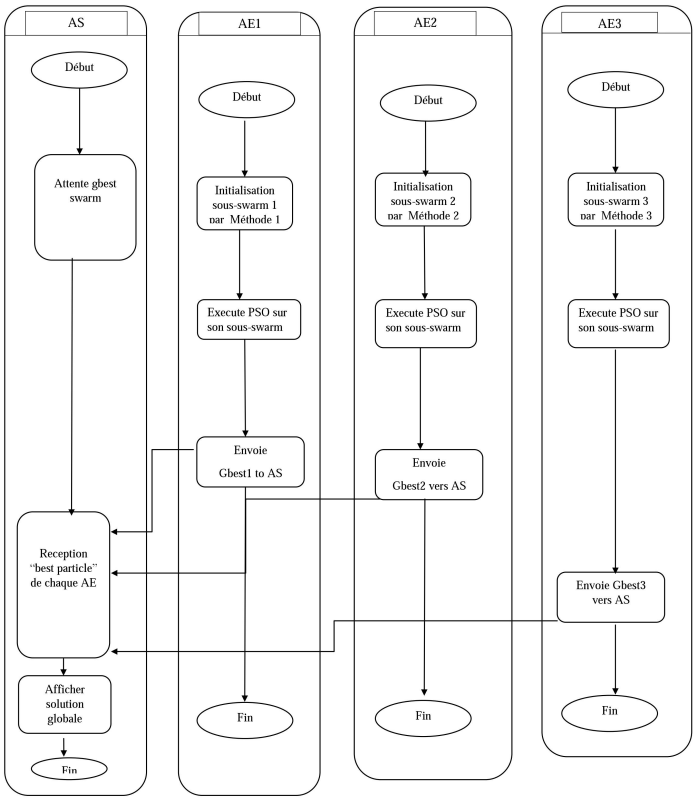

FIGURE 2.7: Architecture MAPSO1 proposée

2.4.2 Multi Agents PSO 2 (MAPSO 2)

Nous avons proposé une deuxième architecture MAPSO plus complexe que la première architecture MAPSO 1. Ce modèle est inspiré de [AZ10]. Cette architecture se compose d'un agent patron ou "Boss Agent" (BA), un agent sychroniseur SA et deux agents exécutants. Dans ce qui suit, nous donnons une description détaillée du comportement de chaque agent.

1. **Agent Patron (BA) :** Les connaissances de BA sont l'agent de synchronisation (SA). Il a deux responsabilités à savoir la création de la population initiale de l'algorithme PSO et l'optimisation globale. En effet, le BA envoie la population initiale à l'agent de synchronisation SA et attendra la réception du meilleur essaim des particules à travers l'agent SA pour enfin exécuter l'algorithme de PSO sur ce swarm et afficher finalement la meilleure particule ayant le minimum makespan.
2. **Agent Synchroniseur(SA)** : Cet agent divise la population reçue du BA en des sous-populations et il les envoie aux agents exécutants (AE). Cet agent est également responsable du déclenchement de la phase de migration et de la phase de collecte (réception de la meilleure particule de EA). Après la réception des meilleures particules, il envoie le meilleur essaim à l'agent patron BA.

3. **Agent Exécutant (EA) :** Cet agent exécute, dans la première étape, l'algorithme de PSO avec une sous-population reçue de l'agent synchroniseur. Dans la deuxième étape, il attend le début de la phase de migration, puis il transmet un sous-essaim de certains migrants à ses voisins conformément à la politique de migration. Ensuite, il exécute une deuxième fois l'algorithme PSO aprés la phase de migration. Enfin, la meilleure solution trouvée est envoyée à l'agent sychroniseur. Dans notre architecture, nous définissons deux agents exécutants.

   A noter que peuvent être configuré autant de nombre d'agents exécutants que de nombre de machines dans la cellule de production. Dans ce cas, nous devons définir l'espace de voisinage de chaque agent exécutant avant la phase de migration.

4. **La politique de migration :** Chaque EA exécute l'algorithme PSO sur sa sous-population, et envoie un message à l'agent de synchronisation l'informant de la fin de son exécution. L'agent sychroniseur SA coordonne la phase de migration. Après avoir reçu un message de tout les EAs indiquant leurs fin d'exécution, le SA diffuse un message notifiant le début de la phase de migration. Dans la phase de migration, chaque EA échange certains de ses meilleures particules avec ses voisins. Puis, il remplace ses mauvaises solutions par les meilleures de ses voisins.

La figure 2.8 montre les couches de notre architecture proposée MAPSO2 pour résoudre le problème FJSP. La figure 2.9 illustre la même architecture MAPSO2 avec plusieurs agent exécutants.

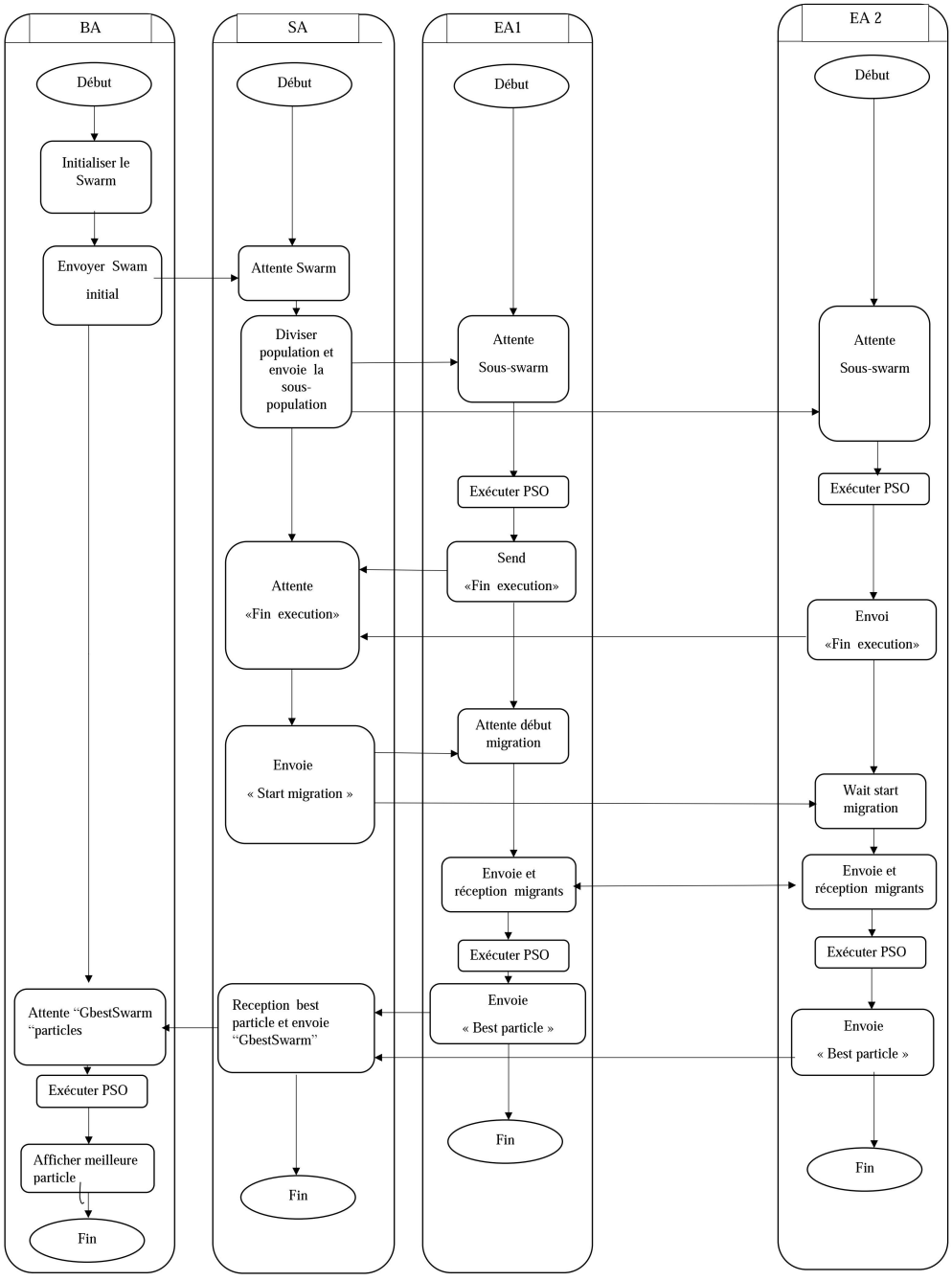

FIGURE 2.8: Architecture MAPSO2 proposée

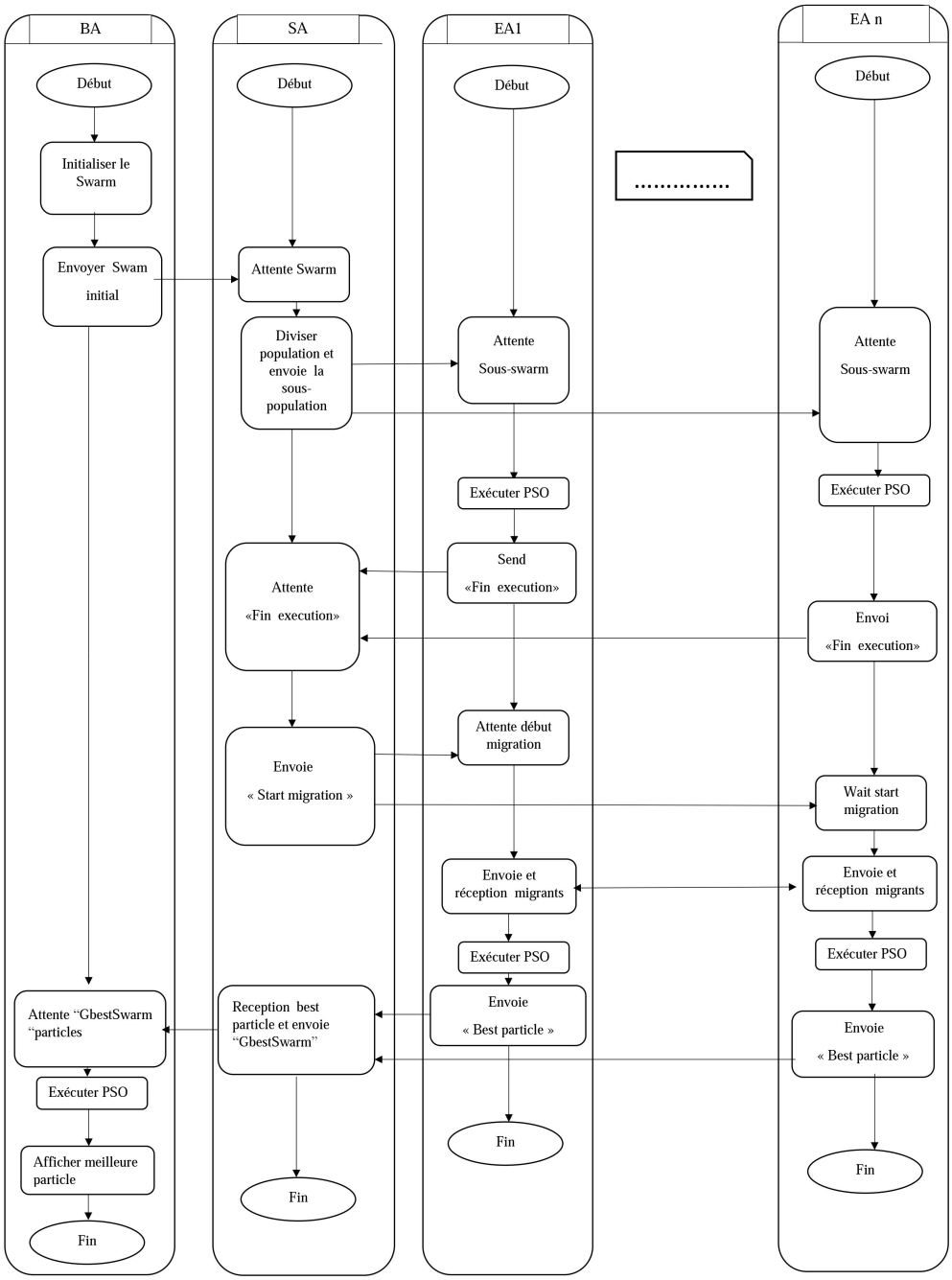

FIGURE 2.9: Architecture MAPSO2 proposée avec plusieurs agents exécutants

2.5 Expérimentation et analyse des résultats

Pour tester l'efficacité et la performance de l'algorithme de PSO abordé, nous avons réalisé des expériences avec trois différentes instances de FJSP : deux instances avec une flexibilité totale TFJSP et une à flexibilité partielle PFJSP. Les caractéristiques de ces exemples sont disponibles dans [JCT07] ; [MlSlH10]. Chaque instance peut être caractérisée par le nombre de job (n), le nombre de machines (m), et les opérations liées O_{ij} associé à chaque job j. Chaque instance est exécutée 20 fois et tous les résultats sont présentés dans la section suivante.

D'autres expériences sont ensuite effectuées sur les grandes instances de Brandimarte [Bra93]. Ces instances sont de grandes tailles dont le nombre des jobs varie de 10 à 20, le nombre de machines de 6 à 15 et le nombre d'opérations de 58 à 232.

Le système multi agent basé sur PSO pour le job shop flexible " the Flexible Jop Scheduling Multi Agent PSO" (MAPSO FJS) a été mis en oeuvre en utilisant l'IDE Netbeans avec le langage Java. Pour le développement de système multi-agents, nous avons choisi JADE (Java Agent Development framework), un logiciel gratuit et une plate-forme de développement de système multi agent open source. Ce middleware facilite la création d'agents et leur communication par l'envoi de messages. La mise en oeuvre a été réalisée sur une machine basée sur un processeur Intel "Core2Duo" cadencé à 2,0 GHz et 3070 MO.

2.5.1 Les instances de petites tailles

1. **Problème 4*5 :** Dans un premier temps, nous utilisons un petit exemple de l'échelle. Cet exemple est repris à partir de [MlSlH10]. Les paramètres de PSO sont les suivants : $swarmsize = 100$, $Max_{Iteration} = 200$.

   Le diagramme de Gantt illustré dans la figure 2.10 représente le meilleur ordonnancement trouvé par notre algorithme.

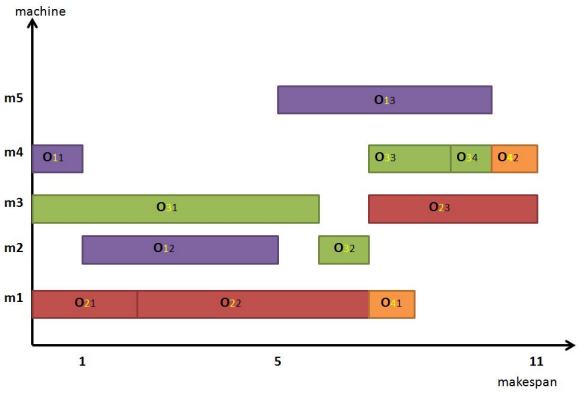

FIGURE 2.10: Le diagramme de Gantt de l'instance 1 (4 jobs * 5 machines)

La comparaison est faite entre notre algorithme et d'autres algorithmes bien connus pris de la littérature. La comparaison est présentée dans le tableau 2.1.

TABLE 2.1: Comparaison des résultats du probème 4 * 5

Algorithme	Makespan
notre PSO	11
AL+GA	16
PSO+SA	11
GA	13

avec :

"AL + GA" fait référence à [KHB02], "PSO+SA" fait référence à [XW05], "PSO+TS" fait référence à [ZSLG09] et "GA+HC" fait référence à [MlSlH10].

2. **Problème 10 * 10 :** Pour évaluer l'efficacité de l'algorithme de PSO proposé, une instance de taille moyenne qui est tiré de [JCT07] a été testé. Les paramètres de PSO sont les suivants : $SwarmSize = 500$, $Max_{Iteration} = 500$. La Figure 2.11 illustre le diagramme de Gantt de l'ordonnancement correspondant à la meilleure solution.

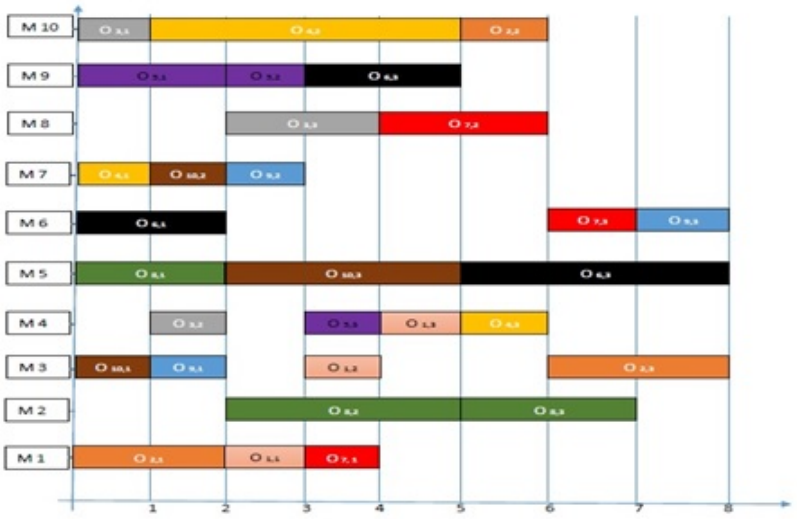

FIGURE 2.11: Le diagramme de Gantt de l'instance 2 (10 jobs * 10 machines)

La comparaison de l'algorithme proposée avec d'autres algorithmes pris de la littérature est présentée au tableau 2.2.

TABLE 2.2: Comparaison des résultats du probème 10 * 10

Algorithme	Makespan
notre PSO	8
Temporal Decomposition	16
Classic GA	7
Approach by Localization	8
PSO + SA	7
PSO + TS	8
GA+HC	8

avec :

"Temporal decomposition", "Classic GA" et "Approach by Localization" font référence à [KHB02], "PSO+SA"fait référence à [XW05], "PSO+TS" fait référence à [ZSLG09] et "GA+HC" fait référence à [MlSlH10].

3. **Problème 8 *8 :** Enfin, une instance de taille moyenne 8 x 8, avec une flexibilité partielle a été testé. Les caractéristiques de cette instance sont disponibles dans [MlSlH10]. Les paramètres de PSO sont les suivants : $swarmsize = 500$, $Max_{iteration} = 500$. La représentation graphique Gantt de l'ordonnancement correspondant à la solution optimiale est illustrée dans la figure 2.12.

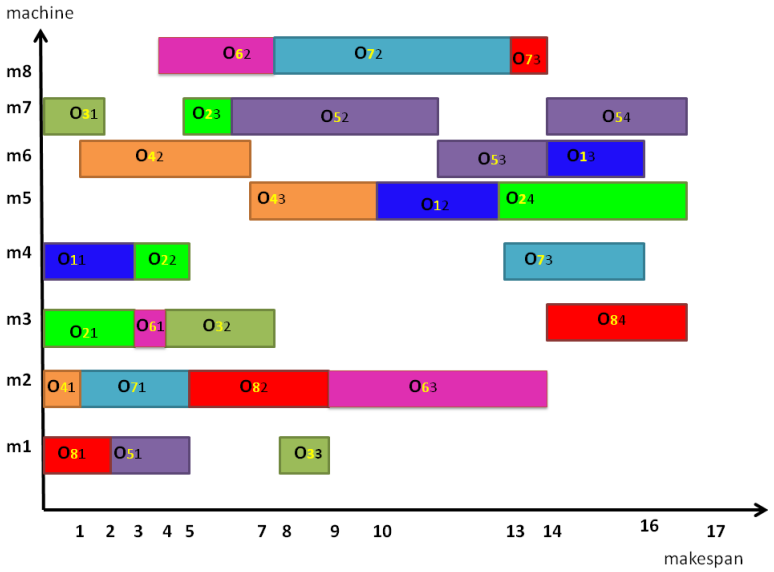

FIGURE 2.12: Le diagramme de Gantt de l'instance 3 (8 jobs * 8 machines)

La comparaison de PSO proposé avec d'autres algorithmes est illustrée dans le tableau 2.3.

TABLE 2.3: Comparaison des résultats du probème 8 * 8

Algorithme	Makespan
notre PSO	17
Classic GA	16
AL + CGA	16
PSO + SA	15
PSO+TS	15

Avec

"Classic GA" et "AL+CGA" font [KHB02], "PSO+SA" fait référence à [XW05], "PSO+TS" fait référence à [ZSLG09].

Les résultats des calculs ont montré que le PSO développé est efficace pour résoudre le problème d'ordonnancement du job shop flexible sans hybridation . Ceci prouve que le codage de la particule utilisé, la façon de mettre en oeuvre le déplacement, l'ajout de calcul de distance entre les particules et la création de la population initiale par plusieurs méthodes permettent de mieux explorer l'espace de recherche et réduirent le risque que l'algorithme de PSO tombe dans un optimum local.

2.5.2 Les instances de Brandimarte

Pour mieux évaluer la performance de notre algorithme, nous avons testé le PSO proposé sur les benchmarks de Brandimarte [Bra93]. Ces instances se caractérisent par un grand

nombre d'opérations et des machines. Les résultats obtenus sont présentés dans le Tableau 2.4 ainsi que la comparaison avec d'autres méthodes de la littérature.

TABLE 2.4: Comparaison des résultats des instances de Brandimarte

Instance	TS	PSO	PSO+TS	MATSLO	notre PSO
MK1	42	40	40	40	41
MK2	32	27	27	32	**26**
MK3	211	204	204	207	207
Mk4	81	62	63	67	65
Mk5	186	178	173	188	**171**
Mk6	86	78	65	85	**61**
Mk7	157	147	145	154	173
Mk8	523	523	523	523	**523**
Mk9	369	341	331	437	**307**
Mk10	296	252	223	380	312

Comme nous pouvons le constater dans le tableau 2.4, notre PSO proposé donne de bons résultats pour plusieurs instances de Brandimarte. Cependant, certains résultats obtenus sont pires ; ceux-ci sont dûs à la convergence prématurée du PSO. Nous rappelons que notre objectif n'est pas seulement d'obtenir le meilleur makespan, mais aussi de trouver le PSO le plus simple possible à mettre en oeuvre dans le système embarqué pour les futurs travaux.

