Ordonnancement dynamique dans les industries agroalimentaires

Fatma Tangour

N° d'ordre : 51

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
ÉCOLE CENTRALE DE LILLE

UNIVERSITÉ DE TUNIS EL MANAR
ÉCOLE NATIONALE D'INGÉNIEURS DE TUNIS

THÈSE

présentée par

Fatma TANGOUR TOUMI

pour l'obtention du grade de

DOCTEUR

en Automatique et Informatique Industrielle

Doctorat délivré conjointement par l'Université des Sciences et Technologies de Lille, L'École Centrale de Lille et l'École Nationale d'Ingénieurs de Tunis

Ordonnancement Dynamique dans les Industries Agroalimentaires

soutenue le 12 juillet 2007 devant le Jury d'Examen composé de :

MM.	Pr.	Noureddine	**ELLOUZE**	Président
	Pr.	Ennaceur	**BENHADJ BRAIEK**	Rapporteur
	Pr.	Abdallah	**ELMOUDNI**	Rapporteur
	Pr.	Mohamed	**BENREJEB**	Examinateur
	Pr.	Pierre	**BORNE**	Examinateur
	Pr.	Slim	**HAMMADI**	Examinateur

Thèse préparée au **L**aboratoire d'**A**utomatique **G**énie **I**nformatique et **S**ignal de l'École Centrale de Lille et à l'Unité de Recherche **LARA**-Automatique de l'École Nationale d'Ingénieurs de Tunis

Avant- propos

Le travail faisant l'objet du présent mémoire a été effectué au sein du Laboratoire dru Recherche en Automatique (UR-LARA) de l'Ecole Nationale d'Ingénieurs de Tunis (ENIT) et du Laboratoire d'Automatique, Génie Informatique et Signal (LAGIS) de l'Ecole Centrale de Lille (EC-Lille).

Nous tenons à exprimer notre vive gratitude à Monsieur Noureddine ELLOUZE, Professeur et Directeur du Laboratoire de Systèmes et Traitement du Signal (LSTS) de l'ENIT, pour nous avoir fait le grand honneur d'accepter de présider le Jury d'Examen. Qu'il trouve ici l'expression de notre profond respect.

C'est un agréable devoir pour nous d'exprimer notre très vive reconnaissance à Monsieur le Professeur Mohamed BENREJEB, Directeur de l'Unité de Recherche UR-LARA à l'ENIT, à Monsieur Pierre BORNE, Professeur à l'EC-Lille, et à Monsieur Slim HAMMADI, Professeur à l'EC-Lille pour nous avoir guider durant toute l'élaboration de ce mémoire avec le sérieux et la compétence qui les caractérisent. Qu'ils trouvent ici le témoignage de notre très profonde gratitude.

Nous tenons à remercier vivement Monsieur Naceur BENHADJ BRAIEK, Professeur et Directeur de l'Unité de Recherche UR-LECAP à l'Ecole Polytechnique de Tunis (EPT), d'avoir accepté de rapporter notre travail. Qu'il soit grandement remercié.

Notre profonde gratitude à Monsieur Abdellah EL MOUDNI, Professeur à l'Université de Technologie Belfort-Monbéliard, pour avoir bien voulu rapporter sur nos travaux de Thèse. Nous lui adressons nos sincères remerciements.

Nous tenons, enfin, à remercier tous les chercheurs du Laboratoire de Recherche en Automatique de l'ENIT et du Laboratoire d'Automatique, Génie Informatique et Signal de l'EC-Lille et toute autre personne qui, par leur aide, par leur amicale présence et par leur sympathie, ont rendu ce travail agréable. Nous leur exprimons ici toute notre gratitude.