2.5.3 Résultats des approches distribuées

Pour illustrer les performances de nos modèles système Muti-agents proposés, des tests sur les benchmarks de Kacem et celles de Brandimarte sont effectués. Nos expériences ont été accentuées sur le makespan et le temps CPU.

Afin d'illustrer la performance des différentes solutions PSO, nous avons calculé le temps d'exécution de chaque approche. Les résultats sont présentés dans le tableau 2.5. Pour comparer le temps CPU de PSO et le temps CPU de MAPSO, nous utilisons la même configuration. Les modèles PSO, MAPSO1 et MAPSO2 proposés sont exécutés 20 fois pour chaque problème.

TABLE 2.5: Comparaison de temps CPU entre PSO centralisé et MAPSO

Instance	Makespan	Temps CPU de PSO (s)	Temps CPU de MAPSO1 (s)	Temps CPU de MAPSO2 (s)
4*5	11	0.353	0.332	14.610
10*10	8	7.316	7.307	36.984
8*8	17	5.698	5.589	27.816

Les résultats montrent que les deux architectures multi agent PSO trouvent la même qualité de solution mais MAPSO1 est plus rapide que PSO classique. Cela est dû à la parallélisation de PSO dans différents agents exécutants de sorte que chaque sous-population évolue séparément.

Pour la MAPSO 2 proposée, l'agent sychroniseur SA divise l'essaim reçu de l'agent BA en deux sous-populations et les envoie à l'agent d'exécutant (AE). La taille de la sous-population de chaque agent est égale à la moitié de la taille des essaims. Le nombre de particules qui seront échangées dans la phase de migration est égal à un tiers du sous-essaim.

L'avantage de modèle MAPSO2 consiste à l'intégration systématique de tous les agents dans la phase d'optimisation ; la politique de migration étant une méthode récemment utilisée afin de guider la recherche dans de nouvelles zones de l'espace de recherche et trouver ainsi la meilleure particule.

Cependant l'architecture MAPSO2 nécessite un temps de traitement plus grand par rapport à celui de PSO et MAPSO1. Cette augmentation de CPU time pour l'architecture MAPSO2 est justifiable. En effet, ceci s'explique d'une part par l'exécution de l'algorithme PSO deux fois par chaque agent comparé aux PSO centralisé et MAPSO1. D'autre part, l'ajout d'une nouvelle phase de migration et l'envoi des messages et la communication entre les agents et l'agent synchroniseur engendrent systématiquement une augmentation dans le temps d'exécution.

2.6 Conclusion

Dans ce chapitre nous avons présenté l'algorithme d'optimisation par essaims particulaires, proposé pour résoudre le problème de job shop flexible, en détaillant le codage de la particule utilisée, les méthodes de création de l'essaim, la configuration des paramètres, etc. Nous avons discuté aussi la distribution de l'algorithme PSO en utilisant les systèmes multi agents. Deux architectures multi agents basées sur le PSO, nommées MAPSO1 et MAPSO2 sont ainsi présentés ainsi qu'une description détaillée du rôle de chaque agent dans les deux

architectures proposées.

Nous avons testé notre algorithme sur des différentes instances de petites et grandes tailles avec une flexibilité totale et partielle. Les résultats obtenus montrent que l'algorithme PSO proposé est efficace pour résoudre le problème de job shop flexible sans la prise en compte des incertitudes. Bien que notre algorithme est distribué sur un ensemble d'agents, les agents restent centralisés sur la même machine. Le déploiement de l'architecture proposée sur un système physiquement distribué s'avère aussi importante. Nous détaillerons cette éventualité dans le chapitre suivant.

Chapitre 3

Implémentation d'une métaheuristique embarquée PSO

3.1 Introduction

Ce travail discute l'ajout des systèmes embarqués dans le déploiement d'un processus d'optimisation intelligent et coopératif comme le PSO afin de résoudre le problème d'ordonnancement sous perturbations dans l'industrie.
Notre approche pour résoudre le problème dynamique d'ordonnancement consiste à rendre la méthode de résolution distribuée en utilisant des systèmes embarqués. Chaque solution est représentée par un ensemble d'entités décentralisées capables de décision.
Afin que ces solutions soient faisables et cohérentes, les entités doivent communiquer et coopérer entre elles.
Dans ce chapitre, nous détaillerons l'implémentation de l'architecture MAPSO2 sur un système physiquement distribué composé de deux systèmes embarqués ainsi que les nouvelles améliorations proposées.

3.2 Les systèmes embarqués

De nos jours les systèmes embarqués sont omniprésents. Ils nous entourent et nous sommes littéralement envahis par eux.
Un système embarqué peut être défini comme un système électronique et informatique autonome, dédié à une tâche précise. Un système embarqué est conçu pour répondre à un besoin particulier, c'est-à-dire pour réaliser une application précise dans un domaine particulier (TV numérique, télécommunication, musique, industrie automobile, industrie spatiale, etc).
Le déploiement de ces technologies informatiques intégrées dans les entreprises contribuera à leur évolution en entreprises manufacturières intelligentes (Smart Manufacturing).

La relation entre les méthodes de recherches opérationnelles et le système embarqué est étroite. De nombreuses méthodes d'optimisation sont proposées pour résoudre les problèmes détectés dans les systèmes embarqués. On peut citer [PAL+13] qui ont utilisé les méta-heurstiques suivantes TS, SA et GA pour résoudre le problème de partitionnement (hardware/software). [SRS12] utilisent les métaheuristiques pour déterminer la structure des données de la mémoire cache réduisant la consommation d'énergie. [PVS09] ont proposé une implémentation matérielle de PSO. Tous ces travaux utilisent les métaheuristiques comme un outil pour résoudre le problème observé dans les systèmes embarqués.
Néanmoins dans notre travail, les métaheuristiques représentent elles-mêmes l'application embarquée. Notre objectif étant d'intégrer ces systèmes embarqués dans l'industrie afin d'offrir une solution flexible robuste et construire un système coopératif souple, capable de prendre les décisions d'ordonnancement tout en prenant en charge promptement les imprévus et les incertitudes internes liés à la nature dynamique de l'environnement de production.

3.3 L'architecture MAPSO2 distribuée

Comme nous l'avons vu au chapitre précédent, l'architecture MAPSO2 est composée d'un agent patron BA , un agent synchroniseur SA et deux agents Exécutants. Tous les agents sont localisés dans le même conteneur "main container" et surviennent sur la même machine (Voir Figure 3.1).

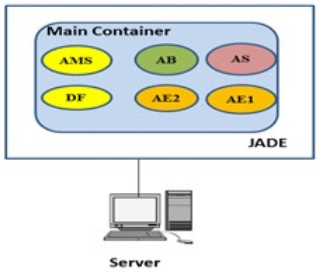

FIGURE 3.1: L'architecture MAPSO2 centralisée

L'agent AMS (Agent Management System) et le DF (Directory Facilitator) existent par défaut dans le "main container" et ne font pas partie de notre modèle. Néanmoins la distribution reste virtuelle et n'est pas une distribution physique à distance. La distribution de la méta-heuristique en utilisant le système multi agent n'est qu'une première étape avant d'être réellement distribuée sur une architecture physiquement distribuée.
Les agents dans ce cas de figure vivent dans un système réel distribué distant et non pas

sur un hôte local centralisé, alors un conteneur est créé pour chaque agent distribué distant. En effet, la vie d'un agent exige la présence d'un conteneur. Selon JADE, tous les nouveaux conteneurs créés doivent être connectés au conteneur parent principal (Main Container).
Ainsi avant d'être déployée sur le système distribué composé des systèmes embarqués, la nouvelle architecture MAPSO2 a été modifiée. La nouvelle est illustrée dans la figure 3.2.

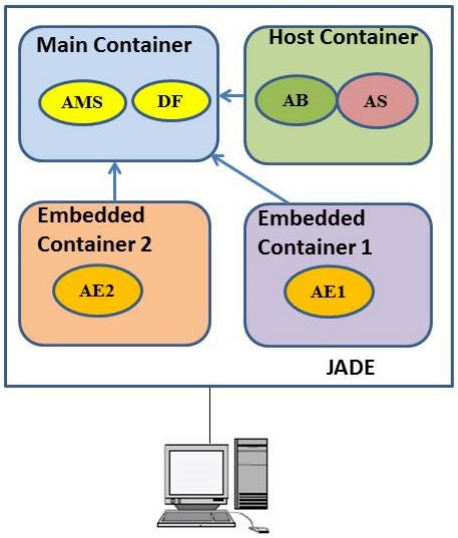

FIGURE 3.2: Nouvelle architecture de MAPSO2 sous JADE

Hostcontainer, Embedded Container 1 et Embedded Container 2 sont des conteneurs respectivement pour l'agent AS et AB, EA1 et EA2. Les agents SA et BA vont être localisés sur la machine centrale "server machine" et chaque agent EA va être intégré sur un système embarqué.
Deux systèmes embarqués ARM A7 sont utilisés, l'un comme conteneur pour AE1 et le second pour l'AE2. La figure 3.3 illustre l'architecture MAPSO2 distribuée.
La simulation, l'analyse et la comparaison entre MAPSO2 centralisée et MAPSO2 distribuée embarquée en termes de temps d'exécution et de la qualité de la solution sont présentées dans la section suivante.

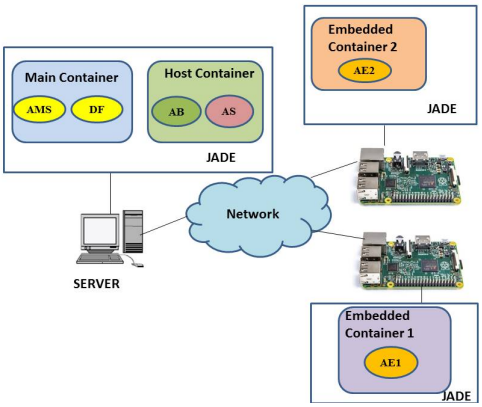

FIGURE 3.3: L'architecture MAPSO2 distribuée

3.4 Expérimentation et validation des performances de l'architecture MAPSO2 distribuée

Pour tester l'efficacité et la performance de l'architecture MAPSO2 distribuée, nous avons effectué des expériences sur trois différentes benchmark FJSP : deux instances avec une flexibilité totale T-FJSP (4*5 et 10*10) et une à flexibilité partielle P- FJSP (8*8). Chaque instance est exécutée pendant 20 fois.

Le système distribué utilisé pour simuler l'architecture MAPSO2 distribué est composé d'un PC et de deux systèmes embarqués ARM A7. Le PC a un processeur Intel "Core2Duo" cadencé à 2, 4 GHz et 8 Go de RAM, tandis que la carte intégrée est à 900 Mhz et 1Go de RAM. Le système embarqué cible exécute la version Linux de Debian. Toutes les machines exécutent Java SE Embedded. La figure 3.4 représente le système distribué utilisé. Le BA

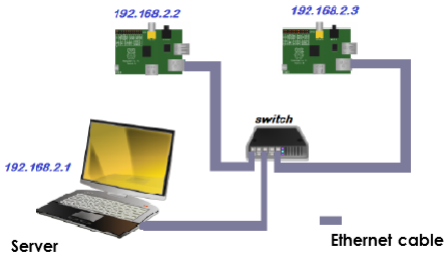

FIGURE 3.4: L'architecture physique du MAPSO2 distribuée

et le SA sont implémentés sur la machine serveur et chaque agent exécutant est implémenté sur un système embarqué cible. Un LAN est utilisé pour la communication entre tous les composants du système distribué. Le tableau 3.1 résume les résultats (makespan et temps de traitement) de la simulation des trois instances FJSP en utilisant le MAPSO2 centralisé sur un seul PC et MAPSO2 distributée en utilisant le deux systèmes embarqués.

TABLE 3.1: Comparaison entre MAPSO2 centralisé et MAPSO2 distribué

Instance	MAPSO2 centralisé		MAPSO2 distribué	
	Makespan	temps CPU (ms)	Makespan	temps CPU (ms)
4*5	**11**	14610	**11**	36819
10*10	**8**	36984	**8**	54732
8*8	**17**	27816	**17**	48654

Nous constatons que les valeurs de makespan sont les mêmes. Cependant, le temps d'exécution est plus lent dans le système physiquemnt distribué comportant deux systèmes embarqués. Comme on peut le constater dans le tableau 3.1, la solution distribuée nécessite plus de temps que la solution trouvée localement. Ceci est dû en premier lieu à la différence de fréquence du processeur et, d'autre part, au coût de la communication entre tous les composants du système physiquement distribué. Nous proposons ultérieurement une amélioration pour réduire le temps d'exécution de l'architecture MAPSO2.

3.5 Les limites de l'architecture MAPSO2 distribuée

Comme vue précédemment, la méthode MAPSO2 implémentée résoud efficacement le problème du FJSP en termes de qualité de la solution (Minimisation du makespan) par rapport à d'autres travaux de référence de la littérature. Cependant, malgré les résultats prometteurs obtenus dans ce modèle, certaines limites sont observées.
En effet, dans l'architecture MAPSO2, l'agent synchroniseur SA entre dans une phase d'attente de réception du message "Fin d'exécution" de tous les agents EA pour lancer la phase de migration en envoyant le message "Démarrer la migration" à tous les EA. Cependant, les machines où les agents exécutants sont mis en oeuvre peuvent subir à tout moment un arrêt accidentel en raison de la nature dynamique du système de fabrication réel (panne des machines aléatoires).
Dans ce cas, SA sera immobilisé dans un état d'attente. Le processus d'optimisation est ainsi bloqué et aucune solution au problème ne sera trouvée.
Outre les pannes et les défaillances des machines susceptibles de survenir à tout moment sur la machine $m1$ ou sur la machine $m2$, il est également important de garantir que EA1 et EA2 mènent les particules aux meilleures solutions. Cet objectif sera atteint en diversifiant la sous-population de chaque agent exécutant. L'agent SA est le responsable pour répartir et diviser la population initiale en sous-populations. Dans ce qui suit, nous proposons une amélioration pour réduire le temps d'exécution CPU de l'architecture MAPSO2 distribué. Nous

proposons aussi une version améliorée de MAPSO2 et nous étudions l'effet de la division de la population initiale sur la qualité de la solution.

3.6 Equilibrage de charge de MAPSO2 distribué

Afin d'améliorer le temps d'exécution qui est un facteur critique, nous proposons dans l'architecture MAPSO2 une approche pour équilibrer la charge de travail de tous les agents. Ceci est réalisé en envoyant une partie de l'essaim initial et non pas la totalité aux agents exécutants. Comme l'agent sychroniseur et l'agent patron sont développés dans un serveur, nous proposons de mieux exploiter cette ressource. Ces expériences sont effectuées pour étudier l'effet de la taille du sous-essaim de chaque agent sur le temps d'exécution global. Le tableau suivant représente le temps d'exécution du MAPSO2 distribué proposé lors de la variation du pourcentage d'envoi de population vers la SA.

TABLE 3.2: Résultats de MAPSO2 distribué en variant le pourcentage du l'essaim envoyé

Instance	taille de l'essaim	Pourcentage envoyé	Max-iteration	Temps CPU(ms)
4*5	200	40%	200	24391
4*5	200	20%	200	13500
4*5	200	10%	200	7703
4*5	200	8%	200	6344

Selon le tableau 3.2, lorsque l'AS reçoit une petite partie de la population initiale, par conséquent, chaque élément agent exécutant doit avoir une charge de traitement plus faible. Ceci est une caractéristique très importante dans le cas où nous visons une implémentation PSO sur plusieurs systèmes embarqués ayant chacun une mémoire limitée et des ressources de puissance de traitement. L'équilibrage de charge de travail des agents améliore le temps d'exécution par rapport au MAPSO2 distribué originale de 36819 ms à 6312 ms (l'itération maximale = 200, la taille des essaims = 200) lorsque la SA ne reçoit que 6% de la population ; Et de 14610 ms à 6312ms par rapport au MAPSO2 centralisé. Sur la base de ces séries d'expériences, la réduction de la taille des essaims de la population amoindrit généralement le temps d'exécution. En utilisant l'équilibrage de charge, l'architecture devient plus flexible même en cas de coupure de communication car l'agent patron BA va exécuter le PSO sur sa sous population

Pour éviter le problème du blocage total de l'architecture MAPSO2 lors de la survenue d'une panne, nous avons proposé une architecture nommée MAPSO2+. Une description détaillée est donnée dans la section suivante.

3.7 L'architecture améliorée MAPSO2+

Pour surmonter les limitations observées dans l'architecture MAPSO2, une version améliorée de multi-agent PSO, nommée (MAPSO2+), est proposée [NBJ+15a]. Tout d'abord, les délais d'attente sont ajoutés à l'agent SA afin d'éviter les états de blocage total. Ces délais sont ajoutés à deux niveaux. Le premier est ajouté lors de l'attente du message "Fin d'exécution". Le second est placé lors de l'attente des meilleures particules trouvées par l'agent exécutant.

De plus, il est important d'ajouter une phase pour envoyer des solutions provisoires à l'agent SA au fur et à mesure du début du processus d'optimisation dans chaque agent EA. La solution intérimaire représente la solution courante "$gbestCurrent$" obtenue par chaque EA avant la fin du processus d'optimisation et/ ou la meilleure particule finale obtenue dans la première phase d'optimisation nommée "$gbestfinal$".

Les couches de l'architecture MAPSO2 améliorée proposée pour résoudre le FJSP sont présentées dans la figure 3.5.

Les changements des rôles de chaque agent par rapport à la première architecture MAPSO2 sont les suivants :

- **Agent Exécutant (AE) :**

  L'unique nouveau rôle de l'agent AE est d'envoyer une solution provisoire à l'agent SA lors de la première exécution de l'algorithme PSO. Ceci est réalisé une première fois après un certain nombre d'itérations pour lesquelles la solution provisoire représentera la meilleure particule courante trouvée et une seconde fois quand le processus d'optimisation est terminé avec succès sans aucune défaillance (voir l'algorithme numéro 2). L'agent exécutant peut envoyer aussi plusieurs solutions courantes avant la fin d'exécution du PSO. Aprés chaque $fixednumber$ itérations, l'agent AE envoie la meilleure solution courante trouvée. Dans ce cas l'algorithme de l'AE est comme suit (voir l'algorithme numéro 3). Néanmoins, cette configuration va augmenter le nombre de messages envoyés à l'agent de sychronisation (SA).

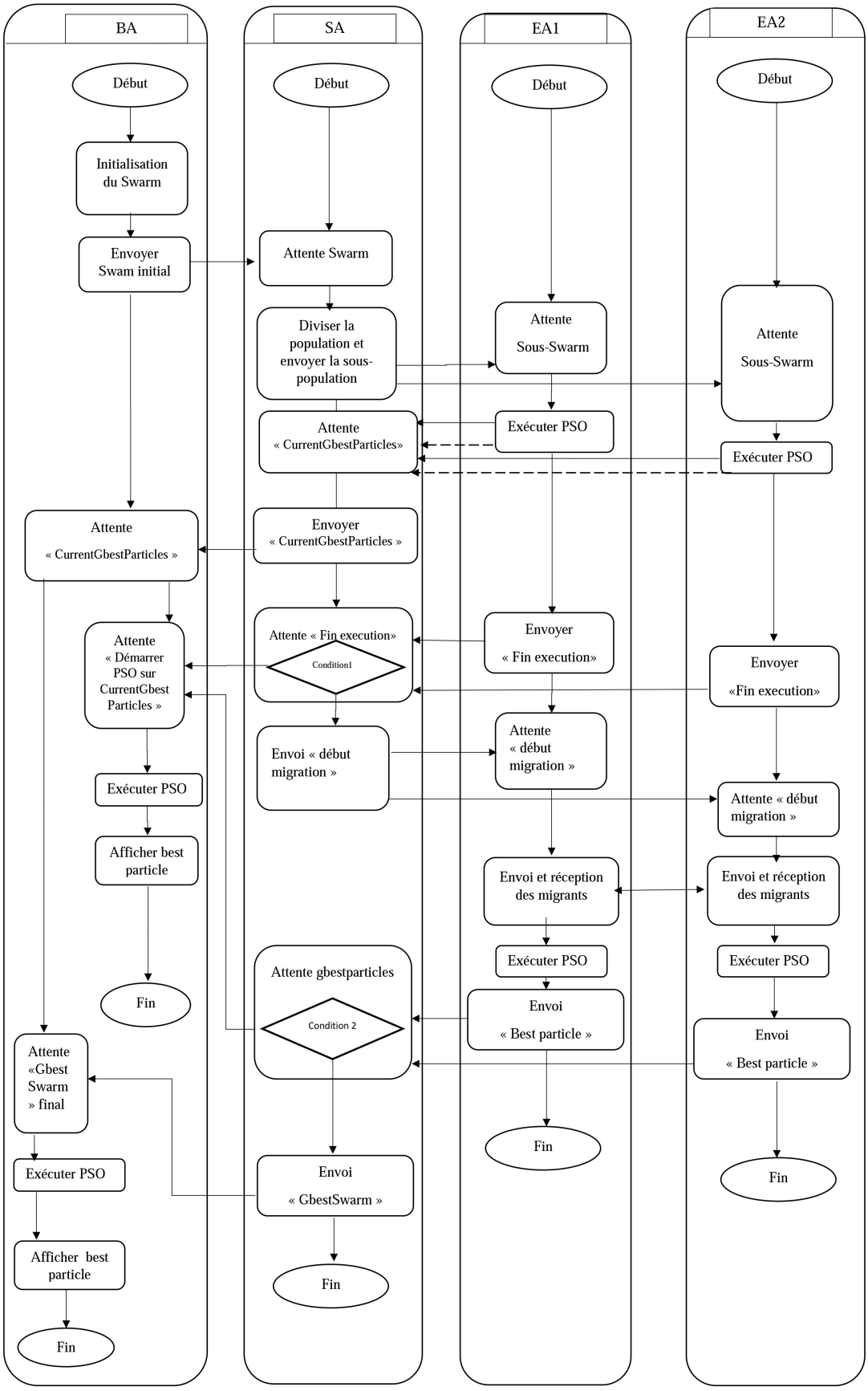

FIGURE 3.5: Organigramme représentant la nouvelle architecture MAPSO2+

Algorithme 2 L'algorithme de l'agent Exécutant EA de l'architecture MAPSO2+(v1)

Step 1 : wait sub population
Step 2 : Run PSO(sub-swarm)
if $(iteration == fixednumber)$ **then**
Send (currentGbestParticle, SA)
Endif
If $(iteration == Max_{iteration})$ **then**
Send (finalGbestParticle, SA)
Endif
Step 3 : Wait "start migration"
Step 4 : Send (migrants, EA)
Step 5 : Run PSO (newSwarm).
Send (gbestParticle, SA)
-END

Algorithme 3 L'algorithme de l'agent Exécutant AE de l'architecture MAPSO2+(v2)

Step 1 : Wait sub population
Step 2 : Run PSO(sub-swarm)
Step 3 : $k = 0$
if $((iteration - k) == fixednumber)$ **then**
{
Send (currentGbestParticle, SA)
$k = iteration$
}
Endif
If $(iteration == Max_{iteration})$ **then**
Send (finalGbestParticle, SA)
Endif
Step 3 : Wait "start migration"
Step 4 : Send (migrants, EA)
Step 5 : Run PSO (newSwarm).
Send (gbestParticle, SA)
-END

- **Boss Agent (BA) :**

  Après l'envoi de la population initiale à l'agent SA, le BA attend la réception de *currentgbestSwarm* de SA. BA attend alors de recevoir soit le message "Démarrer PSO sur currentGbestSwarm " ou un message contenant $finalgbestSwarm$ de SA. À la fin, BA exécute l'algorithme PSO sur *currentgbestSwarm* ou $finalgbestSwarm$ respectivement en fonction du message reçu et délègue la meilleure solution globale *gbest*.

Algorithme 4 L'algorithme de l'agent patron BA de l'architecture MAPSO2+

Step 1 : Initialization of initial swarm
Step 2 : Send (swarm, SA)
Step 3 : Wait actualGbestSwarm from SA
Step 4 : Receive actualGbestSwarm
Step 5 : *if* messagerecu.getContent( ).equalsIgnoreCase("Start PSO on actualGbestSwarm") **then**
Run PSO (actualGbestSwarm)
Show bestparticule.
Else
Wait receiving finalGbestSwarm from SA
Run PSO (finalGbestSwarm).
Show bestparticule
Endif
-END

- **Agent de sychronisation (SA) :**

  L'agent SA commence par diviser la population initiale reçue de l'agent patron BA en sous-populations à envoyer chacune à un agent Exécutant EA. SA attend ensuite de recevoir les meilleures particules trouvées par chaque agent EA nommées *currentbestparticles*. Le *currentgbestSwarm* obtenu est renvoyé à l'agent BA. Ensuite, l'agent SA entre en état d'attente pendant une période de temps donnée. Si le timeout expire et qu'aucun message "Fin Exécution" n'est reçu, le SA envoie le message "Démarrer PSO sur *currentGbestSwarm*" à l'agent BA. Sinon, il continue à se comporter normalement comme dans MAPSO2 en envoyant un message de " Début de phase de migration" aux agents EA. En outre, une défaillance de la machine peut également se produire pendant la phase de migration. Un second délai d'attente est ainsi ajouté lors de la phase de réception de la meilleure particule provenant d'agents EA. De même, si le timeout est atteint ; l'agent SA envoie le message "Démarrer PSO sur currentgbestSwarm" à l'agent BA. Sinon, il envoie le meilleur essaim global nommé $finalgbestSwarm$, après avoir reçu toutes les meilleures particules trouvées après la phase de migration de chaque agent EA, à l'agent patron BA.

Algorithme 5 L'algorithme de l'agent sychroniseur SA de l'architecture MAPSO2+

*Step 1 :*wait initial swarm
*Step 2 :*divide swarm and sending sub population to each EA agent.
*Step 3 :*wait receiving actualGbestParticule from EA
Step 4 : send (actualGbestSwarm, BA)
Step 5 :
do
-wait "End Execution" message from EA.
-time ++ ;
while((time>waitingdelays) || (messagerecu.getContent( ).equalsIgnoreCase("End Execution").
- **if** ((time>waitingdelays) & (! messagerecu.getContent( ).equalsIgnoreCase("End Execution")) **then**.
-send ("Start PSO on actualGbestSwarm", BA)
- **Else**
-send ("start migration", EA)
-wait receiving best particles
-time ++ ;
-**while**((time>waitingdelays) || (messagerecu != null)
-**if** ((time>waitingdelays) & (messagerecu == null) **then**
-send ("Start PSO on actualGbestSwarm", BA)
- **Else**
-send (finalGbestSwarm, BA)
-**Endif**
-**Endif**
-END

3.8 Expérimentation et validation des performances de l'architecture MAPSO2+

Pour tester l'efficacité et la performance de la nouvelle architecture MAPSO2+, nous avons effectué des expériences sur les mêmes trois différentes benchmark FJSP (4*5 ; 10*10 ; 8*8).

Les paramètres PSO sont les suivants : $swarmsize$= 100, $Max_{iteration}$= 200. La variable $fixediteration$ dans l'algorithme de l'agent exécutant définit quand est ce que chaque EA envoie à l'agent sychroniseur SA la meilleure particule courante "$currentgbestParticle$". Dans notre cas, l'agent exécutant envoie la solution provisoire une fois aprés chaque 3 itérations et une dernière fois lorsque ($iteration$== $Max_{iteration}$) est vérifiée.

Des expériences sont menées pour étudier la façon dont l'essaim initial est divisé en sous-populations et voir son effet sur la qualité de la solution trouvée. Dans la deuxième partie d'expériences, nous évaluerons le résultat de l'amélioration par rapport à l'architecture précédente de MAPSO2 en cas de panne des machines $m1$ ou $m2$.

3.8.1 Influence de la division initiale de l'essaim initial

Dans l'architecture MAPSO2+ améliorée, trois approches d'initialisation différentes sont utilisées pour créer l'essaim initial avec des pourcentages différents. L'essaim initial est alors divisé entre les différents agents EA. Dans la première expérience, nous avons commencé à étudier l'effet de la façon dont cette division est effectuée sur la qualité de la solution trouvée par chaque agent exécutant. Trois cas d'études sont présentés. Dans chaque cas, une combinaison donnée des approches d'initialisation est utilisée avec un partitionnement différent de la population initiale parmi les deux agents EA. La qualité de la solution de chaque EA est ensuite discutée.

- **cas numéro 1 :** L'agent patron BA génère la population initiale comme suit : 60% MMKacm, 20% KacemHammadi, 20% aléatoire. L'agent SA divise cet essaim initial et envoie 50% de celui-ci à chaque EA (Voir figure 3.6.a).
- **cas numéro 2 :** Dans ce cas, l'agent BA génère la population initiale comme suit : 20% MMKacm, 30% KacemHammadi, 50% aléatoire (Voir figure 3.6.b).
- **cas numéro 3 :** L'agent SA envoie à chaque EA 10% aléatoire, 30% MMKacem, 10% Kacem (Voir figure 3.6.c).

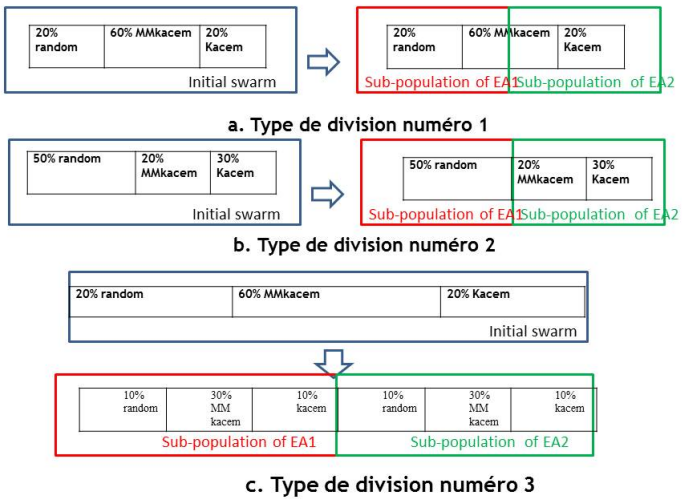

FIGURE 3.6: Les types de division

Le tableau suivant montre la comparaison entre les meilleurs résultats de particules trouvés par chaque EA après la fin du processus d'optimisation en utilisant ces différents types de division. La meilleure performance (valeur minimum de fabrication : makespan) obtenue est imprimée en gras.

TABLE 3.3: Comparaison de la qualité de la solution trouvée par chaque agent exécutant

instance	Cas numéro 1		Cas numéro 2		Cas numéro 3	
	C_{max} trouvé par EA1	C_{max} trouvé par EA2	C_{max} trouvé par EA1	C_{max} trouvé par EA2	C_{max} trouvé par EA1	C_{max} trouvé par EA2
4*5	**11**	13	**11**	12	12	**11**
10*10	**8**	10	**9**	10	**8**	**8**
8*8	**17**	18	**17**	18	**17**	**17**

Comme nous pouvons le constater dans le tableau 3.3, il existe une différence entre les valeurs du makespan de la meilleure particule trouvée par chaque agent exécutant dans le premier et le second cas (cas 1 et 2).

Les défaillances et les pannes pouvant survenir à tout moment sur la machine m1 ou la machine m2, il est important de garantir que EA1 et EA2 mènent le deux à la fois à des particules de bonne qualité. Lorsque l'agent sychroniseur divise la population initiale, il est important de diversifier chaque sous-population pour chaque agent EA afin d'avoir le même espace de recherche varié par toutes les méthodes d'initialisation. Même en cas de panne sur l'une des machines $m1$ ou $m2$, cette division nous garantit de ne pas perdre certaines solutions optimales trouvées par chacune des méthodes d'initialisation existantes. Pour le

troisième cas, EA1 et EA2 ont trouvé le même résultat dans l'instance 10 * 10 et l'instance 8 * 8. Ce dernier cas assure la meilleure façon de division de l'essaim initial car les performances obtenues sont similaires pour chaque EA, la diversification ayant été la même pour chaque sous-population.

3.8.2 Comparaison de performance entre MAPSO2 et MAPSO2+

Dans cette section, nous procèderons à la comparaison entre MAPSO2 et la version améliorée pour surmonter les pannes de machines de contrôle.
Le tableau 3.4 illustre les résultats trouvés. Comme le montre le tableau 3.4, notre algorithme est capable de trouver une solution même en cas de panne des machines de contrôle. Il faut noter que la valeur de makespan indiqué dans le tableau est celle trouvée avant d'appliquer la procédure de réparation des pannes machines.

TABLE 3.4: Comparaison de la qualité de la solution trouvée par MAPSO2 et MAPSO2+

Instance	Makespan de la solution finale par BA	
	MAPSO2+	MAPSO2
4*5	**11**	No solution
10*10	**8**	No solution
8*8	**17**	No solution

Afin de comparer le temps d'exécution de l'architecture MAPSO2 distribuée et celle MAPSO2+, nous avons utilisé le même système distribué composé d'un serveur et de deux systèmes embarqués lors des expérimentations. Le tableau 3.5 illustre les résultats de comparaison.

TABLE 3.5: Comparaison de temps CPU entre MAPSO2 distribué et MAPSO2+

Instance	Temps CPU(ms)	
	MAPSO2 distribué	MAPSO2+
4*5	36819	41837
10*10	54732	59621
8*8	48654	57421

Comme le montre le tableau 3.5, l'architecture améliorée MAPSO2+ nécessite un temps CPU plus grand que celui de l'architecture MAPSO2 distribuée. Cependant, l'architecture MAPSO2+ est beaucoup plus flexible et adaptée à l'environnement dynamique de l'atelier de production.

3.9 Limites de l'architecture MAPSO2+

Des modifications ont été apportées sur l'architecture MAPSO2 afin de lui donner plus de flexibilité tout en s'adaptant à l'environnement dynamique, en particulier lorsque les pannes machine sont considérées.

Ceci a été réalisé en ajoutant une période de temporisation pour contrôler d'une part l'état d'attente de l'Agent Synchroniseur (SA) et d'autre part éviter le blocage total. Ces délais d'attente sont ajoutés à deux niveaux : Le premier timeout est ajouté au niveau d'attente du message "End Execution". Le second est placé lors de la phase d'attente des meilleures particules de l'agent EA.

Une nouvelle phase d'envoi de solutions provisoires intermédiaires est également ajoutée dans le comportement de l'agent exécutant (EA). Ces solutions représentent les meilleures particules courantes trouvées par l'AE à une itération spécifique avant l'achèvement du processus d'optimisation.

Malgré les résultats prometteurs obtenus par le modèle MAPSO2+ centralisé sur la machine, certaines limites sont observées après son utilisation sur le système physiquement distribué. Les améliorations ajoutées présentent certains inconvénients. En effet, l'envoi de solutions courantes à l'agent SA augmente considérablement le temps CPU de l'application embarquée et le trafic entre les composants IoT. Ce temps étant critique ainsi que le coût de communication l'un des plus importants défis à surmonter lors de la conception d'une application IoT industrielle.

Ce travail discute premièrement l'utilisation de système embarqué dans le déploiement d'un processus d'optimisation intelligent et coopératif (dans notre cas la méta-heuristique PSO) et il traite en second lieu l'émergence de la technologie IoT dans les ateliers de production principalement dans la résolution du problème d'ordonnancement sous perturbations des machines.

Pour surmonter les limitations observées dans l'architecture MAPSO2 +, une version améliorée de l'architecture PSO multi-agent, nommée MAPSO2++, est proposée [NJA+16]. La section suivante décrira les améliorations apportées.

3.10 L'architecture MAPSO2++ :

L'agent BA et l'agent SA étants développés pour être déployés dans une machine serveur n'ayant pas des ressources limitées, nous proposons de mieux exploiter cette ressource en ajoutant de nouveaux comportements de contrôle à ces agents.

D'une manière générale, l'information et les connaissances obtenues à partir du réseau sensoriel seront utilisées pour alimenter d'autres processus, comme la prise de décision. Par conséquent, des boucles de contrôle de rétroaction hiérarchiques complexes peuvent être créées sur la base des données sensorielles obtenues.
Ainsi, dans ce travail, nous proposons d'ajouter un capteur de température / humidité dans chaque système embarqué désigné comme étant un conteneur de l'agent exécutant. Les données collectées à partir de ce capteur sont utilisées tout d'abord pour prédire la panne de l'unité de contrôle et deuxièmement pour ajuster et contrôler l'échange de données entre les agents exécutants et l'agent de synchronisation. La figure 3.7 montre la nouvelle architecture MAPSO2 ++ améliorée proposée.

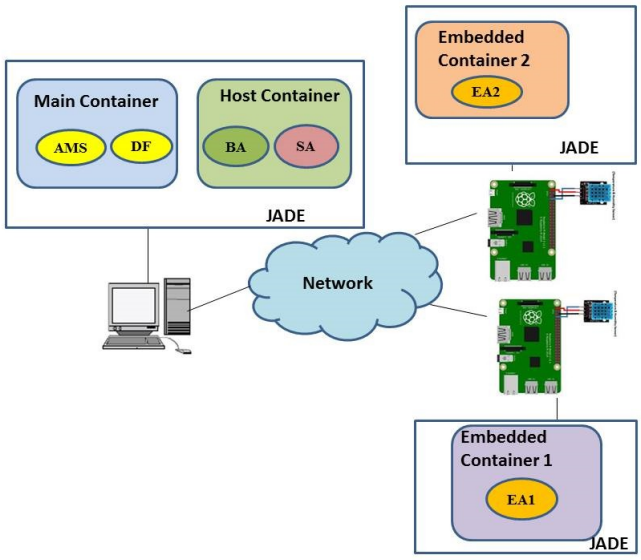

FIGURE 3.7: L'architecture MAPSO2++

En comparaison à l'architecture précédente MAPSO2+, l'agent EA se comporte de la même manière mais sans la phase d'envoi des solutions provisoires. De nouveaux comportements de contrôle sont ajoutés au rôle d'EA. Le comportement des agents SA et BA est aussi identique par rapport à l'ancienne architecture en ajoutant seulement de nouveaux comportements périodiques de contrôle.
Les rôles ajoutés pour chaque agent sont les suivants :

- **Agent Exécutant (EA) :**

  Trois nouveaux rôles sont ajoutés à l'EA :

  — Un comportement cyclique pour lire la température du capteur pendant une période spécifique $p1$ (voir algorithme 6). Selon la valeur trouvée, si la température est supérieure au seuil fixe, cela indique que l'unité va tomber en panne. Dans

ce cas, l'agent prépare la migration des meilleures solutions à l'agent de synchronisation. La figure 3.8 illustre deux exemples de variation de la température au cours du temps. La période $p1$ représente la fréquence de lecture de l'indicateur de panne (Température dans notre cas). Selon la température trouvée, la variable *Decision* est mise à jour (alarm ou false alarm).

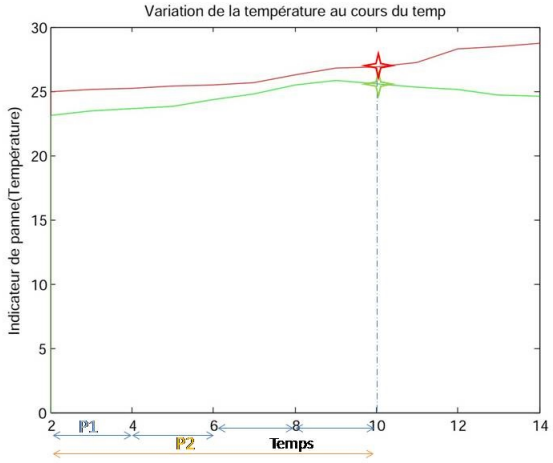

FIGURE 3.8: La variation de la température au cours du temps.

— Le deuxième comportement ajouté consiste à vérifier s'il y a une fausse alarme ou non (voir l'algorithme 7). En effet la période $p2$ de ce comportement étant plus grande que $p1$, l'agent va vérifier que la température croit progressivement pour lancer la migration réelle. En d'autres termes, la période $p2$ représente une fréquence plus englobante pour décider l'envoi des meilleures particules (si la variable *Decision*="alarm") ou l'envoi d'un simple message "No breakdown" à l'agent de synchronisation. La courbe en rouge dans la figure 3.8 illustre un exemple d'une vraie alarme (augmentaion continue de la température) et celle en vert illustre le cas d'une fausse alarme.

  La configuration des périodes peut être en fonction du type de capteur, de l'état du capteur, du type d'application (temps réel ou non), etc.

— Le dernier rôle est de vérifier l'état de l'agent exécutant distant de SA (voir l'algorithme 8). C'est un comportement cyclique de période spécifique $p3$ pour vérifier des pannes externes de la part de l'agent de synchronisation. Il peut alors recevoir un message indiquant qu'aucune panne n'a été détectée sinon il reçoit les solutions migrantes de l'agent en panne. Dans ce dernier cas, l'AE va exécuter l'algorithme PSO sur le nouvel essaim qui combine une partie de ces propres particules et les particules migrantes.

Algorithme 6 Lecture de la température (période p1) du capteur par AE

1: read temperature T1 from sensor
2: If ($T1 \geq fixedTemprature$) then
$Decision = "alarm"$
Select bestparticles found by EA
Message.setcontentObject(bestparticles)
Else
$Decision = "falsealarm"$

Algorithme 7 Vérification Alarm(période p2) par AE

1: If (Decision= "alarm") then
Send (Message(bestparticles), SA)
Else
Send ("no breakdon", SA)

Algorithme 8 Algorithme de contrôle(période p3) par AE

1: receive Controlmessage from SA agent
If (Controlmessgae.getcontent()== "no breakdown") then
continue RunPso(swarm1)
Else newSolutions= Controlmessage.getContentObject()
$swarm2 = 50\%swarm1 + newSolutions$
RunPso(swarm 2)

- **Boss Agent (BA) :**

  Le rôle ajouté à cet agent est de vérifier le message reçu de l'agent SA. S'il reçoit un message "no breakdown", il continue à fonctionner normalement sans interruption. Sinon, il enregistre une copie des meilleures particules trouvées par l'EA avant son échec et il les ajoute à l'essaim "currentgbestswarm" (Voir algorithm 9).

Algorithme 9 Algorithme de contrôle(période p3) par BA

1: receive message from SA agent
If (messgae2.getcontent()== "no breakdown") then
no interruption
Else bestparticles= message2.getContentObject()
add (bestparticles, CurruntgbestSwarm)

- **Agent de sychronisation (AS) :**

  Le rôle ajouté à cet agent est de vérifier le message reçu de la part de l'agent exécutant. S'il reçoit le message "no breakdown", il le diffuse à tous les autres agents de l'architecture (BA et EA) en les informant qu'aucune panne n'est détectée. Sinon, il envoie une copie des meilleures solutions de l'EA en échec à l'autre agent EA en vie et une autre copie à l'agent BA.

La copie de l'essaim envoyé à l'agent patron va servir si la panne inclut les deux agents exécutants. Ainsi, dans ce cas de figure, l'agent patron va ajouter ces particules à l'essaim des solutions courantes "*currentgbestParticules*". L'algorithme 10 illustre le nouveau comportement de SA.

Algorithme 10 Algorithme de contrôle(période p3) par AS

```
1: receive message from EA agent
   If (messgae1.getcontent()== "no breakdown") then
        Send ("no breakdown", BA)
        Send ("no breakdown", EA)
   Else bestparticles= message1.getContentObject()
        message2.setContentObject( bestparticles)
        Send ( message2, EA)
        Send (message2, BA)
   End If
```

3.11 Expérimentation et comparaison entre MAPSO2+ et MAPSO2++

Pour tester l'efficacité et la performance du MAPSO2 ++, nous avons effectué des expériences sur les trois benchmarks FJSP suivants : deux instances avec une flexibilité totale (4 * 5 et 10 * 10) et une avec une flexibilité partielle (8 * 8). Chaque instance est exécutée 20 fois. La simulation de l'architecture MAPSO2 + et MAPSO2 ++ a été faite sur le même système distribué utilisé pour MAPSO2. Ce système est composé d'un PC et deux systèmes embarqués ARM A7. Le BA et le SA sont implémentés sur la machine serveur et chaque agent exécutant est implémenté sur un système embarqué cible qui est connecté cette fois à un capteur de température / humidité. Un LAN est utilisé pour la communication entre tous les composants du système distribué. Dans ce travail, tout le développement est fait en Java. Nous avons utilisé la machine virtuelle SE Embedded (JVM) dédiée aux périphériques moyens intégrés. La figure 3.9 illustre le système physique distribué utilisé.