À la mémoire de mon père

À ma mère

À mon mari et à mon fils

À mes beaux-parents

À toute ma famille

Table des matières

Liste des figures

Introduction générale

Le type et la nature de la consommation actuelle sont, entre autres, caractérisés par une forte demande de produits personnalisés à des prix bas. En effet, l'évolution et la mondialisation de la production ainsi que l'ouverture des marchés internationaux ont poussé les industriels à se diriger vers des systèmes de fabrication flexibles ; ceci conduit à mettre en place une logique industrielle qui remet en cause certains fondements des habitudes de la production. Ainsi, la gestion des ateliers de production a pris une certaine valeur ces dernières années afin d'améliorer la productivité et d'augmenter la réactivité. Ces nouveaux domaines d'application rendent plus complexes les problèmes d'ordonnancement des tâches ; ce qui exige la résolution rigoureuse de ces problèmes. En effet, par une meilleure utilisation et planification des ressources, un atelier de production peut réaliser une grande variété de produits et les coûts peuvent être réduits ce qui constitue un objectif crucial. En particulier, la productivité peut être affectée par l'ordonnancement des opérations sur les machines. Le champ d'application de l'ordonnancement est large : la gestion de la production dans l'industrie, la gestion de projets, l'organisation des emplois du temps, la gestion de la charge des processeurs en informatique, etc. En raison de leur nature fortement combinatoire, l'étude des problèmes d'ordonnancement reste également d'un intérêt théorique toujours renouvelé, car il n'y a pas encore trouvé de méthode de résolution à la fois générale et de faible complexité algorithmique.

Les problèmes d'ordonnancement, très variés, sont caractérisés par un grand nombre de paramètres relatifs aux tâches (préemptives ou non, indépendantes ou non), aux ressources (renouvelables ou consommables), aux types de contraintes portant sur les tâches (précédences, disjonctions), au(x) critère(s) d'optimalité (minimisation de la durée de l'ordonnancement, minimisation du retard maximal des tâches, etc.).

Résoudre un problème d'ordonnancement consiste à organiser ces tâches, c'est-à-dire à déterminer leurs dates de démarrage, et à leur attribuer des ressources, de telle sorte que les contraintes soient respectées, afin d'optimiser un certain objectif.

La nature des produits et des procédés ainsi que les contraintes de qualité et de contrôle des coûts font que les besoins des industriels de l'agroalimentaire en matière de gestion des données techniques sont assez différents de ceux des entreprises manufacturières. Ainsi, l'évolution et les caractéristiques dynamiques des ateliers industriels, en particulier ceux des industries agroalimentaires, imposent la génération, en temps réel, d'une décision du processus d'ordonnancement. En général, les problèmes d'ordonnancement sont des problèmes d'optimisation multicritères ou muti-objectifs. Ainsi, la meilleure solution, appelée aussi solution optimale, est la solution ayant obtenue la meilleure évaluation au regard du ou des critère(s) défini(s).

La résolution d'un problème d'ordonnancement est constituée par deux étapes principales. La première étape consiste à identifier et à modéliser le problème en décrivant les contraintes qui doivent être respectées et mettant en valeur les critères à optimiser. La deuxième étape se traduit par la recherche de la méthode adéquate pour résoudre le problème considéré. En effet, l'exploitation de plusieurs méthodes est importante afin de trouver une solution qui permet, par la comparaison de leur efficacité d'établir des décisions robustes.

Dans cette optique, ce travail de thèse propose des solutions pour le problème d'ordonnancement multi-objectif adapté aux industries agroalimentaires. Les contraintes et des critères considérés sont spécifiques à ce type d'industrie. Ce domaine présente certaines particularités dues à la nature des produits manipulés et fabriqués. Ces produits ont des durées de vie assez courtes. Ainsi, les contraintes et les objectifs retenus par le système de gestion de production mis en œuvre sont liés à ces particularités. En effet, le respect des dates de validité des composants primaires formant les opérations et des produits semi-finis est une contrainte absolue à respecter. Ainsi, la péremption d'un composant engendre une perte matérielle qui se traduit par le prix de revient de ce composant, et cette péremption génère aussi l'empêchement de la réalisation de l'opération prévue induisant des retards de livraison des produits finis. Ces produits finis ont aussi une durée de vie limitée et leurs prix se dégradent proportionnellement aux jours de stockage avant leur livraison. En effet, les surfaces de distribution imposent aux industriels une pénalité, dite discount de distribution, par jour de stockage du produit fini. L'un des objectifs à atteindre dans notre travail est donc de minimiser le discount de distribution. Il faut noter aussi que les produits agroalimentaires sont à caractères saisonnier et irrégulier. Le système de production doit alors prévoir un moyen pour couvrir les variations de la demande à travers des substitutions dans les gammes de produits finis.

Dans le premier chapitre, les problèmes d'ordonnancement sont introduits. Les types d'ateliers ainsi que les paramètres nécessaires pour caractériser un problème d'ordonnancement sont aussi évoqués. Les problèmes d'optimisation et les leurs méthodes de résolution, à savoir les méthodes exactes et les méthodes approchées et la complexité de la résolution de ces problèmes, sont également abordés. Les particularités de l'ordonnancement en industries agroalimentaires sont présentées.

Le deuxième chapitre repose sur la méthode branch & bound et son application pour optimiser deux critères spécifiques à l'industrie agroalimentaire la fonction de coût qui sont : le coût des produits périmés et le coût du discount de distribution. C'est une Procédure de Séparation et d'Évaluation progressive PSE, qui est une méthode exacte d'optimisation basée sur une énumération contrôlée qui fournit, en général, une solution optimale. Elle consiste à construire une arborescence dont la racine correspond à l'espace des solutions du problème initial. Cette procédure est basée sur deux étapes. Une étape de filtration qui consiste à éliminer certaines opérations de l'espace de recherche initial en appliquant des règles de dominance issues des paramètres des composants formant une opération, tels que la date de validité de ces composants et les durées opératoires. Ainsi, la décision d'éliminer ou de maintenir une opération de l'espace de recherche initial, permet d'éviter la péremption de certains composants et donc la minimisation des coûts des produits périmés.

Dans le troisième chapitre, deux métaheuristiques, à savoir les algorithmes génétiques et les algorithmes d'optimisation par colonie de fourmis ont été appliquées pour construire un ordonnancement dynamique multicritère, en optimisant les deux critères cités et un troisième critère plus classique qui représente la date de fin de l'ordonnancement, le C_{max}.

Notations

O_{ij} : $j^{ème}$ opération du produit i

t_{ij} : date effective de début de fabrication de l'opération O_{ij}

r_{ij} : date de début au plus tôt de l'opération O_{ij}

g_{ij} : date de fin effective de l'opération O_{ij}

p_{ij} : durée opératoire de l'opération O_{ij}

P_i : produit fini de l'opération O_{ij}

c_{ijk} : $k^{ème}$ composant de l'ensemble des composants de l'opération O_{ij}

v_{ijk} : date de validité limite du composant c_{ijk}

C_{P_i} : date de fin de fabrication du produit P_i

$d_{P_i}^{liv}$: date de livraison du produit P_i

d_{fP} : date de fin de la séquence P

Dv_{P_i} : durée de vie du produit P_i

Dr_{P_i} : délai de retour du produit P_i

P_{ijk}^{rev} : prix de revient du composant c_{ijk}

$P_{P_i}^{ven}$: prix de vente unitaire du produit P_i

$C_{P_i}^{stk}$: coût du stockage, par unité de temps, d'une unité du produit P_i

Chapitre 1

Ordonnancement : état de l'art et particularités en industries agroalimentaires

1.1. Introduction

L'ordonnancement représente un domaine d'application de la recherche opérationnelle et de la gestion de la production qui vise à améliorer l'efficacité d'une entreprise en termes de coût de production et de délai de livraison.

Les problèmes d'ordonnancement apparaissent dans tous les domaines : informatique [Bep96], industrie, transport, construction, administration... [Car88].

Résoudre un problème d'ordonnancement consiste à organiser ou ordonnancer des tâches, c'est-à-dire à déterminer leurs dates de démarrage et à leurs attribuer des ressources matérielles ou humaines nécessaires, de telle sorte que les contraintes soient respectées, afin d'optimiser un certain objectif préalablement défini [Got93], [Lop01]. Cette planification des ressources pour réaliser une grande variété de produits finis dans des délais impératifs, est un garant de la compétitivité d'une entreprise. C'est pour ces raisons que les problèmes d'ordonnancement deviennent plus complexes et plus importants.

Ainsi le résultat de la résolution d'un problème d'ordonnancement se décompose en trois étapes fondamentales :

- l'allocation des ressources nécessaires pour chaque tâche,
- le séquencement qui précise l'ordre de passage des tâches sur chaque ressource,
- l'affectation d'une date de début et d'une date de fin pour chaque tâche.

Cette résolution passe, alors, par deux phases nécessaires :

- une phase d'identification et de modélisation,
- une phase de recherche de la méthode adéquate.

Dans notre travail, on s'intéresse aux problèmes d'ordonnancement d'atelier.