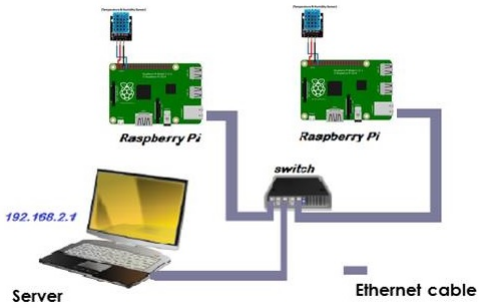

FIGURE 3.9: Le système distribué utilisé pour implémenter MAPSO2++

Le tableau 3.6 présente les résultats de simulation (makespan et temps de traitement CPU) du MAPSO2 + et du MAPSO2 ++ des trois instances FJSP. Nous remarquons que les valeurs de makespan trouvées sont les mêmes pour les instances (8 * 8) et (10 * 10). Cependant, il existe une différence entre les valeurs de makespan de la meilleure particule pour l'instance (4 * 5). Comme nous avons précedemment examiné dans la sous-section 3.8.1, il est important de diversifier chaque sous-population pour chaque EA afin d'avoir le même espace de recherche varié par toutes les méthodes d'initialisation.

TABLE 3.6: Compraison entre MAPSO2+ et MAPSO2++ (MAX-ITERATION=200 ; SWARM SIZE=200)

Instance	MAPSO2+		MAPSO2++	
	Makespan	Temps CPU (ms)	Makespan	Temps CPU(ms)
4*5	12	46819	11	**28819**
10*10	8	64732	8	**46732**
8*8	17	58654	17	**48654**

Lors de la comparaison du temps CPU, les résultats montrent que le temps d'exécution de MAPSO2 + est plus lent que le MAPSO2 ++ proposé tout en utilisant le même système distribué. Comme nous pouvons le constater dans le tableau 3.6, le MAPSO2++ distribué nécessite moins de temps que le MAPSO2+ proposé. Le temps CPU passe de 46819 ms à 28819 ms pour l'instance FJSP 4*5, de 64732 ms à 46732 ms pour l'instance 10*10 et de 58654 ms à 48654 ms pour l'instance 8*8. Ce gain est justifié par la réduction du taux d'échange entre l'AE et l'AS.

Dans l'architecture MAPSO2 +, la variable $fixeditération$ dans l'algorithme de l'agent exécutant définit quand l'EA envoie à l'agent SA les meilleures particules courantes *currentgbestParticle*. Comme nous l'avons constaté dans les expérimentations précédentes de l'architecture MAPSO2+, la configuration de cette variable est $fixeditération = 3$. Chaque EA envoie la meilleure particule courante *currentgbestParticle* à l'agent SA après

chaque 03 itérations. La valeur de cette variable affecte le nombre de messages transmis et le temps CPU de l'application.
Afin de comparer le trafic entre EA et SA dans les deux architectures, nous configurons la valeur de période $p1$ du comportement de l'EA "lecture de la température" à la même valeur de $fixeditération$. Cependant, dans l'architecture MAPSO2++, l'EA envoie ses meilleures particules à l'agent synchroniseur seulement après avoir vérifié l'état du système pour éliminer les fausses alertes. Pour atteindre cet objectif, nous configurons la période de ce comportement cyclique (vérification d'alarme) par une valeur plus grande que la valeur de la période $p1$.

TABLE 3.7: Le nombre de messages envoyés à l'agent Sychroniseur SA

Etat du system	MAPSO2+	MAPSO2++
sans perturbation	Max_iteration/fixediteration Messages	**0 message**
avec perturbation	Max_iteration/fixediteration Messages	**1 message**

Le tableau 3.7 illustre le nombre de messages envoyés par l'EA aux SA dans MAPSO2 + et MAPSO2 ++ lors de la résolution de FJSP respectivement avec ou sans perturbations.
Comme le montre le tableau 3.7, le MAPSO2 + a un grand trafic de messages avec et sans interruption. Cependant, le MAPSO2 ++ proposé réduit cette valeur des messages de $Max_{iteration}/fixediteration$ à un seul message après avoir vérifié l'état du périphérique incorporé. Si l'on considère que l'architecture distribuée contient n EA selon le nombre d'unités de contrôle dans le système de fabrication, le nombre de messages envoyés au SA par tous les agents EA est égal à $n * (Max_{iteration}/fixediteration)$.
Le MAPSO2++ proposé nous permet de réduire la communication inter-composants IoT et par conséquent la minimisation de la consommation d'énergie et l'accroissement de l'autonomie des dispositifs IoT intégrés.

3.12 Conclusion

Dans ce chapitre, nous avons discuté l'ajout des systèmes embarqués et les capteurs dans le déploiement d'une méthode d'optimisation comme le PSO pour résoudre le problème d'ordonnancement FJSP sans et avec incertitudes.
Nous avons distribué l'architecture MAPSO2 sur un système physiquement distribué com-

posé d'un serveur et deux systèmes embarqués ARM A7. Une étude comparative du temps d'exécution et de la qualité de solution entre MAPSO2 centralisé et MAPSO2 distribué a été faite.

Nous avons proposé une architecture MAPSO2+ afin de surmonter les limites de l'architecture MAPSO2 pour plus de flexibilité et d'adaptabilité pour les environnements dynamiques. Néanmoins, après avoir déployé MAPSO2+ sur l'architecture réellement distribuée, nous avons proposé d'intégrer les capteurs de températures et ajouter de nouveaux comportements de contrôle aux agents pour minimiser le temps d'exécution en réduisant la communication inter-composant IoT. La nouvelle architecture est nommée MAPSO2++. L'ajout de capteur dans le système embarqué nous a permis de prédire la rupture et d'ajuster la communication entre les agents en fonction de l'état de l'unité de contrôle. Les expériences montrent que le MAPSO2 ++ proposé offre de meilleurs résultats en terme de temps CPU et de qualité de solution. Le trafic entre les composants IoT est également réduit.

Nous avons proposé aussi une approche permettant d'améliorer le temps d'exécution en jouant sur la configuration de la taille de sous essaim délégué à l'agent exécutant intégré dans chaque système embarqué distant. Cette amélioration augmente la flexibilité de l'architecture même en cas de coupure de communication car l'agent patron BA va exécuter PSO indépendamment de secours des autres agents exécutants.

Bien que l'architecture développée est plus flexible lors de la survenue d'une panne, il est primordial d'utiliser une technique de réparation de l'ordonnancement affecté par la panne. Dans le chapitre suivant, nous proposons une technique de ré-ordonnancement ainsi qu'un algorithme PSO pour résoudre FJSP avec la prise en compte des pannes machines.

Chapitre 4

Particle swarm optimization pour l'ordonnancement robuste : cas de FJSP

4.1 Introduction

Dans ce chapitre, nous présenterons une nouvelle approche basée sur l'algorithme d'optimisation par essaims particulaires pour la résolution du problème d'ordonnancement du job shop flexible sous incertitudes. Dans notre travail, les incertitudes portent sur les pannes aléatoires des machines. L'objectif est donc de proposer un algorithme d'optimisation par essaims particulaires permettant de fournir une solution de bonne qualité mais également robuste et peu sensible aux perturbations.
A cet effet, il est nécessaire de développer un algorithme efficace pour avoir un ordonnancement robuste. Nous décrivons comment l'algorithme d'optimisation par essaims particulaires pour l'ordonnancement déterministe, présenté dans le chapitre précédent, peut être modifié pour trouver des solutions pas seulement de bonne qualité (makespan minimum) mais aussi robustes face aux perturbations des machines.
Dans ce qui suit, nous présentons une description détaillée de l'algorithme proposé pour l'ordonnancement robuste 2s-PSO. Nous détaillerons les fonctions objectives utilisées ainsi que la simulation de panne. Un exemple illustratif expliquant bien le fonctionnement de l'algorithme est aussi présenté. Nous décrivons aussi l'heuristique de ré-ordonnancement proposée pour réparer l'ordonnancement affecté par les pannes.

4.2 Description détaillée de l'algorithme 2s-PSO

Plusieurs méta-heuristiques ont été proposées et adaptées afin de résoudre le problème de FJSP sous incertitudes.
Une stratégie commune pour la résolution du problème d'ordonnancement prédictif en te-

nant compte des possibles défaillances de machines, consiste à insérer à l'avance un temps d'inactivité approprié dans l'ordonnancement prédictif. Ceci est réalisé en premier temps, en générant un ordonnancement initial supposant l'absence de panne, puis un temps d'inactivité est inséré avant chaque tâche de sorte que les dates de début et de fin de l'opération du job en question changent tout en respectant l'enchainement des jobs. Ces temps d'inactivité servent de tampon temporel pour absorber l'effet des perturbations.

Cependant, cette méthode présente deux difficultés : la recherche de la quantité de temps d'inactivité à insérer et la découverte des emplacements appropriés pour y insérer ces temps d'inactivité.

Notre approche fait partie de la famille des méthodes prédictives de l'ordonnancement sous incertitudes. Nous proposons une méthode d'insertion de temps non-inactif en utilisant l'algorithme d'optimisation par essaims particulaires.

En effet, nous optons pour le choix de cette méta-heuristique car après recherche, on n'a trouvé aucun travail se basant sur l'algorithme PSO permettant de réaliser un ordonnancement préventif robuste et à notre connaissance il n'y a pas non plus d'études comparatives entre les différents ordonnancements prédictifs proposés à l'aide des différentes méta-heuristiques.

Dans ce travail, nous étendons l'algorithme PSO proposé pour résoudre le problème de job shop flexible statique et proposons une approche prédictive basée sur le même algorithme pour résoudre le problème FJSP dynamique sous incertitudes. La méthode proposée a pour objectif de générer un ordonnancement préventif qui minimise la dégradation de performance en cas de pannes des machines. Nous effectuons aussi une étude comparative entre notre approche et une approche prédictive basée sur l'algorithme génétique proposé par [AHE11] tout en considérant le même type d'incertitude. Une analyse de variance entre les résultats de deux méthodes est enfin effectuée.

L'idée globale de notre approche préventive est d'intégrer simultanément la connaissance de la distribution de probabilité de panne de machine tout en exploitant la flexibilité de routage des machines disponibles [AHE11]. De ce fait, le programme prédictif sera capable d'assigner et de séquencer les opérations sur les machines afin de minimiser l'impact d'une panne sur les performances globales de l'ordonnancement. Cet impact sera évalué en fonction des perturbations ayant affecté la qualité de l'ordonnancement préventif. Dans notre cas, la mesure sera la dégradation de la valeur du makespan trouvé dans l'ordonnancement préventif par rapport à celle de l'ordonnancement actuel, ainsi que la stabilité de l'ordonnancement.

Notre approche nommé "two stage PSO" (2s-PSO) [NBJ+17] comporte deux phases : la pre-

mière consiste à exécuter l'algorithme d'optimisation par essaim particulaire en supposant que l'environnement est déterministe et en ignorant les perturbations et la deuxième consiste à exécuter le PSO en intégrant les probabilités de pannes de machine. La figure 4.1 montre l'organigramme général de notre algorithme. La représentation des particules est la même dans les deux phases de l'algorithme proposé.

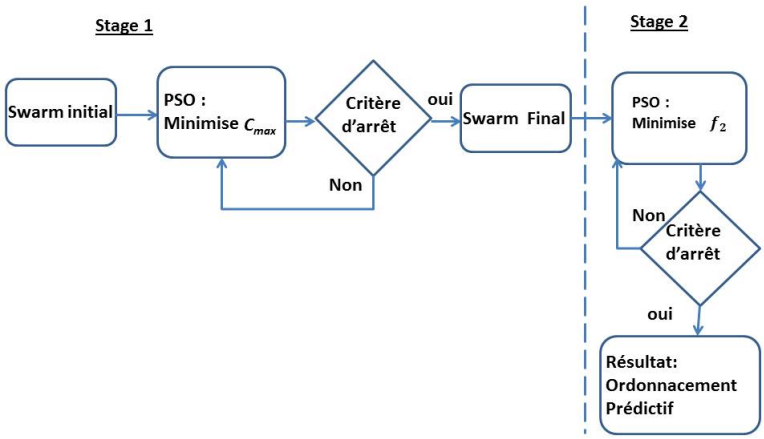

FIGURE 4.1: Organigramme de l'algorithme 2s-PSO

Comme illustré dans la figure précédente, durant la première phase, l'essaim est évalué selon l'objectif de minimisation de la valeur du makespan C_{max}. Après un certain nombre de générations, le système converge vers un essaim constitué par les meilleures solutions trouvées ayant le minimum makespan. Cette population trouvée par le stage 1 sera considérée comme population initiale de la deuxième phase de l'algorithme. Ensuite, l'algorithme continue à évoluer mais la population sera évaluée cette fois par une fonction bi-objective basée à la fois sur le makespan, la robustesse et la stabilité tout en intégrant la probabilité de pannes de machine dans la procédure de codage.

En fait, l'historique d'un atelier de production peut fournir une répartition approximative des pannes des machines. Ainsi cette distribution sur les incertitudes peut être utilisée pour générer l'ordonnancement prédictif [AHE11].

En effet, dans notre approche, nous intégrons les probabilités de pannes pour perturber la solution prédictive, et nous utilisons la mesure de la robustesse et de la stabilité pour évaluer l'effet des perturbations sur la solution. Cela nous amènera à obtenir des solutions plus robustes et plus stables.

Durant la deuxième phase de l'approche proposée, et à chaque génération, toutes les parti-

cules seront évaluées sur un ensemble de scénarios perturbés (c'est-à-dire que la population courante et les nouvelles particules trouvées sont évaluées sur les mêmes scénarios perturbés). Cet ensemble de scénarios change d'une génération à une autre.
Ce mécanisme permet d'augmenter la probabilité d'obtenir une solution robuste sur un nombre important de scénarios perturbés. Si par exemple le nombre d'itérations maximales de l'algorithme PSO est $max_{iteration} = 100$ et le nombre de scénarios perturbés est $N = 10$, alors la solution robuste est évaluée sur $100 * 10 = 1000$ scénarios perturbés. Chaque particule peut être aussi évaluée sur son propre ensemble de scénarios perturbés qui change d'une génération à une autre. Cela permettra de sélectionner des solutions plus robustes et plus stables.
La mesure de la robustesse et de la stabilité sera utilisée pour évaluer la solution prédictive perturbée. Dans ce qui suit nous détaillerons la fonction utilisée afin d'évaluer la robustesse des nouvelles particules et le modèle utilisé pour représenter les pannes de machines et la méthode de réordonnancement utilisée.

4.3 La Fonction objective d'évaluation

L'optimisation est exprimée par une fonction devant minimiser ou maximiser un critère donné. La plupart des travaux de recherche ont porté sur une seule optimisation objective, comme la minimisation du retard du job (Tardness) , la réduction de la charge de travail totale, la minimisation du makespan, etc.
En présence d'incertitude associée à la défaillance des machines, les ordonnancements devraient fonctionner dans des environnements dynamiques. Ainsi, l'ordonnancement doit satisfaire deux objectifs : avoir une valeur minimale de makespan et être en même temps stable et robuste face aux perturbations des machines.

4.3.1 Minimisation du makespan

Dans ce travail, notre premier objectif étant la minimisation du makespan (C_{max}), qui est la date de fin d'exécution de la dernière tâche. Il s'agit de trouver l'ordonnancement qui a le plus petit temps total d'achèvement. La fonction objective est la suivante :

$$C_{max} = max t_{ij} \tag{4.1}$$

Avec t_{ij} est le temps d'achèvement de l'opération O_{ij}.

4.3.2 Les mesures des stabilités

Plusieurs mesures de la stabilité et de la robustesse ont été proposées pour le problème de job shop flexible avec incertitudes. [AHE11] ont proposé trois mesures de stabilité, la suivante est la mieux adaptée. Elle est définie comme suit :

$$S = min \sum_{i=1}^{n} \sum_{j=1}^{q_i} (|t_{ijP} - t_{ijR}|) \tag{4.2}$$

Avec

-n est le nombre de jobs,

-q_i est le nombre des opérations de chaque job i,

-t_{ijP} est le temps prévu de fin de traitement de l'opération j du job i,

-t_{ijR} est le temps réalisé pour achever l'opération j du job i.

-R réfère à l'ordonnancement réalisé ou actuel tout en considérant les pannes de machines.

-P réfère à l'ordonnancement prévu ou l'originale.

Dans ce cas, la stabilité d'un ordonnancement prédictif ou original est mesurée par la somme des écarts absolus des temps d'achèvement des jobs entre l'ordonnancement réalisé après la panne et l'ordonnancement prédictif.

4.3.3 La fonction Bi-objective

Lors de la génération d'un ordonnancement prédictif, la plupart des travaux précédents considéraient souvent la robustesse ou bien la stabilité. [AHE11] tiennent compte à la fois de la robustesse et de la stabilité lors de la génération de la solution en utilisant une approche introduite par [LGX07] pour résoudre un problème de machine unique (single machine) avec la prise en compte des pannes aléatoires de machines.

La robustesse d'un ordonnancement est liée au degré de sa dégradation en cas de perturbations. L'ordonnancement est considéré comme stable lorsque l'ensemble de tous les écarts absolus de ses temps d'achèvement des opérations de l'ordonnancement réalisé est faible. La robustesse et la stabilité d'un programme sont étudiées selon une approche bi-objective.

La fonction bi-objective est la suivante :

$$f_2 = (\gamma minMS_R) + (1 - \gamma)S \tag{4.3}$$

Avec :

-MS_R est le makespan de l'ordonnancement réalisé après la panne.

-S est la mesure de stabilité utilisée.

-γ est un facteur de pondération entre [0,1] fixé par le décideur et qui reflète l'importance qu'il accorde à chacun des objectifs (MS_R and S).
Cette fonction est utilisée lors du second stage de l'algorithme.
Afin de prouver l'efficacité de l'algorithme proposé, nous le comparerons à d'autres méthodes utilisant différentes fonctions bi-objectives :

— Premièrement, la mesure de robustesse connue sous le terme "slack-time based robustness measure" est utilisée. Cette mesure est proposée par [JLDWS94]. Elle est donnée par l'équation suivante :

$$f_3 = MS_{min} - RD(sc) \tag{4.4}$$

avec :
-MS_{min} est le makespan de l'ordonnancement(sc).
-$RD(sc)$ est la moyenne de marge dans l'ordonnancement(sc).

— Deuxièmement, la mesure de robustesse de voisinage connue sous le terme "the neighborhood-based robustness measure", proposée par [Mat96], est utilisée. Elle est donnée par l'équation suivante :

$$f_4 = \frac{1}{|N_1(sc)|} \sum\nolimits_{sc' \in N_1(sc)} (MS_{min}(sc^{'})) \tag{4.5}$$

avec :
- $N_1(sc)$ est l'ensemble de voisinage de l'ordonnancement (sc) qui contient tous les ordonnancement faisables crée de (sc) en changeant deux opérations consécutives dans la machine.
- $MS_{min}(sc^{'})$ est le makespan de l'ordonnancement$(sc^{'})$.
- $(sc^{'})$ est un ordonnancement qui appartient à l'ensemble de voisinage.

Le nouvel ordonnancement réalisé après avoir intégré les pannes machines est calculé par une méthode spécifique de ré-ordonnancement. Nous présenterons ultérieurement la technique de ré-ordonnancement utilisée dans notre approche. Dans ce qui suit, nous détaillerons la méthode utilisée pour simuler les pannes de machines.

4.4 La simulation des pannes

Afin de résoudre le problème d'ordonnancement sous incertitudes, nous avons besoin de définir exactement comment les pannes peuvent être générées et modélisées. La forme la plus évidente de panne dans un atelier de production est la défaillance d'une machine la rendant inutilisable et indisponible pendant un certain temps. Nous supposons que des informations

sur l'incertitude de pannes de machines est disponible à l'avance, et peut être quantifiées par certaines distributions.

Simuler une panne comprend le choix de la machine défaillante, l'instant de panne et le moment où elle sera à nouveau opérationnelle.

Dans ce cas, trois paramètres sont nécessaires :

- Choix de la machines en panne : la plupart des travaux antérieurs choisissaient au hasard une machine. Dans notre travail nous supposons que les probabilités de pannes de machines sont liées à la charge de travail. Alors dans ce cas, le choix de la machine est en fonction de son temps de traitement occupé. En effet, le temps maximal d'occupation d'une machine est égal à sa charge de travail. Une machine ayant une lourde charge de travail est la plus susceptible de tomber en panne. La probabilité de la machine Mk en échec est approchée par la relation empirique suivante :

$$\rho_k = \frac{MBT_K}{MBT_{tot}} \tag{4.6}$$

  avec : MBT_k est la charge de la machine k et MBT_{tot} est la charge totale de toutes les machines.

- Le temps de panne.
- La durée de réparation.

Les deux paramètres précédents sont générés en utilisant les distributions uniformes suivantes respectivement :

$$t_k = [\alpha_1 MBT_K, \alpha_2 MBT_k] \tag{4.7}$$

$$t_{k,duration} = [\beta_1 MBT_K, \beta_2 MBT_k] \tag{4.8}$$

Où t_k est le temps de panne de la machine k, MBT_k est le temps occupé par la machine k, $t_{k,duration}$ est la durée de panne de la machine k et α et β varient selon le type de pannes.

Nous supposons deux niveaux de panne machine, le bas niveau et le haut niveau, et deux intervalles de temps de panne machine, précoces et tardives. Cela conduit à une combinaison de quatre paramètres de pannes données dans le tableau 4.1. Pour chaque type de panne, les paramètres α et β sont réglés en conséquence.

TABLE 4.1: Les niveaux des pannes

Type de panne	Niveau de perturbation	α_1	α_2	β_1	β_2
BD1	faible, tôt	0	0.5	0.1	0.15
BD2	faible, tard	0.5	1	0.1	0.15
BD3	élevée, tôt	0	0.5	0.35	0.4
BD4	élevée, tard	0.5	1	0.35	0.4

Si les valeurs de β entre 0.1 et 0.15, le niveau de panne est relativement faible, tandis que l'augmentation de ces valeurs (entre 0.35 et 0.4) augmentera la durée de la panne. De même, en limitant les valeurs de α entre 0 et 0.5, nous garantissons que la rupture se produit au cours de la première moitié de l'ordonnancement, par contre, en modifiant ces valeurs entre 0.5 et 1 nous assurons que la machine tombe en panne tardivement c'est-à-dire elle se produit au cours de la seconde moitié de l'ordonnancement.