1.2. Problèmes d'ordonnancement : état de l'art

1.2.1. Caractérisation

Les problèmes d'ordonnancement d'atelier consistent à affecter plusieurs tâches à des moyens de fabrication afin de réaliser des produits (travaux ou jobs) tout en respectant les contraintes de fabrication et en optimisant certains critères.

Les principaux éléments d'un problème d'ordonnancement sont les suivants : les tâches (ou opérations), les ressources, les contraintes et les critères.

1.2.1.1. Les tâches

Les tâches représentent toutes les opérations à effectuer pour la fabrication des produits. Une tâche, c'est à dire un ensemble d'opérations, requiert pour son exécution certaines ressources qu'il faut programmer de façon à optimiser un certain objectif. Si l'ensemble des tâches à exécuter au cours du temps est donné a priori, c'est-à-dire leurs dates (date de début et date de fin) et leurs ordres d'exécution sont connus à l'avance, le problème d'ordonnancement est dit *statique.* Dans le cas contraire, si l'ensemble des tâches à exécuter évolue dans le temps, le problème est dit *dynamique.*

Dans notre travail, on s'intéresse aux problèmes d'ordonnancement dynamique.

Deux types de tâches sont distingués :

- les tâches morcelables (préemptives) qui peuvent être exécutées en plusieurs fois facilitant ainsi la résolution de certains problèmes,
- les tâches non morcelables (indivisibles) qui sont exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles soient complètement terminées.

Chaque tâche i qui correspond à la réalisation d'un produit (ou job) j est alors caractérisée par :

- t_{ijk} : sa date de début d'exécution sur la machine k,
- r_{ijk} : sa date de disponibilité ou date de début au plus tôt sur la machine k,
- p_{ijk} : son temps d'exécution ou sa durée opératoire sur la machine k,
- C_{ijk} : sa date de fin d'exécution sur la machine k,
- d_{ijk} : sa date de fin au plus tard sur la machine k.

Ainsi, pour qu'un ordonnancement soit réalisable, la condition suivante est nécessaire :

$$\forall i \in I,\ r_{ijk} \le t_{ijk} \le C_{ijk} \le d_{ijk}$$

où I représente l'ensemble de tâches à ordonnancer.

1.2.1.2. Les gammes

Une gamme de fabrication précise l'ordre de passage des opérations permettant la réalisation du produit sur l'ensemble des machines ainsi que le nombre d'opérations relatives au produit. Si le choix des types de machines pour les opérations n'est pas imposé, la gamme est dite *logique*, les entités qui interviennent sont les opérations. Suivant l'ordre d'exécution des opérations pour achever un produit, trois types de gammes sont distingués :

- la gamme libre : l'ordre est totalement libre,
- la gamme linéaire : l'ordre est entièrement déterminé,
- la gamme mixte : l'ordre est partiellement déterminé.

1.2.1.3. Les ressources

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. Ce moyen technique est donc nécessaire et indispensable pour le bon fonctionnement du cycle de fabrication.

Dans un atelier, plusieurs types de ressources sont distingués :

- les ressources renouvelables, qui, après avoir été allouées à une tâche, redeviennent disponibles et qui peuvent être réutilisées (machines, personnel, etc.),
- les ressources consommables, qui, après avoir été allouées à une tâche, ne sont plus disponibles, et sont donc épuisées (argent, matières premières, etc.),
- les ressources partageables qui peuvent être partagées entre plusieurs tâches.

Ces ressources peuvent être classées d'une autre manière :

- les ressources de type disjonctif qui ne peuvent exécuter qu'une opération ou une tâche à la fois,
- les ressources de type cumulatif qui peuvent exécuter plusieurs opérations simultanément.

Une machine est considérée comme un type de ressource qui n'est caractérisé que par son horaire de travail [Mes99].

1.2.1.4. Les contraintes

Les contraintes représentent les conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter lors de la résolution de ces problèmes.

Ces contraintes peuvent être classées en deux types : endogène et exogène.

- ***Les contraintes endogènes***

Elles constituent des contraintes liées directement au système de production et à ses performances telles que :

- les dates de disponibilité des machines et des moyens de transport,
- les capacités des machines et des moyens de transport,
- les séquences des actions à effectuer ou les gammes des produits.

- ***Les contraintes exogènes***

Ces contraintes sont imposées extérieurement. Et sont indépendantes du système de production ; on distingue :

- les dates de fin de fabrication au plus tard du produit imposées généralement par les commandes,
- les priorités de quelques commandes et de quelques clients,
- les retards possibles accordés pour certains produits.

Une autre classification consiste à distinguer :

- les contraintes de gamme ou de précédence, caractérisant l'ordre d'exécution des tâches selon la gamme,
- les contraintes de capacités, caractérisées par un ensemble de conflits pour l'utilisation des ressources, pouvant se diviser en :
  - contraintes disjonctives où une seule ressource est partagée par deux ou plusieurs tâches à la fois,
  - contraintes cumulatives où il y'a partage de plusieurs ressources par plusieurs tâches,
- les contraintes liées directement aux produits finis (dates de livraison des commandes, priorités, dates de péremption, etc.) [Gar03].

1.2.1.5. Les critères

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettant d'évaluer la qualité de l'ordonnancement établi.

- Les critères usuels sont :
  - la durée totale de l'ordonnancement, le makespan $C_{\max}$
  - le respect des dates au plus tard,
  - la minimisation d'un coût, etc.
- Les critères réguliers sont :
  - la minimisation de la date de fin effective de la dernière opération de la gamme,
  - la minimisation des dates effectives de fin de toutes les opérations,
  - la minimisation du plus grand retard, etc.
- Les critères irréguliers sont :
  - l'équilibre de charge des ressources,
  - l'optimisation des changements d'outils, etc. [Gar03].

Les différents critères ne sont pas indépendants, certains même sont équivalents. Par définition, deux critères sont équivalents si une solution optimale pour l'un est aussi optimale pour l'autre et inversement. Par exemple, la moyenne des dates de fin est équivalente au retard algébrique moyen [Car88].

1.2.2. Types d'ateliers

La classification des types de problèmes d'ordonnancement se fait en fonction du type des machines utilisées ou selon l'organisation de l'atelier étudié.

1.2.2.1. Le type flow-shop

Les ateliers de type « flow-shop » pour lequel la ligne de fabrication est constituée de plusieurs machines en série ; toutes les opérations de toutes les tâches passent par toutes les machines dans le même et unique ordre.

Ce type d'atelier est dit à cheminement unique. Dans les ateliers de type « flow-shop hybride », une machine peut exister en plusieurs exemplaires identiques et parallèles.

1.2.2.2. Le type job-shop

Dans les ateliers de type « job-shop », les opérations sont réalisées selon un ordre total bien déterminé, variant selon la tâche à exécuter. Ce type d'atelier est nommé aussi atelier à cheminements multiples. Dans ce cas, plusieurs changements d'outils sont à envisager.

- ***Le type job-shop flexible***

Le job-shop flexible est une extension du modèle job-shop classique. Sa particularité essentielle réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations. Plus précisément, une opération est associée à un ensemble contenant toutes les machines pouvant effectuer cette opération.

1.2.2.3. Le type open-shop

Aucun ordre de fabrication n'est imposé, dans ce cas ; l'acheminement de toutes les opérations est multiple et libre ; ces opérations peuvent être exécutées dans n'importe quel ordre.

Des extensions à ces types de base ont été définies dans le but de se rapprocher des ateliers réels de production. Par exemple, la flexibilité des machines qui consiste à associer à chaque opération un ensemble de machines c'est-à-dire que cette opération peut être traitée par une machine quelconque de cet ensemble. Un problème d'affectation est alors ajouté au problème initial d'ordonnancement.

Les types suivants sont distingués :

- les problèmes à « machines parallèles » où les opérations indépendantes et sont traitées par le même ensemble de machines,
- les problèmes de type «flow-shop hybride » constituant une extension du problème du type « flow-shop », où la première opération de chaque produit est traitée par le premier ensemble, la deuxième opération par le deuxième ensemble, et ainsi de suite.

Dans les problèmes d'ordonnancement avec affectation généralisée, pour chaque opération, un ensemble de machine est défini, une machine pouvant naturellement faire partie de plusieurs groupes, et une opération pouvant être traitée par n'importe quelle machine du groupe [Gza01].

1.2.3. Complexité

La complexité des problèmes d'ordonnancement est définie suivant la complexité des méthodes de résolution et celle des algorithmes utilisés [Gar79], [Car84a] [Car88], [Cha96]. Certains problèmes d'ordonnancement de taille relativement importante peuvent avoir un niveau de complexité si important que leur résolution devient très difficile.