4.5 La technique de Ré-ordonnancement utilisée

Comme nous avons mentionnée dans le premier chapitre, il existe plusieurs méthodes de ré-ordonnancement dans la littérature. Pour une raison de comparaison, nous utilisons dans ce travail la même méthode de ré-ordonnancement utilisée par [AHE11] qui est nommée "Modified Affected operation Rescheduling" (mAOR) proposé par [SR03].
Cette méthode utilise le même concept principal que la méthode AOR (Affected Operation rescheduling) mais prend compte des différents types de facteurs de ré-ordonnancement autres que la panne de la machine (la variation du temps de traitement d'une opération, l'arrivée d'un job inattendu, un job urgent, etc).
Le principe de cette méthode est de réaffecter les opérations directement ou indirectement affectées par la panne de machine tout en gardant le même enchainement des opérations sur chaque machine comme dans l'ordonnancement initial [AS97].
En exécutant le mAOR, la séquence des opérations reste la même que celle dans l'ordonnancement initial prédictif. Ceci a pour objectif de réduire l'effet et les coûts engendrés par toute sorte de déviations de séquence.
Seules les opérations directement et indirectement affectées sont poussées dans le temps pour tenir compte de la perturbation. Ainsi, le temps d'achèvement des opérations dans le nouvel ordonnancement réalisé après la survenue de la défaillance de la machine peut être calculé comme suit :

$$\mathrm{t}_{ijR} = \begin{cases} SO_{ijR} + p_{ijk} & \texttt{for unaffected operations,or} \\ SO_{ijR} + p_{ijk} + \tau_{k,duration} & \texttt{for affected operations} \end{cases} \tag{4.9}$$

avec :

-p_{ijk} est le processing time de l'opération j du job i on machine k,

-$\tau_{k,duration}$ est la durée aggregé de panne de machine k ;

-SO_{ijR} est la date de début de traitement de l'opération j du job i et il est calculé comme suit :

$$SO_{ijR} = max(t_{i(j-1)R}, ti_k) \tag{4.10}$$

où ti_k est le temps de disponibilité de la machine k.
Dans la section suivante, nous illustrons notre algorithme par un exemple réel du problème job shop flexible.

4.6 Exemple illustratif

Pour illustrer notre approche, nous considérons un exemple pour la résolution du problème du job shop flexible avec quatre job et cinq machines (4*5). Les diagrammes de Gantt représentés dans les figures 4.3, 4.5, 4.6 et 4.7 sont obtenus par l'algorithme proposé d'optimisation des essaims de particules à deux phases (2S-PSO). La figure 4.2 et la figure 4.4 montrent deux solutions possibles au problème trouvé par l'algorithme PSO qui minimise seulement le makespan [NJA+13].
Lorsqu'on considère le problème FJSP déterministe ayant comme fonction objective la minimisation de la valeur makespan, il n'y a aucune préférence dans le choix d'un ordonnancement devant être sélectionné comme solution globale car les deux ont la même valeur du makespan qui est égale à 13 (voir la figure 4.2 et la figure 4.4)
Toutefois, lorsqu'on envisage un FJSP en cas de perturbations, le PSO en deux phases proposé (2s-PSO), choisit l'ordonnancement le plus robuste et le plus stable. En effet, lorsque la probabilité de panne de la machine est intégrée dans le problème, on peut sélectionner un ordonnancement pouvant absorber l'impact de la future défaillance de la machine.
Par exemple, les données historiques indiquent que la machine $M5$ a un risque élevé de défaillance au vu de sa charge élevée de travail. Comme le montre les figures 4.3 et 4.5, les deux ordonnancements prédictifs initiaux, présentés dans les figures 4.2 et 4.4, sont soumis à la même panne spécifiée par un rectangle jaune. L'occupation de panne machine est tôt(type de panne est BD3).
L'algorithme 2s-PSO proposé évalue l'effet des perturbations sur la solution tout en utilisant la mesure de la robustesse et de la stabilité. En effet, l'ordonnancement illustré dans la figure 4.2 comporte trois opérations affectées : deux affectées de façon directe (O_{11}, O_{02}) situées sur la machine défaillante $M5$ et une autre indirectement O_{12} liées à la contrainte de précédence entre les opérations de la même tâche 1 (voir la figure 4.3). Le makespan de l'ordonnancement réalisé après la mise en oeuvre de la méthode de ré-ordonnancement est

égal à 15.

L'ordonnancement dans la figure 4.5 comporte six opérations affectées par la panne : deux directement affectées (O_{11}, O_{22}) et quatre non dirigées O_{12}, O_{30}, O_{31}, O_{23}. La valeur du makepan MS_r obtenu, après avoir appliqué la méthode de ré-ordonnancement, est égale à 16.

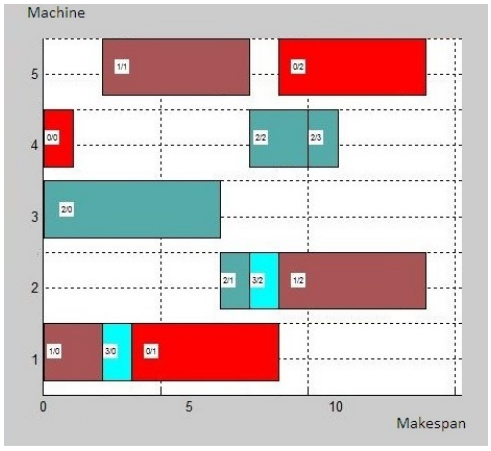

FIGURE 4.2: Exemple numéro 1 d'un ordonnancement prédictif obtenu par PSO minimisant Cmax

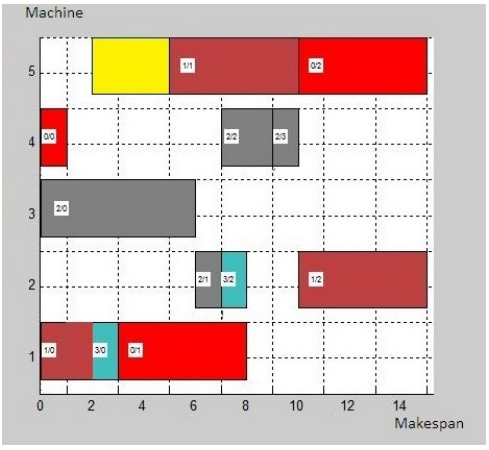

FIGURE 4.3: Exemple numéro 1 aprés le ré-ordonnancement (Panne BD3 marquée en jaune) par 2s-PSO

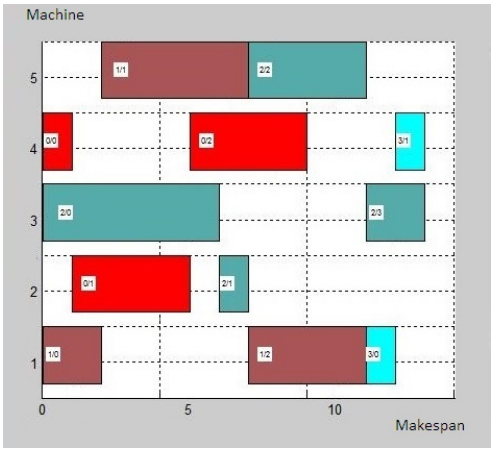

FIGURE 4.4: Exemple numéro 2 d'un ordonnancement prédictif obtenu par PSO minimisant Cmax

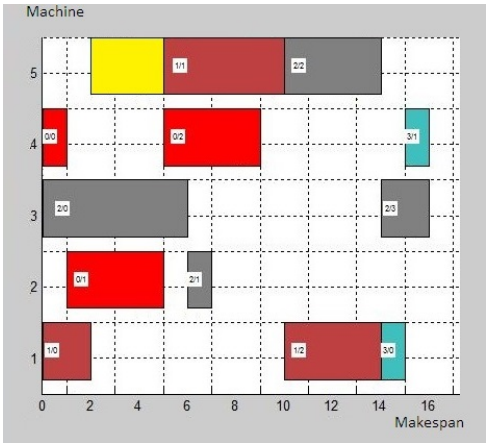

FIGURE 4.5: Exemple numéro 2 aprés le ré-ordonnancement (Panne BD3 marquée en jaune) par 2s-PSO

Ainsi, l'ordonnancement de la figure 4.2 est plus approprié pour être sélectionné par l'approche proposée 2s-PSO. En effet, cette solution est plus robuste que l'ordonnancement présenté dans la figure 4.4 parce que la dégradation de la performance sous perturbations est minimale. Cette solution est aussi plus stable vu le nombre limité d'opérations perturbées. Par conséquent, la somme des écarts absolus des temps de fin d'exécution des opérations par rapport à l'ordonnancement réalisé est plus faible. Nous testons le même exemple sur

un autre type de panne. Les deux ordonnancements prédictifs initiaux, présentés dans les figures 4.2 et 4.4 , sont soumis à un autre type de panne BD4 spécifiée par un rectangle jaune. Ce type de panne s'est produit à un stade avancé.
La figure 4.6 et la figure 4.7 montrent l'ordonnancement réalisé après le ré-ordonnancement.

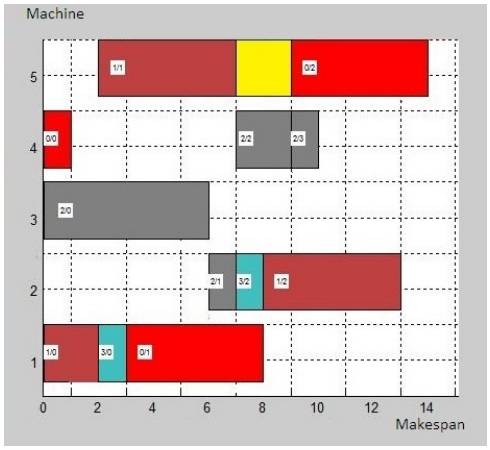

FIGURE 4.6: Exemple numéro 1 aprés le ré-ordonnancement (Panne BD4 marquée en jaune) par 2s-PSO

FIGURE 4.7: Exemple numéro 2 aprés le ré-ordonnancement (Panne BD4 marquée en jaune) par 2s-PSO

L'ordonnancement de la figure 4.6 est plus robuste et plus stable que celui de la figure 4.7. En effet, son makespan (14) est meilleur par rapport à l'autre ordonnancement (15).
En conclusion, l'intégration de la probabilité de panne lors de la résolution de problème permet de perturber l'ordonnancement, il sera alors évalué par la valeur de la robustesse et de la stabilité.
Afin de valider l'efficacité et la performance de notre algorithme, nous l'avons testé sur plusieurs Benchmarks du problème d'ordonnancement du job shop flexible connu dans la littérature. Les expérimentations et l'analyse des résultats sont présentées dans la section suivante.

4.7 Expérimentations et analyse de perfomance de 2s-PSO

Dans cette section, nous présenterons les résultats des expérimentations de l'algorithme d'optimisation par essaim particulaire robuste 2s-PSO.
Afin de valider les performances de l'algorithme 2s-PSO, nous avons effectué des expériences avec différentes instances de benchmark mettant en oeuvre à la fois le cas de job shop totale T-FJSP et partiel P-FJSP. Les instances sont comme suit :

— Deux instances de type T-FJSP : Ex1 est de taille 10 * 10 et Ex2 est de taille 15 * 10.

— Onze instances de type P-FJSP : Ex3 est une instance de petite taille 8 * 8 et Mk1, Mk2, .., Mk10 sont instances de grandes tailles proposées par Brandimarte. Pour les benchmarks de Brandimarte, le nombre de job varie de 10 à 20, le nombre de machines de 6 à 15 et le nombre d'opérations de 58 à 232.

Notre algorithme a été implémenté avec l'IDE Netbeans 6.0. L'implémentation a été réalisée sur une machine doté d'un processeur Intel "Core2Duo" cadencé à 2,0 GHz avec 4 Go de mémoire. Ces expériences sont menées pour évaluer les performances de l'ordonnancement prédictif issu de notre approche 2S-PSO. Une comparaison entre les résultats de notre algorithme et celle de [AHE11] a été aussi effectuée. Le deux algorithmes **2S-PSO** et l'algorithme génétique hybride proposé par [AHE11] utilisent la même fonction objective définie dans l'équation 4.3.

De plus, nous avons comparé aussi les résultats de notre algorithme 2s-PSO aux ordonnancements prédictifs issus par l'algorithme génétique hybride HGA tout en optimisant les mesures de robustesse et de stabilité définies f_3 et f_4 respectivement par les équations 4.4 et 4.5.

Pour ce faire, chaque benchmark est soumis aux quatre types de pannes (BD) avec 5 réplications par instance par type de panne. Ensuite, chaque ordonnancement prédictif généré est soumis à 400 pannes de machine aléatoires, ce qui entraîne $4 * 5 * 400 = 8000$ test par instance.

Après plusieurs expérimentations, nous avons retenu les paramètres suivants pour l'algorithme d'optimisation par essaims particulaires dans le deux phases :

-Taille de la population ou l'essaim $swarmsize = 500$.

-Critère d'arrêt de l'algorithme 2S- PSO proposé est le nombre maximal d'itérations $Max_{Iteration} = 100$.

L'algorithme passe à la deuxième étape après 100 itérations aussi.

La valeur du paramètre de pondération γ dans la fonction bi-objective peut être n'importe quelle valeur dans [0,1]. Si nous cherchons un compromis entre l'efficacité et la robustesse d'ordonnancement, la valeur de γ dépendra du point de vue du décideur. Pour $\gamma = 1$, nous recommandons de minimiser le makespan de l'ordonnancement initial. Pour $\gamma = 0$, nous recommandons de minimiser l'écart entre le makespan du planning initial et celui réalisé après l'intégration de la rupture. Le tableau suivant illustre l'effet de variation du facteur de pondération γ sur la qualité de la solution trouvée (dans notre cas la valeur de makespan). Selon le tableau 4.2, obtenu en testant 2S-PSO pour des cas différents, plus la valeur de γ

diminue, plus la valeur du makespan de l'ordonnancement initial augmente avec diminution systématique de la déviation entre le makespan de tous les scénarios perturbés et le makespan du scénario initial.

Pour des raisons de comparaison, la valeur du paramètre γ dans la fonction bi-objective $f2$, définie par l'équation 4.3, est fixée à la même valeur utilisée dans l'article de référence [AHE11] qui est égale à 0.6.

TABLE 4.2: Les résultats en variant la valeur de γ

Instance	γ	Makespan
4*5	γ=1	11
	γ=0.5	16
	γ=0	20
10*10	γ=1	8
	γ=0.5	12
	γ=0	18
8*8	γ=1	17
	γ=0.5	20
	γ=0	26

4.7.1 Test et Comparaison des performances de l'algorithme 2s-PSO

Pour évaluer la performance de l'ordonnancemnt prédictif obtenu par notre algorithme, nous calculons d'abord le pourcentage moyen d'amélioration de la valeur du makespan de l'ordonnancement réalisé AMS_RI(Average realized makespan improvement percentage) défini par :

$$AMS_RI = \frac{\sum_{q=1}^{5}\sum_{p=1}^{400} RMS_R(q)_p - \sum_{q=1}^{5}\sum_{p=1}^{400} DMS_R(q)_p}{\sum_{q=1}^{5}\sum_{p=1}^{400} DMS_R(q)_p} * 100 \tag{4.11}$$

avec :

- RMS_R est le makespan réalisé de l'ordonnancement obtenu en utilisant la méthode robuste après la panne.

- DMS_R est le makespan de l'ordonnancement obtenu en utilisant la méthode déterministe après avoir simulé une panne machine. La méthode déterministe dans notre cas est le PSO initial statique qui minimise la valeur du makespan.

-q représente le nombre de réplication et p représente le nombre des pannes.

Nous calculons aussi le pourcentage moyen d'amélioration de la stabilité $ASTBI$ (Average Stability Improvement percentage) défini par l'équation suivante :

$$ASTBI = \frac{\sum_{q=1}^{5}\sum_{p=1}^{400} RSTB(q)_p - \sum_{q=1}^{5}\sum_{p=1}^{400} DSTB(q)_p}{\sum_{q=1}^{5}\sum_{p=1}^{400} DSTB(q)_p} * 100 \tag{4.12}$$

avec :

-RSTB est la stabilité de l'ordonnancement obtenu en utilisant la méthode robuste (dans notre cas 2s-PSO).

-DSTB est la stabilité de l'ordonnancement obtenu en utilisant la méthode déterministe.

-q représente le nombre de réplication et p représente le nombre des pannes.

Le tableau 4.3 montre les résultats trouvés en termes de pourcentage d'amélioration moyen du makespan réalisé et de pourcentage moyen d'amélioration de la stabilité en simulant des pannes de type BD1 et BD2. Les résultats obtenus pour les types de panne BD3 et BD4 sont illustrés sur le tableau 4.4.

Le tableau 4.3 et le tableau 4.4 se composent de 11 colonnes : la première et la deuxième colonne représentent le nom et la taille de l'instance. Le reste des colonnes représente le pourcentage AMS_RI et $ASTBI$ de chaque instance obtenu par notre algorithme en comparaison avec les algorithmes génétiques hybrides HGA1, HGA2 et HGA3 lorsqu'ils sont soumis à des types de pannes spécifiques. HGA1, HGA2 et HGA3 représentent l'algorithme génétique hybride proposés par [AHE11] en optimisant respectivement les mesures de robustesse et de stabilité f_2, f_3 et f_4.

Les valeurs négatives dans les tableaux indiquent qu'il y a des améliorations alors que les valeurs positives indiquent qu'il y a des dégradations lors de la comparaison des différentes méthodes.

Comme illustré dans le tableau 4.3 et le tableau 4.4, l'utilisation de notre algorithme proposé 2S-PSO améliore la stabilité et l'efficacité dans toutes les instantes testées par rapport à l'algorithme génétique hybride HGA3 en considérant les types de panne BD1, BD2 et BD4. Concernant le type de panne BD3, notre algorithme proposé donne des meilleurs résultats par rapport à l'algorithme HGA2 et HGA3 dans toutes les instances à l'exception de Mk01, MK03, MK04 et MK07. Cela signifie que la fonction bi-objective utilisée dans notre méthode proposée est meilleure que la mesure de robustesse basée sur le voisinage proposée par [Mat96] utilisée dans HGA3.

TABLE 4.3: Résultats des instances soumises à des pannes de type BD1 et BD2

			BD1				BD2			
Instance	taille		notre 2s-PSO	HGA1	HGA2	HGA3	notre 2s-PSO	HGA1	HGA2	HGA3
Ex1	10*10	AMS_RI	-6.97	-5.41	0	-2.7	-6.44	-5.41	0	-2.7
		$ASTBI$	-90.45	-100	50	-40	-82.5	-90,91	-63.64	-72.73
Ex2	15*10	AMS_RI	-8.55	-1.64	4.82	6.56	-10.41	3.28	4.92	6.56
		$ASTBI$	-98.16	-96.97	-3.56	9.09	-13.46	-58.26	11.3	32.17
Ex3	8*8	AMS_RI	-7.98	-1.35	0	1.35	-6.95	-4,37	-2.56	-2.56
		$ASTBI$	-39.03	-73.33	-20	-20	-37.105	-37.12	38.46	-30.77
MK01	10*6	AMS_RI	0.77	-4.7	-4,89	-3.17	4.80	0.91	2.87	4.78
		$ASTBI$	-13.26	-62.64	-40.64	-44.65	-16.65	-81.74	1.68	11.77
MK02	10*6	AMS_RI	-1.64	0	2	4.67	-9.79	0	-1.33	1.32
		$ASTBI$	-17.23	-55.61	18.56	15.81	-16.21	-5.89	51.9	13.92
MK03	15*8	AMS_RI	-2.24	-4.45	-5.45	-4.45	6.92	-7.27	-8.52	-7.18
		$ASTBI$	-26.95	-85.85	-20.12	-21.89	8.49	-66.79	-35.95	-50.55
MK04	15*8	AMS_RI	-6.46	0.29	4.39	3.51	3.72	0.3	3.25	4.14
		$ASTBI$	-15.58	-49.09	31.63	26.18	10.85	-32.63	-3.76	-11.68
MK05	15*4	AMS_RI	-3.56	-1.33	0.41	1.64	0.54	-1.05	-0.51	1.11
		$ASTBI$	-65.76	-61.02	5.81	-9.27	-4.23	-9.01	1.71	-1.81
MK06	10*15	AMS_RI	-2.45	-1.31	0.28	1.01	-2.65	0.25	-0.85	2.54
		$ASTBI$	-60.65	-58.78	-23.6	-29.68	-78.34	-42.61	-51.66	21.54
MK07	20*5	AMS_RI	-0.43	-1.87	0.79	2.53	1.32	-2.5	-0.13	3.87
		$ASTBI$	-64.21	-83.42	-8.32	-7.18	-23.35	-40.26	-56.12	41.82
MK08	20*10	AMS_RI	-2.31	-4.05	1.35	1.28	2.32	1.72	2.32	3.8
		$ASTBI$	-90.23	-86.57	7.54	15.81	-50.45	-31.54	79.02	161.22
MK09	20*10	AMS_RI	-3.38	-4.23	-1.61	-2.39	-3.42	-2.51	-0.87	-0.66
		$ASTBI$	-70.32	-66.60	-10.96	-10.75	-23.65	-46.96	-17.24	-16.15
MK10	20*15	AMS_RI	1.23	2.87	-0.48	3.36	1.65	0.74	0.49	3.3
		$ASTBI$	-50.21	-58.23	-16	-3.96	-30.87	-48.32	6.10	-7.82
Moyenne		AMS_RI	-3.38	-2.31	0.116	1.01	-1.41	-1.22	-0.07	1.40
		$ASTBI$	-51.96	-78.84	-2.28	-9.09	-27.50	-45.54	5.69	6.99

TABLE 4.4: Résultats des instances soumises à des pannes de type BD3 et BD4

			BD3				BD4			
Instance	taille		notre 2s-PSO	HGA1	HGA2	HGA3	notre 2s-PSO	HGA1	HGA2	HGA3
Ex1	10*10	AMS_RI	-3.72	-6.98	-2.33	-2.33	-13.54	-9.52	-7.14	0
		$ASTBI$	-17.22	-46.15	0	-5.13	-8.70	-60	-6.67	26.67
Ex2	15*10	AMS_RI	10.61	-2.86	8.57	1.79	-11.39	-1.76	-2.7	5.41
		$ASTBI$	12.62	-82.73	23.21	4.96	-42.94	-37.34	-23.81	-19.05
Ex3	8*8	AMS_RI	-19.86	-13.33	-5.56	0	-27.58	2.47	6.17	6.17
		$ASTBI$	-8.74	-83.96	0.53	-5.88	-24.82	-86.96	139.13	143.48
MK01	10*6	AMS_RI	0.93	-9.39	-6.56	-2.22	-6.27	-8.8	-6.4	-3.2
		$ASTBI$	8.92	-69.71	-8.42	-35.62	-19.26	-44.67	-18.27	-18.78
MK02	10*6	AMS_RI	1.58	0,	3.28	-6.53	-0.83	-7.73	-3.31	4.97
		$ASTBI$	-21.74	-20.94	20.56	-20.94	-34.63	-11.3	75.67	80.27
MK03	15*8	AMS_RI	6.16	-5.24	-5.14	0	-14.96	-13.18	-11.09	-6.11
		$ASTBI$	-33.96	-70.51	-9.24	-15.13	- 50.83	-49.82	0.93	14.58
MK04	15*8	AMS_RI	12.73	-8.74	4.92	-4.24	-14.49	-13.83	-1.61	-0.25
		$ASTBI$	-10.40	-96.38	28.05	22.28	-99.15	-92.11	-4.1	-13.82
MK05	15*4	AMS_RI	-9.35	-7.32	0.61	3.55	-6.54	-13.87	-0.34	1.52
		$ASTBI$	-89.34	-70.19	-28.63	-28.2	-78.26	-60.6	20.77	26.36
MK06	10*15	AMS_RI	-8.21	-7.28	6.31	1.5	-6.18	-4.23	-0.32	1.38
		$ASTBI$	-42.56	-66.23	23.16	1.23	-43.28	-36.99	-21.77	-23.63
MK07	20*5	AMS_RI	-2.76	-0.31	2.15	4.09	-5.76	-6.28	-2.94	2.53
		$ASTBI$	-26.23	-31.18	-14.22	-30.34	-22.87	-32.28	4.8	-19.41
MK08	20*10	AMS_RI	-4.2	-8.4	0.47	1.02	-0.54	-1.93	4.4	5.94
		$ASTBI$	-43.23	-61.78	6.65	-13.47	-43.65	-33.38	23.87	78
MK09	20*10	AMS_RI	-6.2	-5.4	-2.05	-2.51	-1.98	-5.58	-1.29	-0.46
		$ASTBI$	-50.23	-36.99	-4.3	-11.4	-11.65	-26.34	-18.75	-25.78
MK10	20*15	AMS_RI	-5.23	-9	-0.38	-2.77	-1.76	-3.11	-0.46	2.35
		$ASTBI$	-49.76	-75.88	-9.11	-7.56	-3.76	-2.04	-6.61	53.56
Moyenne		AMS_RI	-2.11	-6.48	0.33	-0.66	-7.45	-6.71	-2.07	1.55
		$ASTBI$	-21.95	-62.48	2.17	-11.16	-21.96	-44.14	12.65	23.265

Notre algorithme 2S-PSO obtient de meilleurs résultats que HGA2 face aux pannes de type BD1 et également les pannes de type BD2 à l'exception des instances MK03 et MK07. Concernant les pannes de type BD4, notre algorithme est plus efficace que l'algorithme HGA2 dans toutes les instances et également en terme de stabilité à l'exception des instances MK09 et MK10.