Deux catégories de complexité sont distinguées : algorithmique et problématique

1.2.3.1. Complexité algorithmique

La théorie de la complexité a pour objectif d'analyser les coûts de résolution, notamment en terme de temps de calcul, des problèmes d'optimisation combinatoire [Coo71]. Elle permet d'établir une classification des problèmes en plusieurs niveaux de difficulté, et fait la distinction entre un problème d'optimisation et un problème de décision [Gra79] [Car88] [Cha96]. IL a été démontré que la plupart des problèmes d'ordonnancement sont difficiles [Lop01].

La complexité algorithmique se mesure par rapport au temps alloué pour l'exécution de l'algorithme ou encore par rapport à l'espace mémoire requis. Le temps de calcul est fonction du nombre d'instructions et de la taille des données manipulées.

La complexité en espace est une fonction qui associe à la taille d'une instance d'un problème donné, un ordre de grandeur du nombre de cases mémoires utilisées pour les opérations nécessaires à la résolution de ce problème [Pas05].
La durée d'un algorithme, notée $T(N)$, se déduit du nombre d'instructions élémentaires de cet algorithme ainsi que de la durée d'exécution d'une instruction. Cette dernière est majorée par un nombre qui est fonction de l'outil informatique et du programme utilisés.
Un algorithme est dit de complexité $O(f(N))$, s'il existe une constante K et un entier N_0 tels que pour tout $N \geq N_0$; $T(N) \leq Kf(N)$.
Un algorithme est dit polynomial d'ordre p si $f(N) = N^p$. Dans le cas où aucune fonction f polynomiale n'existe, l'algorithme est dit non Polynomial (NP) ou exponentiel.

1.2.3.2. Complexité problématique

La complexité problématique dépend du problème à résoudre ainsi que de la méthode de résolution choisie pour construire la solution optimale au sens des critères retenus. Dans la littérature, les problèmes d'ordonnancement sont souvent classés en deux catégories caractérisant leurs degrés de complexité :

- Les problèmes indécidables qui sont les problèmes d'ordonnancement les plus difficiles pour lesquels il n'existe aucune méthode de résolution connue,
- les problèmes décidables qui sont de classe *P* ou de classe *NP* [Cha96].

* *Définitions*

- Un problème de décision est un problème qui comprend deux parties : une partie données du problème et une question binaire ayant « oui » ou « non » comme réponse possible.
- Un problème de recherche est un problème constitué d'un ensemble de données et pour lequel à chaque ensemble de données correspond un ensemble de solutions. Résoudre un problème de recherche consiste à calculer pour chaque ensemble de données *D* l'ensemble des solutions S (*D*) associées.
- Un problème d'optimisation est un problème de recherche qui donne à chaque solution une valeur quantitative. On cherche une valeur minimale si on a une fonction économique à minimiser, ou maximale pour d'autres cas. À chaque problème d'optimisation on peut associer un problème de décision [Cha96]. Ainsi, l'étude de la complexité d'un problème de décision permet de donner les indications relatives au problème d'optimisation associé.
- Un problème de décision est de classe P s'il existe un algorithme polynomial pour le résoudre.
- Un problème de décision est de classe NP, il est dit également NP-difficile, s'il ne peut pas être résolu en un temps polynomial par les algorithmes déterministes [Car88], [Lop99] mais peut être résolu en un temps polynomial par des méthodes approchées.
- Un problème de décision est dit NP-complet s'il appartient à la classe NP et s'il est résolu, au mieux, en un temps exponentiel.
- Un problème d'optimisation est dit NP-difficile si le problème de décision associé est NP-complet.

1.2.4. Modélisation et représentation des problèmes d'ordonnancement

1.2.4.1. Notations

Une méthode de notation a été proposée dans la littérature [Gra79] [Bru95] pour représenter un problème d'ordonnancement d'une manière simple. Cette notation consiste à représenter le problème sous forme de trois champs *a*, *b* et *c* (a | *b* | *c*) :

- le champ *a* permet de décrire le type d'atelier, le nombre de jobs à réaliser et le nombre e machines disponibles,
- le champ *b* est utilisé pour préciser les contraintes du problème et les différentes hypothèses sur le mode d'exécution des tâches (préemption, précédence, etc.),
- le champ *c* est dédié au(x) critère(s) à optimiser.