En Comparant notre algorithme par rapport à l'algorithme hybride HGA1 qui utilise la même fonction objective $f2$ pour sélectionner les meilleurs ordonnancements robustes , le 2S-PSO améliore l'efficacité et la stabilité pour quelques instances en considérant différents types de pannes. Selon le tableau 4.3 l'algorithme proposé donne des meilleurs résultats en termes d'AMSRI et de stabilité ASTBI pour les instances EX1, EX2, Ex3, Mk2, mK4, mk5

et mk6 en considérant la panne de machine de type BD1 par rapport à HGA1. Ceci peut être attribué à la nature anticipée de la panne de type BD1 et à sa faible durée. Ainsi l'ordonnancement prédictif peut contenir des temps morts d'inactivité qui vont absorber l'effet de l'incertitude.
Notre méthode a trouvé de meilleurs résultats également face aux pannes de type BD2 pour les instances Ex1, Ex2, Ex 3, Mk2, Mk8, MK9 et face aux pannes de type BD3 aussi pour les instances Ex1, Ex3, MK5, MK6, MK7, MK9 par rapport à HGA1.
En ce qui concerne le type de panne BD4, l'occurrence de la perturbation se fait tardivement. Cela donne moins de chances de trouver des opérations critiques parmi les opérations affectées par la panne. Cela signifie que le nombre d'opérations affectées sera petit. Ainsi l'ordonnancement peut être au pire de cas un peu retardé. Par conséquent la stabilité sera minimisée dans la fonction bi-objective.
Selon le résultat illustré dans le tableau 4.4, il y a une amélioration de la stabilité et de l'efficacité dans toutes les instances. De plus, nous calculons la moyenne de toutes les instances pour chaque type de panne. Comme le montre le tableau 4.3 et le tableau 4.4, la valeur de AMS_RI de l'ordonnancement trouvé par l'algorithme HGA1 avec celle trouvée par notre méthode est passée de -2,31 à -3,38 pour BD1, de -1,22 à -1,41 pour BD2 et de -6,71 à -7,45. Cela indique qu'il ya eu des améliorations sur le makespan réalisé et que les ordonnancements prédictifs sont plus compacts et denses autour de la région où les perturbations réelles sont survenues.

4.7.2 Analyse de variance

Bien que les résultats trouvés favorisent notre algorithme 2S-PSO par rapport aux autres algorithmes HGA1, HGA2 et HGA3 dans la plupart des instances, une analyse de variance (ANOVA) a été effectuée sur les résultats de simulations obtenus afin de déterminer une différence statistique qui avalisera nos conclusions. Chaque cas de test est soumis aux quatre niveaux de pannes. Pour analyser plus en détail l'impact du type de panne sur la performance de l'approche proposée, un test ANOVA à un seul critère (one way ANOVA) est effectué.
Les tableaux 4.5, 4.6 et 4.7 représentent les résultats d'ANOVA appliqués pour l'AMS_RI et les $STBI$ de la méthode proposée en comparaison respectivement avec HGA1, HGA2 et HGA3.
Les tableaux 4.5, 4.6 et 4.7 montrent le rapport F et la valeur P des résultats obtenus à partir des expériences précédentes. Les résultats de ce test montrent l'effet de la mesure

de la robustesse et de la stabilité utilisée, l'instance de test considérée, le type de panne et l'interaction entre ces facteurs sur la qualité relative de l'ordonnancement prédictif. Dans cette étude, les effets sont considérés comme significatifs si la valeur P est inférieure à 0,05. Les meilleures valeurs inférieures à 0,05 sont imprimées en gras.

TABLE 4.5: Résultats ANOVA comparant AMS_RI et $STBI$ de la méthode proposée et HGA1

type de panne BD	AMS_RI		STBI	
	P-value	F-ratio	P-value	F-ratio
BD1	**0.0025**	5.37	**0.046**	4.86
BD2	**0.0415**	3.11	0.65	0.35
BD3	0.110	2.75	0.48	0.73
BD4	**0.046**	3.54	0.138	2.54

TABLE 4.6: Résultats ANOVA comparant AMS_RI et $STBI$ de la méthode proposée et HGA2

Type de panne BD	AMS_RI		STBI	
	P-value	F-ratio	P-value	F-ratio
BD1	**0.00813**	8.32	**0.000202**	19.15
BD2	0.46	0.54	**0.02854**	5.42
BD3	0.3812	0.79	**0.000222**	11.72
BD4	**0.014**	6.95	**0.002776**	11.11

TABLE 4.7: Résultats ANOVA comparant AMS_RI et $STBI$ de la méthode proposée et HGA3

Type de panne BD	AMS_RI		STBI	
	P-value	F-ratio	P-value	F-ratio
BD1	**0.00213**	11.83	**0.000208**	19.06
BD2	0.147	2.24	0.0614	3.84
BD3	0.0576	0.32	0.0587	3.93
BD4	**0.00024**	18.56	**0.0011**	13.54

Les résultats dans les tableaux 4.5, 4.6 et 4.7 indiquent que le type de panne a un effet significatif sur les mesures de la valeur P et du rapport F. Selon le tableau 4.5, le 2S-PSO est statistiquement différent du HGA1 lorsqu'il fait face à BD1, BD2 et BD3 (valeur P <0,05). Comme on peut le constater dans les tableaux 4.6 et 4.7 pour la plupart des cas, il existe une différence statistiquement significative entre le 2S-PSO et le HGA2 et le HGA3 lorsque l'on utilise les mesures de robustesse f3 et la mesure de robustesse f4 basée sur le voisinage. Dans la plupart des cas, la 2S-PSO offre une meilleure performance que les autres approches.

4.7.3 Comparaison de temps CPU entre PSO et 2s-PSO

Nous avons effectué une comparaison entre le temps de traitement de PSO déterministe et de PSO à deux stages tout en conservant la même configuration pour chaque approche ($swarmsize = 100$, $Max_{Iteration} = 100$). Les résultats obtenus sont présentés dans le tableau 4.8.

TABLE 4.8: Comparaison de temps CPU entre PSO et 2s-PSO

Instance	PSO (*) Temps CPU (s)	2S-PSO proposé Temps CPU (s)
4*5	0.353	19.402
10*10	7.316	52.604
8*8	5.698	31.375

Comme le montre ce tableau, le temps CPU de l'algorithme proposé 2s-PSO est beaucoup plus élevé que le temps CPU dans le cas $\gamma = 1$. Le 2s-PSO augmente significativement le temps de calcul. Ceci s'explique par le fait qu'il y a beaucoup de fonctions supplémentaires ajoutées par rapport à l'algorithme PSO de base. Parmi ces fonctions on peut citer la simulation de pannes, la détermination des opérations directement et indirectement affectées, le calcul de la mesure de stabilité, la procédure de réordonnancement, et enfin l'exécution de l'algorithme PSO une fois pour minimiser le makespan et une autre fois pour optimiser la fonction bi-objective avec intégration de la probabilité de panne machine.

À notre avis, cette augmentation du temps de calcul n'est pas un inconvénient majeur. En effet, le modèle proposé dans ce travail est prédictif (hors ligne). Par conséquent, il est concevable de consacrer plus de temps à trouver un ordonnancement prédictif plus robuste, plus stable et ayant le makespan le plus bas.

Le problème du ré-ordonnancement a une importance fondamentale égale à celle du problème d'ordonnancement initial. Dans ce travail, nous proposons une méthode de ré-ordonnancement qui ne reproduit que les opérations directement et indirectement affectées par une perturbation. Nous nous limitons ici uniquement aux éventuelles perturbations dues à l'indisponibilité temporaire d'une machine. L'idée est de remettre à jour le programme de production existant lorsque l'état du système de fabrication le rend inapplicable, afin de minimiser l'effet des pannes de machine sur la performance globale et augmenter également la stabilité de l'ordonnancement. Cela sera réalisé en réattribuant les opérations affectées sur d'autres machines disponibles sans prendre en compte la machine défectueuse.

Dans la section suivante nous décrivons en détail notre heuristique de ré-ordonnancement ainsi proposée.

4.8 Une nouvelle heuristique de ré-ordonnancement pour FJSP

Les systèmes réels de production sont dynamiques et sont sujets à plusieurs perturbations et des événements non planifiés peuvent surgir à tout moment. La survenue d'une panne rend souvent l'ordonnancement prédictif invalide. Le besoin d'une méthode de ré-ordonnancement est alors primordial.

L'objectif étant d'atténuer la dégradation des performances suite aux aléas tout en ayant un ordonnancement avec un retard minimal (réparer et réduire l'effet de cette perturbation en un minimum de temps). La méthode de ré-ordonnancement utilisée influence la valeur du makespan de l'ordonnancement réalisé par la méthode prédictive de l'ordonnancement dynamique.

Une caractéristique très importante pour la méthode de ré-ordonnancement est le fait que le système doit trouver un ordonnancement de haute qualité avec une promptitude de réaction. Cependant, les méthodes existantes n'étudient pas le temps de calcul pour trouver le nouvel ordonnancement et l'instant de ré-ordonnancement. De ce constat, nous proposons une nouvelle heuristique de ré-ordonnancement ayant le moindre temps de calcul de la méthode de ré-ordonnancement.

Notre méthode se base sur l'extraction d'une sous-particule contenant toutes les opérations concernées par la défaillance afin de modifier leur assignement (affectation). En effet, ils sont attribués soit à la machine ayant le moindre temps de traitement soit à la première disponible soit de façon aléatoire.

L'idée est de mettre à jour un ordonnancement initial existant lorsque l'état du système de fabrication le rend inapplicable, de sorte qu'il minimise l'effet des pannes de machine sur la performance globale en sauvegardant la stabilité de l'ordonnancement. Cela sera réalisé en réattribuant les opérations affectées sur d'autres machines disponibles en faisant abstraction de la machine en panne.

Dans cette section, nous présentons d'abord l'algorithme utilisé pour générer l'ordonnancement initial (prédictif). Nous présentons ensuite les détails de la méthode de ré-ordonnancement proposée.

4.8.1 Construction de l'ordonnancement initial

La figure suivante illustre l'organigramme global de la méthode. En effet, la méthode de ré-ordonnancement va être appliquée sur l'ordonnancement initialement construit dans la phase offline. L'analyse de performance de la méthode de ré-ordonnancement se fait toujours à travers une comparaison de l'ordonnancement initial perturbé et l'ordonnancement construit après la phase de réparation. De ce fait, la qualité de l'ordonnancement intial est très importante et affectera la phase de ré-ordonnancement.
Dans notre approche, nous utilisons l'algorithme d'optimisation par essaims particulaires PSO, présenté dans le chapitre 2, pour trouver l'ordonnancement préventif. L'objectif étant de minimiser le makespan trouvé.

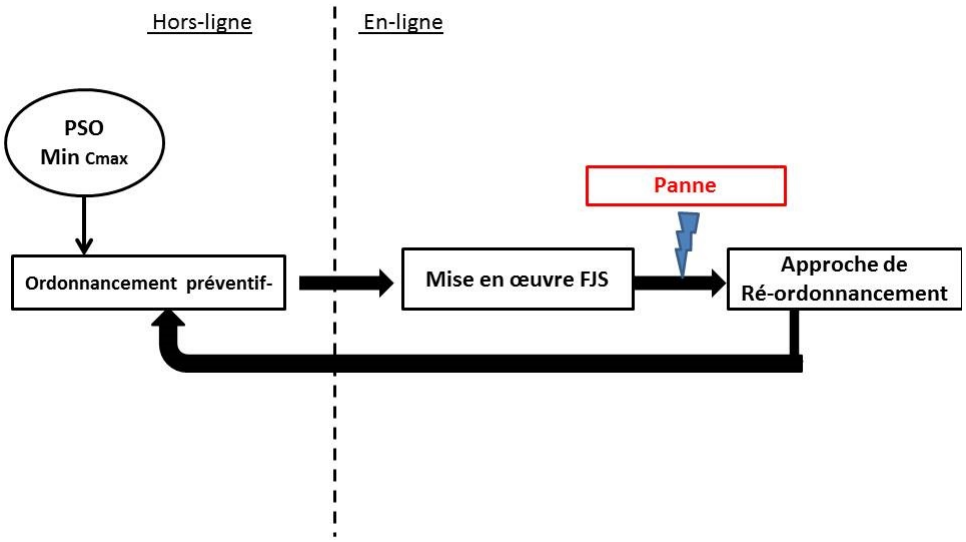

FIGURE 4.8: l'approche prédictive réactive basée sur l'algorithme PSO

4.8.2 L'algorithme de l'heuristique de ré-ordonnancement

Nous supposons qu'il ne peut y avoir à la fois qu'une seule machine en panne, cette machine étant irréparable et sans reprise des opérations. Cela signifie que toutes ces opérations redémarreront quand elles sont interrompues par une panne. Nous supposons que certaines informations sur l'incertitude des pannes de machines sont disponibles à l'avance et peuvent être quantifiées par certaines distributions.
Dans ce travail, nous proposons d'utiliser la flexibilité de routage pour modifier l'affectation des opérations affectées aux autres machines. La première étape de l'algorithme proposé consiste à extraire une sous-particule contenant les opérations directement et indirectement affectées pour modifier leurs affectations. Les opérations directes sont celles situées sur la

machine en échec et les autres opérations indirectement perturbées sont liées à la contrainte de précédence.

L'ordonnancement prédictif étant présenté comme une particule composée de deux vecteurs de $Process[\ ]$ et de $Machine[\ ]$, de même la solution de la méthode de réordonnancement sera représentée comme une particule composée de deux vecteurs : $Process[\ ]$ définit toutes les opérations affectées alors que le vecteur $Machine[\ ]$ représente les machines affectées à ces opérations.

La différence principale entre les deux particules de l'étape d'ordonnancement et celle de ré-ordonnancement est la taille de deux vecteurs $Process[\ ]$ et $Machine[\ ]$. En fait, dans le cas d'ordonnancement, la taille est fixée, elle est égale au nombre total d'opérations dans le problème d'ordonnancement prédictif. Cependant, quand il s'agit de ré-ordonnancement, la sous-particule a une taille variable en fonction du nombre d'opérations directement et indirectement affectées. Cette représentation nous permet de donner plus de flexibilité lors de la recherche de la nouvelle solution.

Une deuxième amélioration consiste à ajouter un autre vecteur dans la procédure de codage qui est le vecteur du temps de disponibilité de la machine nommé $MachineAvailability[\ ]$. La taille de ce vecteur est égale au nombre total d'opérations et prend compte de l'évolution de la disponibilité de chaque machine après chaque opération. Ce vecteur résout la première opération affectée suite à la panne de la machine et détermine également la première machine disponible.

Dans notre méthode, nous réassignons les opérations affectées en utilisant ces deux méthodes :

— **Méthode aléatoire :** qui réattribue chacune des opérations affectées à une machine aléatoire capable de traiter ces opérations : pour chaque composant du vecteur $Process[\ ]$ de sous-particules, nous choisissons une nouvelle affectation de machine au hasard parmi les machines disponibles à l'exception de la machine en échec . La variable $Alea_{iteration}$ définit le nombre d'exécutions de cette méthode pour trouver une autre sous-particule avec une nouvelle affectation. En cas de flexibilité totale et lorsqu'il n'y a qu'une seule opération affectée dans la sous-particule, alors le nombre de solutions non redondantes est égal au nombre total de machines à l'exception de celle qui a échoué. S'il y a une flexibilité partielle, ce nombre est égal au nombre de machines disponibles fixées pour cette opération à l'exception de celle en panne.

— **Méthode de sélection de la machine la plus tôt disponible :** ou encore " Minimum Earliest method ". Cette méthode conduit à obtenir une seule solution. Pour chaque composant du vecteur *Process*[ ] de la sous-particule, on choisit la machine ayant le temps de traitement minimum pour cette opération. Si plusieurs machines répondent à ce critère, alors nous choisissons la première disponible. A noter que dans le FJSP partiel, en cas où une seule opération est affectée et peut s'exécuter uniquement sur la machine défaillante, nous utilisons alors systématiquement la méthode de ré-ordonnancement RSR [AS97].

La procédure de ré-ordonnancement global est la suivante :

Algorithme 11 L'heuristique de ré-ordonnancement proposée

Step1 : **Input parameters** : the prechedule p, number of breakdown machine m_b, the start time of breakdown st, the duration of repair procedure d
Step2 : Extract **subparticle** that contains the directly affected operations and indirectly affected to modify their assignments.
Step3 : Construction new subparticles by reassigning each affected operation on new machines while using the two methods cited above
-Random method ($Alea_iteration$)
-Minimum Earliest method (1 iteration).
Step4 : Construction of *swarmReschedule* that contains all particles with the new assignments :
-Reconstruction of particles : the same sequence as the preschedule p with the new machines assignment of affected operations.
*Step5 :*Re-Evaluate the fitness value of all particles of *SwarmReschedule* with considering the duration of breakdowns.
*Step6 :*Select the best particle after reschedule with lowest makespan value.

4.8.3 Exemple illustratif

Pour illustrer l'heuristique de ré ordonnancement, nous considérons un exemple pour le job shop flexible avec quatre jobs et cinq machines. La figure 4.9 montre l'ordonnancement prédictif trouvé après l'application de l'algorithme PSO qui minimise le makespan. La valeur du makespan est ici égale à 12. La particule correspondante à cet ordonnancement est illustrée dans la figure 4.10.

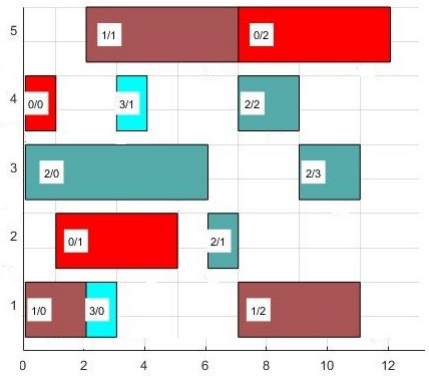

FIGURE 4.9: Ordonnancement préventif obtenu par PSO minimisant Makespan

Process	2	1	0	0	3	3	1	1	2	2	2	0
Machine	3	1	4	2	1	4	5	1	2	4	3	5

a. Particle

Breakdowns level = **low early** Start time of breakdown = **3**
Number of machine breakdowns=**5** Duration of reparation =**1**

b. Breakdown Simulation

FIGURE 4.10: Particule correspondante à l'ordonnancement dans la figure 4.9

Cet ordonnancement prédit est soumis à une panne spécifiée par le rectangle jaune. Par exemple, les données historiques indiquent que la machine $M5$ a un risque élevé de défaillance, cette machine étant fortement sollicitée et de fiabilité limitée. La figure 4.11 montre l'ordonnancement après l'application de la méthode de ré-ordonnancement de décalage à droite RSR. La valeur du makespan est égale à 14 (voir figure 4.11).

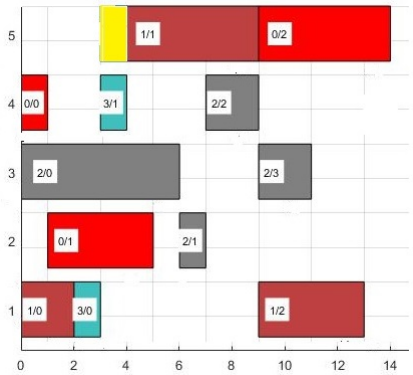

FIGURE 4.11: Le nouveau ordonnancement aprés l'application de RSR

Selon l'étape 2 de notre algorithme de ré-ordonnancement, on extrait d'abord une sous-particule contenant les opérations affectées (voir figure 4.12.a). La deuxième opération du

job 1 et la dernière opération du job 0 sont les opérations directement affectées par la panne, car elles se trouvent sur la machine $M5$ défaillante. La dernière opération du job 1 est indirectement affectée par la défaillance en fonction de la contrainte de précédence.
Ensuite, les opérations affectées sont réaffectées sur de nouvelles machines(étape 3). On obtient des nouvelles sous-particules avec de nouvelles machines après l'application d'une méthode aléatoire. Le nombre des nouvelles solutions trouvées par cette méthode est égale à d'$Alea_{iteration}$. La figure 4.12.b illustre un exemple d'une nouvelle sous-particule obtenue par la Méthode Aléatoire.
Après avoir appliqué la méthode "Minimum Earliest", nous n'obtenons qu'une seule nouvelle sous-particule illustrée dans la figure 4.12.c. Pour la deuxième opération du job 1, il n'y a qu'une machine avec un temps de traitement minimal $M1$ alors que pour la dernière opération du job 1, il y a deux machines $M1$ et $M3$ ayant le même temps de traitement (égal à 4). Dans ce cas, on choisit la première machine disponible, déterminée selon le vecteur $MachinesAvailablity[\ ]$. Ainsi, la machine $M3$ sera sélectionnée pour traiter la dernière opération du job1 car elle est disponible à l'instant 6 alors que la machine $M1$ n'est disponible qu'à l'instant 11 (voir la figure 4.13).

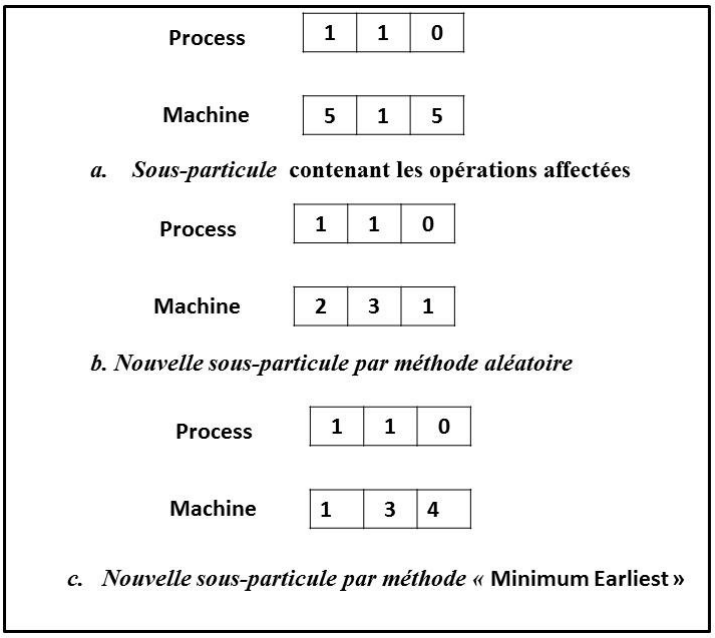

FIGURE 4.12: La *sousparticule* et les nouvelles *sousparticules*

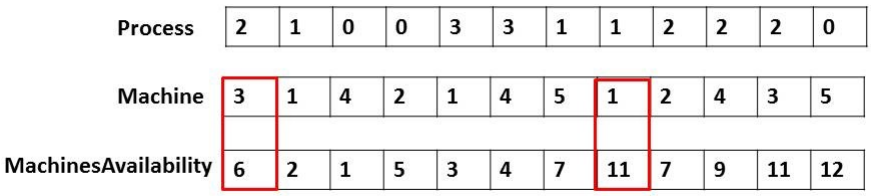

FIGURE 4.13: Le vecteur *MachinesAvailability*[ ]

Ensuite, en se basant sur toutes les sous-particules ainsi obtenues, un essaim regroupant les nouvelles particules avec les différentes nouvelles affectations, est construit (étape 4). Cet essaim est appelé *SwarmReschedule*.

Afin de réduire l'écart entre l'ordonnancement initial et celui réparé, la séquence de l'ordonnancement original est maintenue et seules les machines des opérations affectées sont modifiées.

Après avoir réévalué la valeur de la fonction objective de toutes les particules de *SwarmReschedule* tout en tenant compte de la durée des pannes, la particule ayant le plus faible makespan sera choisie.

Dans cet exemple, la meilleure particule trouvée après l'application de la méthode de ré-ordonnancement proposée est illustrée dans la figure 4.14.

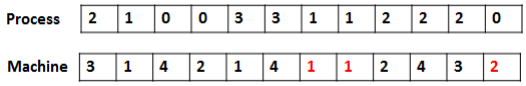

FIGURE 4.14: Meilleure particule obtenue par notre heuristique

La figue 4.15 montre le meilleur ordonnancement après avoir appliqué notre heuristique de ré-ordonnancement.

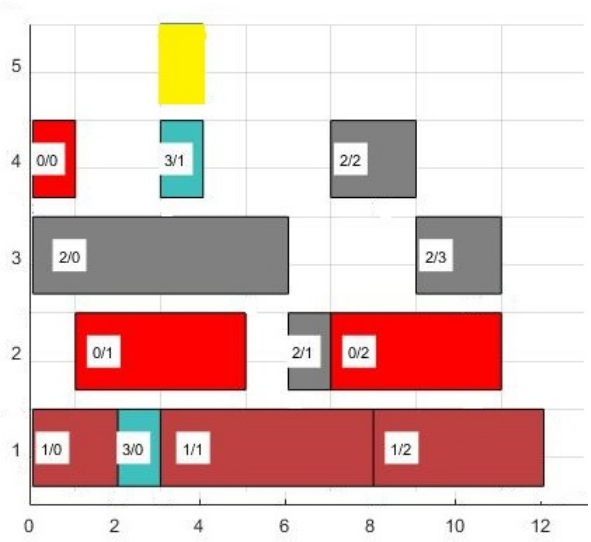

FIGURE 4.15: Diagramme de Gantt de la meilleure particule obtenue par notre heuristique

Comme on peut le constater, le makespan de l'ordonnancement réparé est égal à la valeur du makespan de l'ordonnancement initial soit 12. Cependant, si nous appliquons la méthode RSR, il y a une dégradation de la valeur du makespan à 14. Ceci montre l'efficacité de l'heuristique proposée par rapport à la méthode RSR.

4.8.4 Expérimentations et analyse de performance de la nouvelle heuristique de ré-ordonnancement

Des expérimentations sont menées pour valider l'efficacité de notre heuristique de ré-ordonnancement proposée. Dans ce qui suit, nous présentons exactement comment la perturbation doit être générée et comment évaluer la performance de la méthode proposée.

4.8.4.1 Simulation de panne

Pour mesurer son efficacité à réagir contre les perturbations, l'ordonnancement sera exécuté pour différents scénarios de perturbations. Le makespan obtenu après le ré-ordonnancement sera appelé makespan réparé (the repaired makespan). Simuler une panne comprend le choix de la machine affectée, et à quel moment elle redeviendra opérationnelle. Dans ce travail, la génération de pannes de machines est simulée selon la même procédure présentée dans [MlSlH10] bien détaillée dans la section précédente numéro 4.4. Nous avons testé notre algorithme sur les différents types de panne BD1, BD2, BD3 et BD4.

4.8.4.2 Les métriques de performance de l'heuristique de Ré-ordonnancement

Dans ce travail nous utilisons deux types de mesure de performance proposées dans [SR03] à savoir l'efficacité et la stabilité.

- **L'efficacité :** cette mesure indique l'efficacité de l'ordonnancement réparé. Elle est définie comme étant la variation en pourcentage de la valeur du makespan reparé et celle de l'ordonnancement initial. Elle est définie par l'équation suivante :

$$\eta = 1 - \frac{M_{new} - M_0}{M_0} * 100 \tag{4.13}$$

Avec :

-M_{new} : makespan de l'ordonnancement réparé en utilisant la nouvelle méthode de ré-ordonnancement,

-M_0 : makespan de l'ordonnancement original.

Le makespan de l'ordonnancement réparé est toujours supérieur ou égal à celui de l'ordonnancement initial. Le meilleur processus de réparation est celui qui engendre une augmentation minimale de la valeur du makespan de l'ordonnancement réparé. Par conséquent, le rendement maximum 100% est atteint si les makespan de l'ordonnancement original et celui réparé sont identiques.

- **La stabilité :** La stabilité d'un ordonnancement est mesurée en termes d'écarts des dates de début des opérations de job par rapport à l'ordonnancement initial. Un ordonnancement sera stable s'il s'écarte de façon minimale de l'horaire initial. L'écart dans le temps de départ est calculé comme la somme absolue de la différence dans les temps de départ des opérations entre l'ordonnancement initial et celui réparé. Il est normalisé par le nombre total d'opérations dans l'ordonnancement. Une meilleure heuristique de réparation minimise l'écart normalisé qui est défini comme suit :

$$\xi = \frac{\sum_{j=1}^{k} \sum_{i=1}^{p_j} |S^*_{ji} - S_{ji}|}{\sum_{j=1}^{k} p_j} \tag{4.14}$$

Avec

-ξ : écart normalisé,

-p_j nombre d'opérations du job j,

-K : nombre des jobs,

-S_{ji} Heure de début de la ième opération de la tâche j dans la planification originale,

-S^*_{ji} heure de début de la ième opération de la tâche j dans la planification réparée.

4.8.4.3 Résultats expérimentaux

Pour tester l'efficacité et la performance de l'algorithme de ré-ordonnancement proposé, nous avons effectué des expériences avec quatre instances de benchmark FJSP différentes : trois avec une flexibilité totale et une avec une flexibilité partielle. Chaque instance peut être

caractérisée par le nombre de jobs (n), le nombre de machines (m) et les opérations associées O_{ij} à chaque job.

Pour ces cas, le nombre de job varie de 4 à 15, le nombre de machines de 5 à 10 et le nombre d'opérations de 12 à 56. Les données concernants ces instances sont disponibles dans [MlSlH10]. La méthode proposée est comparée à la méthode RSR en fonction de la mesure de l'efficacité et de la stabilité. Les ordonnancements prédictifs initiaux sont générés en utilisant l'algorithme PSO qui minimise le makespan (décrit dans le chapitre 2).

Les ordonnancements sont soumis à des perturbations de différentes dimensions. L'ordonnancement initial est alors réparé à l'aide de la méthode RSR et de notre heuristique de ré-ordonnancement. Chaque instance est testée pour les quatre scénarios de perturbations. Chaque instance est exécutée 10 fois pour chaque scénario de panne. Les tableaux 4.9, 4.10, 4.11 et 4.12 montrent respectivement les résultats comparés en termes d'efficacité et de stabilité face aux pannes de type BD1, BD2, BD3 et BD4.

Les tableaux se composent de trois colonnes. La première colonne représente la taille de l'instance. Les autres colonnes représentent l'efficacité et la stabilité de chaque ordonnancement réparé obtenues en utilisant respectivement notre algorithme de ré-ordonnancement et la méthode RSR lorsqu'ils sont soumis à un type de panne spécifique.

TABLE 4.9: Résultats de l'heuristique avec BD1

Instance	Efficacité%		Stabilité	
	Notre méthode	RSR	Notre méthode	RSR
4*5	**87.11**	86.285	**0.5661**	0.5666
10*10	81.162	90.105	1.709	1.425
15*10	**93.543**	88.647	**0.680**	0.911
8*8	72.67	96.534	0.454	0.291
Moyenne	83.623	90.392	0.852	0.798

TABLE 4.10: Résultats de l'heuristique avec BD2

Instance	Efficacité%		Stabilité	
	Notre méthode	RSR	Notre méthode	RSR
4*5	**78.9891**	69.8762	**0.6911**	1.507
10*10	**93.193**	75.968	**0.029**	0.1245
15*10	**92.074**	86.683	**0.575**	0.679
8*8	83.88	92.891	0.281	0.096
Moyenne	**87.034**	81.354	**0.394**	0.601

TABLE 4.11: Résultats de l'heuristique avec BD3

Instance	Efficacité%		stabilité	
	Notre méthode	RSR	Notre méthode	RSR
4*5	**85.603**	68.934	**1.482**	2.256
10*10	**92.33**	86.996	**0.506**	0.922
15*10	**89.339**	68.575	**1.065**	2.36
8*8	70.89	86.79	1.483	0.95
Moyenne	**84.54**	77.823	**1.134**	1.622

TABLE 4.12: Résultats de l'heuristique avec BD4

Instance	Efficacité%		Stabilité	
	notre méthode	RSR	Notre méthode	RSR
4*5	**79.186**	45.662	**1.336**	2.448
10*10	**98.888**	70.216	**0.052**	1.186
15*10	**97.705**	65.385	**1.271**	2.494
8*8	**88.478**	72.962	**0.225**	0.659
Moyenne	**91.06**	63.556	**0.721**	1.696

Les résultats indiquent que l'approche proposée donne des performances significativement meilleures que l'heuristique RSR en termes d'efficacité et de stabilité.

Un petit aperçu des différents tableaux révèle que l'utilisation de la méthode proposée améliore la moyenne de la stabilité de 0.601 à 0.394 face à BD2, de 1.622 à 1.134 face à BD3 de 1.696 à 0.721 face à BD4 et augmente la moyenne d'efficacité de 81.354% jusqu'à 87.034% en faisant face à BD2, de 77.823% à 84.54% en regard de BD3 et de 63.556% à 91.06% en regard de BD4.

L'augmentation de la valeur de l'efficacité signifie que pour les nouveaux ordonnancements, les makespans obtenus en utilisant RSR étaient significativement plus élevés que ceux obtenus en utilisant la nouvelle heuristique de ré-ordonnancement proposée. En face de BD1, notre méthode donne de meilleurs résultats en testant l'instance 4 * 5 et 15 * 10.

Cependant, la technique RSR semble avoir un meilleur rendement en face de BD1, BD2 et BD3 pour l'instance 8 * 8 avec une flexibilité partielle. La méthode de ré-ordonnancement proposée donne des performances inférieures pour les instances avec flexibilité partielle parce que dans de tels cas il y a moins de flexibilité pour les affectations de machines. Néanmoins, quand on considère le BD4, l'algorithme proposé fonctionne mieux dans tous les types d'instances que ce soit avec une flexibilité totale ou partielle.

En fait, le type de panne BD4 se produit à un stade tardif et a une durée élevée ; ainsi un nombre inférieur d'opérations affectées dans la sous-particule est trouvé, ce qui préserve plus la stabilité de l'ordonnancement réparée. D'aprés ces expériences, la taille de l'ordon-

nancement n'a alors aucun impact sur la performance du processus de ré-ordonnancement. Cependant, la durée de la panne de la machine a un effet significatif. Si la durée de la dégradation est faible, indépendamment de l'apparition précoce ou tardive du programme (BD1, BD2), elle est accommodée avec une facilité relative et dans ce cas la méthode RSR simple peut résoudre le problème avec une performance suffisante. Néanmoins, la technique RSR montre des résultats de performance médiocres pour des pannes de durées élevées. Le plus mauvais résultat est obtenu pour le type de panne BD4 (durée : élevée-temps :tard). Dans ce cas, il est très difficile pour la méthode RSR d'accommoder une perturbation tardive d'une durée élevée et donc l'augmentation du makespan est inévitable.

En outre, l'une des caractéristiques les plus importantes de la procédure de réparation ou de ré-ordonnancement est le temps de traitement de la méthode qui définit la rapidité avec laquelle la procédure de réparation réagit à la perturbation.

Ainsi, le temps de traitement CPU de l'algorithme de ré-ordonnancement proposé est étudié et comparé à celui engendré par la méthode RSR. Les résultats obtenus sont indiqués dans le tableau 4.13.

TABLE 4.13: Comparaison de CPU entre notre heuristique et RSR

Instance	Temps CPU de notre méthode (s)	Temps CPU de RSR(s)
4*5	0.255	0.017
10*10	0.512	0.02
15*10	0.917	0.182
8*8	0.453	0.037

Comme on peut le constater dans le tableau 4.13, notre heuristique de ré-ordonnancement a besoin de plus de temps pour trouver l'ordonnancement réparé le plus stable et le plus efficace. Cependant, la technique RSR nécessite moins de temps CPU pour trouver une solution. Cela s'explique par le fait que le résultat de la méthode RSR offre une seule solution (un seul ordonnancement fixe) tandis que la méthode proposée cherche de nombreuses solutions réalisables afin de sélectionner la meilleure.

Néanmoins ce temps varie dans la pratique, l'ordonnancement trouvé par la méthode RSR n'est applicable dans l'atelier de production qu'après réparation de la machine en échec. Par contre notre heuristique ne nécessite pas ce temps d'attente de réparation.

Pour réduire le temps de la méthode proposée, la valeur de la variable $Alea_{iteration}$ doit être configurable et adaptable au type de panne et à la taille de la sous-particule. En pratique,

le ré ordonnancement est effectué soit occasionnellement ou d'une facon périodique afin de planifier les activités futures en fonction de l'état du système. Cependant notre méthode est conçue pour être lancée suite à un événement. Le temps de ré-ordonnancement de l'heuristique proposée est similaire ou inférieur au temps de traitement le plus court pour toutes les instances testées. Ainsi, il peut être négligé et il fournit une réaction rapide en fonction des échecs.

4.9 Conclusion

Dans ce chapitre, nous avons présenté notre approche prédictive pour résoudre le problème de FJSP dynamique. Dans ce travail, le terme incertitude est consacré aux pannes machines. Nous avons présenté comment l'algorithme PSO développé pour résoudre le cas statique de FJSP peut être étendu pour résoudre le cas dynamique sous incertitudes. Nous avons détaillé les différentes étapes et un exemple illustratif a été présenté.
Des expérimentations ont été réalisées pour tester l'efficacité de notre algorithme 2S-PSO pour l'ordonnancement robuste. Cet algorithme a été comparé à trois méthodes de résolution présentées dans la littérature. Les résultats prouvent que cet algorithme permet de donner des solutions acceptables dites de compromis entre efficacité et robustesse, pour différents degrés d'incertitudes. De plus, notre algorithme converge rapidement quel que soit le type d'instance. Ceci nous garantira une meilleure maîtrise des risques, pour faire face aux perturbations auxquelles un système de production sera confronté en phase d'exploitation. Le temps d'exécution de l'algorithme est aussi étudié et comparé avec celui de la méthode déterministe.
Dans ce chapitre nous avons abordé aussi le problème du ré-ordonnancement du FJSP face à une panne machine. Une nouvelle heuristique de ré-ordonnancement est proposée. Les résultats de calcul indiquent que notre méthode a un niveau de performance en termes d'efficacité et de stabilité supérieur à la technique RSR.

Conclusion Générale

L'optimisation du problème d'ordonnancement dans le secteur industriel est un enjeu primordial pour augmenter la productivité et l'efficacité.
Cependant, face à un environnement dynamique et de plus en plus incertain, les approches de résolution du problème d'ordonnancement doivent être plus flexibles et réactives afin de répondre au mieux aux demandes pressantes du marché. C'est dans ce contexte que s'inscrit notre thèse.
La résolution du problème d'ordonnancement d'ateliers de type job shop flexible sans et avec incertitudes constitue la principale contribution des travaux consignés dans cette thèse.
La première tâche entreprise a été la familiarisation avec la notion d'ordonnancement, sa modélisation et ses composantes, une analyse des différents types d'ateliers de production (job-shop, flow-shop et open-shop) ainsi que la panoplie de méthodes d'optimisation utilisées pour leurs résolutions. Parmi ces méthodes on peut citer les méthodes exactes et méthodes approchées.
Un état d'art est ensuite effectué sur les méthodes d'optimisation utilisées pour la résolution du problème FJSSP sans incertitudes. Nous avons opté de structurer cette étude bibliographique selon l'architecture utilisée lors du développement de la méthode qu'elle soit centralisée ou distribuée.
Un autre tour d'horizon est aussi effectué sur les méthodes d'ordonnancement sous incertitudes et les techniques de ré-ordonnancement.
Par la suite, notre intérêt s'est focalisé sur l'étude de la métaheuristique PSO. Notre contribution dans ce contexte est de proposer une construction de la population initiale tout en utilisant plusieurs méthodes d'initialisation avec un pourcentage différent. Nous avons apporté une modification sur la méthode d'initialisation "approche de localisation de kacem" donnant naissance ainsi à notre nouvelle méthode nommée "MMkacem". Nous avons testé plusieurs configurations pour les paramètres ainsi que les topologies de voisinages utilisés afin de trouver les meilleures en sens de convergence de l'algorithme.
Ensuite nous avons proposé deux architectures multi agent à base de PSO nommé MAPSO1

et MAPSO2. Nous avons proposé également d'intégrer une nouvelle phase de migration de meilleures particules entre les agents exécutants afin de diversifier l'espace de recherche.
Les résultats obtenus, comparés à ceux relatifs aux méta-heuristiques optimisant le même objectif, ont montré que la méthode PSO proposée permet d'obtenir des solutions meilleures pour plusieurs instances de tests.
Notre objectif étant de construire une solution souple, capable de prendre les décisions d'ordonnancement et de faire face aux incertitudes internes liées aux pannes, nous avons proposé d'utiliser des systèmes embarqués pour le déploiement d'un processus d'optimisation intelligent et coopératif comme le PSO.
Ainsi l'architecture MAPSO2 a été distribuée sur un système physiquement distribué comportant deux systèmes embarqués ARM A7 pour décentraliser le processus d'optimisation sur des entités décentralisées, intelligentes et communicantes. Dans ce contexte, nous avons étudié l'apport en performance en cas d'implication des systèmes embarqués dans la résolution PSO. Suite aux expérimentations exécutées, plusieurs améliorations de l'architecture ont été proposées donnant naissance à MAPSO2+ et MAPSO2++. Ajuster la communication entre les entités distantes d'une application distribuée permet de réduire le temps d'exécution. Pour ce faire, des capteurs de température sont ajoutés sur chaque système embarqué destiné comme conteneur d'Agent Exécutant, réduisant ainsi le côut de communication menant à un temps de traitement plus réduit. Les expérimentations ont montré l'efficacité du modèle proposé.
Une autre contribution dans ces travaux consiste à proposer une approche prédictive basée sur PSO. En effet, selon nos connaissances, il n'existe pas une méthode à base de l'algorithme PSO traitant le problème FJSP avec la prise en compte des pannes de machines. Ainsi nous avons opté pour étendre l'algorithme PSO (proposé pour FJSP statique) afin de trouver un ordonnancement avec minimum de makespan à la fois robuste et stable. Ceci a été réalisé en développant un nouveau mécanisme intégré dans l'algorithme PSO. Ce mécanisme consiste à évaluer, à chaque génération de l'algorithme PSO, toutes les particules sur un ensemble de scénarios perturbés. Ces évaluations ont été effectuées par une fonction bi-objective tenant compte à la fois de la robustesse et de la stabilité.
Nos expérimentations ont montré que notre algorithme proposé 2s-PSO donne de meilleurs résultats par rapport à l'algorithme génétique hybride développé par [AHE11] en utilisant plusieurs fonctions bi-objectives.
Ainsi, et comme dernière contribution, nous avons proposé une heuristique de ré-ordonnancement pour réparer l'ordonnancement en cas d'inefficacité sous incertitudes.

Notre heuristique exploite la flexibilité de routage pour réaffecter les opérations perturbées par les pannes machines. Les résultats obtenus, ont montré l'efficacité de notre heuristique par rapport à la méthode de décalage à droite (Right Shift Rescheduling).
Enfin, les travaux réalisés dans le cadre de cette thèse, ouvrent la voie à de nouvelles perspectives pour l'exploitation de nos contributions dans les études futures, théorique (recherche), méthodologique ou pratique (applications). Nous envisageons de compléter nos travaux en développant d'autres aspects :

— Tenir compte d'autres contraintes lors de la résolution : comme le temps de transport considéré négligeable dans ce travail.

— Optimiser un autre critère important à savoir la consommation d'énergie dans un système distribué distant composé de systèmes embarqués.

— Implémenter un composant matériel dédié à l'algorithme d'optimisation PSO.

— Etudier la performance d'une implémentation logicielle (software) et une implémentation matérielle (hardware) d'un algorithme d'optimisation comme le PSO dans un système embarqué.

— Tenir compte d'autres types d'incertitudes autres que les pannes machines.

— Intégrer la nouvelle heuristique de ré-ordonnancement dans l'algorithme 2s-PSO et voir son effet sur la qualité de l'ordonnancement préventif ainsi développé.

— Intégrer la méthode de ré-ordonnancement développée sur un système embarqué.

— Proposer une hyper heuristique basée sur un mécanisme d'apprentissage afin de sélectionner et générer le bon procédé de ré-ordonnancement selon l'état du système de fabrication.

Liste des publications

- **Revues Internationales :**
  - **Maroua Nouiri**, Abdelghani Bekrar, Abderrazak Jemai, Damien Trentesaux, Ahmed Chiheb Ammari, Smail Niar. Two stage particle Swarm optimisation to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. *Computers & Industrial Engineering*, 2017.
  - **Maroua Nouiri**, Abdelghani Bekrar, Abderrazak Jemai, Smail Niar, Ahmed Chiheb Ammari. An effective and distributed particle swarm optimization to solve flexible job shop scheduling problem.*Journal of Intelligent Manufacturing*, pp 1-13, 2015.
- **Conférences Internationales :**
  - **Maroua Nouiri**, Abderrazak Jemai, Ahmed Chiheb Ammari, Abdelghani Bekrar, Damien Trentesaux, Smail Niar. Using iot in breakdown tolerance : Pso solving fjsp. *In 11th International Design & Test Symposium (IDT)*, IEEE, Hammamet, Tunisia, 18-20 Dec 2016.
  - Ismat Chaib draa, **Maroua Nouiri**, Smail Niar, Abdelghani Bekrar. Device Context Classification for Mobile Power Consumption Reduction. *Digital System Design,*(DSD), Limassol, Cyprus, 31 Octobre-2 Septembre 2016.
  - **Maroua Nouiri**, Abdelghani Bekrar, Abderrazak Jemai, Damien Trentesaux, Ahmed Chiheb Ammari, Smail Niar. An improved multi agent particle swarm optimization to solve Flexible job shop scheduling problem. *In International Conference on Computers & Industrial Engineering, (CIE45)*, Metz, France, 28-30 octobre 2015.
  - **Maroua Nouiri**, Abderezak Jemai, Ahmed Chiheb Ammari, Abdelghani Bekrar, Smail Niar. An effective particle swarm optimization algorithm for Flexible job-shop scheduling problem. *In International Conference on Industrial Engineering and Systems Management (IESM)*, IEEE, Rabat, Marocco, 28-30 Oct 2013.

Bibliographie

[ABPR02] Christian Artigues, Cyril Briand, Marie-Claude Portmann, and François Roubellat. Pilotage d'atelier basé sur un ordonnancement flexible. In *Méthodes du pilotage des systèmes de production*. Hermes Lavoisier, 2002.

[AEJG12] Ameni Azzouz, Meriem Ennigrou, Boutheina Jlifi, and Khaléd Ghédira. Combining tabu search and genetic algorithm in a multi-agent system for solving flexible job shop problem. In *11th Mexican International Conference on Artificial Intelligence (MICAI), IEEE*, pages 83–88, San Luis Potosi, Mexico, 27 oct-4 Nov 2012.

[AHE11] Nasr Al-Hinai and Tarek Y ElMekkawy. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2) :279–291, 2011.

[AHE12] Nasr Al-Hinai and Tarek Y ElMekkawy. Solving the flexible job shop scheduling problem with uniform processing time uncertainty. *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, 6(4) :848–853, 2012.

[AS97] Raida Jarrar Abumaizar and Joseph A Svestka. Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35(7) :2065–2082, 1997.

[AZ10] Leila Asadzadeh and Kamran Zamanifar. An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. *Mathematical and Computer Modelling*, 52(11) :1957–1965, 2010.

[BD93] Roy Bernard and Bouyssou Denis. Aide multicritère à la décision : Méthodes et cas. *Economica*, 1993.

[BFCFdM09] Carmelo JA Bastos-Filho, Danilo F Carvalho, Elliackin MN Figueiredo, and Péricles BC de Miranda. Dynamic clan particle swarm optimization. In *Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09, IEEE*, pages 249–254, Pisa, Italy, 30 Nov-3 Dec 2009.

[BGWT09] Jun-Jie Bai, Yi-Guang Gong, Ning-Sheng Wang, and Dun-Bing Tang. An improved pso algorithm for flexible job shop scheduling with lot-splitting. In *International Workshop on Intelligent Systems and Applications, ISA, IEEE*, pages 1–5, Wuhan, China, 23-24 May 2009.

[BMS05] Jean-Charles Billaut, Aziz Moukrim, and Eric Sanlaville. *Flexibilité et robustesse en ordonnancement*. Hermes Science, 2005.

[BMZ95] Bernadette Bouchon-Meunier and Lotfi Asker Zadeh. *La logique floue et ses applications*. Editions Addison-Wesley France, 1995.

[BOP88] Donald L Byrkett, Mufit H Ozden, and Jon M Patton. Integrating flexible manufacturing systems with traditional. *Production and Inventory Management Journal*, 29(3) :15, 1988.

[Bra93] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3) :157–183, 1993.

[BT06] Anna Bonfill Teixidor. *Proactive management of uncertainty to improve scheduling robustness in process industries*. PhD thesis, Universitat Politècnica de Catalunya, 2006.

[BVLB09] Julien Bidot, Thierry Vidal, Philippe Laborie, and J Christopher Beck. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12(3) :315–344, 2009.

[CC88] Jacques Carlier and Philippe Chrétienne. *Problèmes d'ordonnancement : modélisation, complexité, algorithmes*. Masson, 1988.

[CCAT14] Tarek Chaari, Sondes Chaabane, Nassima Aissani, and Damien Trentesaux. Scheduling under uncertainty : Survey and research directions. In *International Conference on Advanced Logistics and Transport (ICALT), IEEE*, pages 229–234, Hammamet, Tunisia, 1-3 May,2014.

[CK02] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1) :58–73, 2002.

[CLW04] YAN Chen, Zeng-Zhi Li, and Zhi-Wen Wang. Multi-agent based genetic algorithm for jssp. In *International Conference on Machine Learning and Cybernetics, IEEE*, pages 267–270, Shanghai, China,26-29 Aug 2004.

[CMM03] Richard W Conway, William L Maxwell, and Louis W Miller. *Theory of scheduling*. Courier Corporation, 2003.

[COP04] Peter I Cowling, Djamila Ouelhadj, and Sanja Petrovic. Dynamic scheduling of steel casting and milling using multi-agents. *Production Planning & Control*, 15(2) :178–188, 2004.

[DB00] Andrew J Davenport and J Christopher Beck. A survey of techniques for scheduling with uncertainty. *http ://tidel.mie.utoronto.ca/publications.php*, 2000.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant colony system : a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1) :53–66, 1997.

[DJ12] Yao-Hsiang Dong and Jaejin Jang. Production rescheduling for machine breakdown at a job shop. *International Journal of Production Research*, 50(10) :2681–2691, 2012.

[DM12] Vahid Majazi Dalfard and Ghorbanali Mohammadi. Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints. *Computers & Mathematics with Applications*, 64(6) :2111–2117, 2012.

[EG04] Meriem Ennigrou and Khaled Ghédira. Approche multi-agents basée sur la recherche tabou pour le job shop flexible. *14ème congrès francophone de reconnaissance des formes et intelligence artificielle*, 2004.

[EG08] Meriem Ennigrou and Khaled Ghédira. New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach. *Autonomous Agents and Multi-Agent Systems*, 17(2) :270–287, 2008.

[EKBA17] Fatima El Khoukhi, Jaouad Boukachour, and Ahmed El Hilali Alaoui. The dual-ants colony : A novel hybrid approach for the flexible job shop scheduling problem with preventive maintenance. *Computers & Industrial Engineering*, 106 :236–255, 2017.

[EL99] Patrick Esquirol and Pierre Lopez. *L'ordonnancement*. Economica, 1999.

[FF10] Parviz Fattahi and Alireza Fallahi. Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. *CIRP Journal of Manufacturing Science and Technology*, 2(2) :114–123, 2010.

[GJ09] BS Girish and Natarajan Jawahar. A particle swarm optimization algorithm for flexible job shop scheduling problem. In *IEEE International Conference on Automation Science and Engineering, CASE*, pages 298–303, Bangalore, India,22-25 Aug 2009.

[Glo89] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3) :190–206, 1989.

[Glo90] Fred Glover. Tabu search-part ii. *ORSA Journal on computing*, 2(1) :4–32, 1990.

[GSG08] Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9) :2892–2907, 2008.

[GST+15] Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Mehmet Fatih Tasgetiren, Quan Ke Pan, and Qiang Qiang Sun. Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. *Computers & Industrial Engineering*, 90 :107–117, 2015.

[HE13] Abir Henchiri and Meriem Ennigrou. Particle swarm optimization combined with tabu search in a multi-agent model for flexible job shop problem. In *International Conference in Swarm Intelligence*, pages 385–394, Harbin, China, 12-15 June 2013.

[HL05] Willy Herroelen and Roel Leus. Project scheduling under uncertainty : Survey and research potentials. *European journal of operational research*, 165(2) :289–306, 2005.

[Hol92] John H Holland. *Adaptation in natural and artificial systems : An introductory analysis with application to biology, control, and artificial intelligence*. MIT press, 1992.

[HSL13] Wei He, Dihua Sun, and Xiaoyong Liao. Applying novel clone immune algorithm to solve flexible job shop problem with machine breakdown. *Journal of Information & Computational Science*, 10(9) :2783–2797, 2013.

[J+01] Mikkel T Jensen et al. *Robust and flexible scheduling with evolutionary computation*. BRICS, 2001.

[JCT07] Zhaohong Jia, Huaping Chen, and Jun Tang. An improved particle swarm optimization for multi-objective flexible job-shop scheduling problem. In *IEEE International Conference on Grey Systems and Intelligent Services, GSIS*, pages 1587–1592, Nanjing, China, 18-20 Nov 2007.

[Jen03] Mikkel T Jensen. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on evolutionary computation*, 7(3) :275–288, 2003.

[JLDWS94] V Jorge Leon, S David Wu, and Robert H Storer. Robustness measures and robust scheduling for job shops. *IIE transactions*, 26(5) :32–43, 1994.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, 1995.

[Ken99] James Kennedy. Small worlds and mega-minds : effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, volume 3, pages 1931–1938. IEEE, Washington, DC, USA, 6-9 July 1999.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, 1983.

[KHB02] Imed Kacem, Slim Hammadi, and Pierre Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(1) :1–13, 2002.

[KHL86] J Kaltwasser, A Hercht, and R Lang. Hierarchical control of flexible manufacturing systems. *IFAC Information Control Problems in Manufacturing Technology, Suzdal, USSR*, pages 37–44, 1986.

[KVR15] Ketrina Katragjini, Eva Vallada, and Rubén Ruiz. Rescheduling flowshops under simultaneous disruptions. In *International Conference on Industrial Engineering and Systems Management (IESM), IEEE*, pages 84–91, Seville, Spain, 21-23 Oct 2015.

[La05] Hoang Trung La. *Utilisation d'ordres partiels pour la caractérisation de solution robustes en ordonnancement*. PhD thesis, INSA de Toulouse, 2005.

[LAR07] Ning Liu, Mohamed A Abdelrahman, and Srini Ramaswamy. A complete multiagent framework for robust and adaptable dynamic job shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(5) :904–916, 2007.

[LGX07] Lin Liu, Han-yu Gu, and Yu-geng Xi. Robust and stable scheduling of a single machine with random machine breakdowns. *The International Journal of Advanced Manufacturing Technology*, 31(7) :645–654, 2007.

[LP91] Claude Le Pape. *Constraint propagation in planning and scheduling*. 1991.

[LPX+10] Jun-qing Li, Quan-ke Pan, Sheng-xian Xie, Bao-xian Jia, and Yu-ting Wang. A hybrid particle swarm optimization and tabu search algorithm for flexible job-shop scheduling problem. *International Journal of Computer Theory and Engineering*, 2(2) :189, 2010.

[Mat96] Dirk C Mattfeld. *Evolutionary search and the job shop : investigations on genetic algorithms for production scheduling*. 1996.

[MCP10] Wiem Mouelhi-Chibani and Henri Pierreval. Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58(2) :249–256, 2010.

[MK03] Rui Mendes and James Kennedy. Avoiding the pitfalls of local optima : how topologies can save the day. 2003.

[ML93] BL MacCarthy and Jiyin Liu. A new classification scheme for flexible manufacturing systems. *The International Journal of Production Research*, 31(2) :299–309, 1993.

[MlSlH10] Arash Motaghedi-larijani, Kamyar Sabri-laghaie, and Mahdi Heydari. Solving flexible job shop scheduling with multi objective approach. *International Journal of Industrial Engineering & Production Research*, 21(4) :197–209, 2010.

[MU99] SV Mehta and RH Uzsoy. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1) :15–38, 1999.

[NBJ+15a] Maroua Nouiri, Aabdelghani Bekrar, Abderrazak Jemai, Damien Trentesaux, AChiheb Ammari, and Smail Niar. An improved multi agent particle swarm optimization to solve flexible job shop scheduling problem. In *International Conference on Computers & Industrial Engineering, (CIE45)*, Metz, France, 28–30 octobre 2015.

[NBJ+15b] Maroua Nouiri, Abdelghani Bekrar, Abderezak Jemai, Smail Niar, and Ahmed Chiheb Ammari. An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, pages 1–13, 2015.

[NBJ+17] Maroua Nouiri, Abdelghani Bekrar, Abderrazak Jemai, Damien Trentesaux, Ahmed Chiheb Ammari, and Smail Niar. Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. *Computers & Industrial Engineering*, 2017.

[NJA+13] Maroua Nouiri, Abderezak Jemai, Ahmed Chiheb Ammari, Abdelghani Bekrar, and Smail Niar. An effective particle swarm optimization algorithm for flexible job-shop scheduling problem. In *International Conference on Industrial Engineering and Systems Management (IESM), IEEE*, pages 1–6, Rabat, Marocco, 28-30 Oct 2013.

[NJA+16] Maroua Nouiri, Abderrazak Jemai, Ahmed Chiheb Ammari, Abdelghani Bekrar, Damien Trentesaux, and Smail Niar. Using iot in breakdown tolerance : Pso solving fjsp. In *11th International Design & Test Symposium (IDT), IEEE*, pages 19–24, Hammamet, Tunisia, 18-20 Dec 2016.

[PAL+13] Humberto Díaz Pando, Sergio Cuenca Asensi, Roberto Sepúlveda Lima, Jenny Fajardo Calderín, and Alejandro Rosete Suárez. An application of fuzzy logic for hardware/software partitioning in embedded systems. *Computación y Sistemas*, 17(1) :25–39, 2013.

[PB01] Laure Pichot and Pierre Baptiste. Ordonnancement des ressources humaines : étude de cas d'une entreprise d'injection plastique. *conference francophone de Modélisation et Simulation "Conception, Analyse et Gestion des Systemes Industriels", MOSIM'01*, pages 707–714, Troyes, France, 25-27 Avril 2001.

[PC94] Young S Pyoun and Byoung K Choi. Quantifying the flexibility value in automated manufacturing systems. *Journal of Manufacturing Systems*, 13(2) :108–118, 1994.

[PMC08] F Pezzella, G Morganti, and G Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10) :3202–3212, 2008.

[PVS09] Parviz Palangpour, Ganesh K Venayagamoorthy, and Scott C Smith. Particle swarm optimization : A hardware implementation. In *International Conference on Computer Design, CDES*, pages 134–139, Las Vegas Nevada, USA, 13-16 July 2009.

[PYY13] Feng Shan Pan, Chun Ming Ye, and Jiao Yang. Flexible job-shop scheduling problem under uncertainty based on qpso algorithm. In *Advanced Materials Research*, volume 605, pages 487–492, 2013.

[SLW+15] Lu Sun, Lin Lin, Yan Wang, Mitsuo Gen, and Hiroshi Kawakami. A bayesian optimization-based evolutionary algorithm for flexible job shop scheduling. *Procedia Computer Science*, 61 :521–526, 2015.

[SM16] Manas Ranjan Singh and Siba Sankar Mahapatra. A quantum behaved particle swarm optimization for flexible job shop scheduling. *Computers & Industrial Engineering*, 93 :36–44, 2016.

[Sno01] Marko Snoek. Anticipation optimization in dynamic job shops. In *Evolutionary Algorithms for Dynamic Optimization Problems*, pages 43–46, San Francisco, California, USA, 7 Jully 2001.

[SOS93] Norman Sadeh, Shinichi Otsuka, and Robert Schnelbach. Predictive and reactive scheduling with the micro-boss production scheduling and control system. In *Proceedings, IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling and Control*, 1993.

[SR03] Velusamy Subramaniam and Amritpal Singh Raheja. maor : A heuristic-based reactive repair mechanism for job shop schedules. *The International Journal of Advanced Manufacturing Technology*, 22(9) :669–680, 2003.

[SRS12] María Soto, André Rossi, and Marc Sevaux. A mathematical model and a metaheuristic approach for a memory allocation problem. *Journal of Heuristics*, 18(1) :149–167, 2012.

[SSH13] Mehdi Souier, Zaki Sari, and Ahmed Hassam. Real-time rescheduling metaheuristic algorithms applied to fms with routing flexibility. *The International Journal of Advanced Manufacturing Technology*, pages 1–20, 2013.

[TPB+13] Damien Trentesaux, Cyrille Pach, Abdelghani Bekrar, Yves Sallez, Thierry Berger, Thérèse Bonte, Paulo Leitão, and José Barbosa. Benchmarking flexible job-shop scheduling and control systems. *Control Engineering Practice*, 21(9) :1204–1225, 2013.

[Tre03] Ioan Cristian Trelea. The particle swarm optimization algorithm : convergence analysis and parameter selection. *Information processing letters*, 85(6) :317–325, 2003.

[TZLZ11] Jianchao Tang, Guoji Zhang, Binbin Lin, and Bixi Zhang. A hybrid algorithm for flexible job-shop scheduling problem. *Procedia Engineering*, 15 :3678–3683, 2011.

[UG09] Prakarn Unachak and Erik Goodman. Adaptive representation for flexible job-shop scheduling and rescheduling. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 511–516. ACM, Shanghai, China, 12-14 June 2009.

[VHL03] Guilherme E Vieira, Jeffrey W Herrmann, and Edward Lin. Rescheduling manufacturing systems : a framework of strategies, policies, and methods. *Journal of scheduling*, 6(1) :39–62, 2003.

[VSS06] Gerhard Venter and Jaroslaw Sobieszczanski-Sobieski. Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, 3(3) :123–137, 2006.

[WCCC09] LH Wu, Xin Chen, XD Chen, and Qing Xin Chen. The research on proactive-reactive scheduling framework based on real-time manufacturing information. In *Materials Science Forum*, volume 626, pages 789–794, August 2009.

[WD12] Xiong Wei and Fu Dongmei. Multi-agent system for flexible job-shop scheduling problem based on human immune system. In *IEEE Proceedings of the 31st Chinese Control Conference*, pages 2476–2480, Hefei, China, 25-27 July 2012.

[WSPC93] S David Wu, Robert H Storer, and Chang Pei-Chann. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20(1) :1–14, 1993.

[WX08] Yu-Xuan Wang and Qiao-Liang Xiang. Particle swarms with dynamic ring topology. In *IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 419–423, Hong Kong, China, 1-6 June 2008.

[XW05] Weijun Xia and Zhiming Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2) :409–425, 2005.

[XXC13] Jian Xiong, Li-ning Xing, and Ying-wu Chen. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics*, 141(1) :112–126, 2013.

[Zad65] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3) :338–353, 1965.

[ZGS11] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4) :3563–3573, 2011.

[ZPST12] Nadine Zbib, Cyrille Pach, Yves Sallez, and Damien Trentesaux. Heterarchical production control in manufacturing systems using the potential fields concept. *Journal of Intelligent Manufacturing*, 23(5) :1649–1670, 2012.

[ZSLG09] Guohui Zhang, Xinyu Shao, Peigen Li, and Liang Gao. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 56(4) :1309–1318, 2009.