Exemple

$$J, 10, 6 \mid Prec,\ r_i \mid C_{\max}$$

Il s'agit d'un problème d'ordonnancement d'un atelier de type job-shop J, de dix jobs à six machines. Le deuxième champ montre que les jobs présentent une contrainte de précédence, Prec, et une contrainte r_i de dates de début au plus tôt. En plus la préemption et interdite (n'est pas mentionnée dans le champ *prem*). Le dernier champ indique que l'objectif est de minimiser le makespan, $C_{\max}$.

1.2.4.2. Modélisation

La modélisation, est en général, une étape très importante dans la résolution d'un problème d'ordonnancement. C'est une écriture simplifiée de toutes les données du problème permettant d'en traduire tous les détails pour mieux représenter la réalité des choses.

Deux méthodes de modélisation sont distinguées : les méthodes graphiques et les méthodes mathématiques.

* **Les méthodes graphiques**

Un problème d'ordonnancement peut être modélisé par un graphe, dit graphe potentiel - tâches, figure 1.1, constitué par :

- des noeuds représentant les tâches,
- des arcs conjonctifs illustrant les contraintes de précédence (en indiquant les durées des tâches),
- des arcs disjonctifs indiquant les contraintes de ressources [Roy70], [Got93] et [Jai99].

Les méthodes graphiques ont connu une très importante évolution surtout avec l'apparition des Réseaux de Pétri (RdP) [Chr83], [Car84b] qui permettent de traduire plusieurs notions fondamentales qui ont un lien avec les problèmes d'ordonnancement telles que :

- les conflits sur les ressources,
- les durées opératoires certaines (RdP temporisés),
- les durées opératoires aléatoires (RdP stochastiques),
- les gammes,
- les disponibilités, les multiplicités et les capacités des ressources (RdP synchronisés et à capacités),
- la répétitivité (RdP cycliques).

Exemple [Kac03]

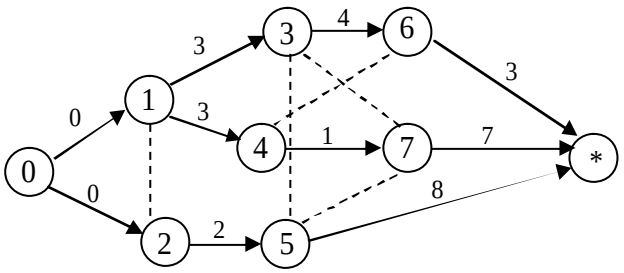


Fig.1.1. Graphe potentiel - tâches

* **La modélisation polyédrique**

Le principe de ce type de modélisation est de trouver les contraintes linéaires définissent des demi-espaces. L'intersection de ces demi-espaces donne un polyèdre dont les sommets représentent une solution réalisable du problème. Si la fonction objectif est linéaire, l'utilisation de la programmation linéaire permet d'obtenir la solution optimale. Cette approche s'est beaucoup développée dans le domaine de l'optimisation combinatoire [Gro88].

De nombreux problèmes comme le voyageur de commerce font intervenir ce genre de modélisation pour leur description [Law87].
Le modèle polyédrique pour des problèmes d'ordonnancement introduit par Balas [Bal85], définit un vecteur solution *t* dont les composantes sont les dates de début calculées de chaque opération à ordonnancer. *T* est alors l'ensemble de tous les vecteurs *t* vérifiant les contraintes du problème. Il considère le polyèdre *P* comme étant l'enveloppe convexe des points de *T*.

* **Les méthodes mathématiques**

Les méthodes mathématiques consistent à écrire les données, les contraintes et la fonction d'évaluation des critères sous forme d'équations mathématiques. Ces méthodes sont courantes et très utilisées ; elles permettent, en général, d'obtenir une écriture simple, puissante et facilement exploitable en programmation.
Par exemple, les données suivantes sont relatives à l'exemple de la figure 1.1, $i \in \{1,2,3,4,5,6,7\}$, représente l'ensemble des tâches,l'objectif étant de :

- calculer t_i (date de début d'exécution de la tâche *i*)
- minimiser la date de fin d'exécution de la dernière tâche, le $C_{\max}$,
- respecter les contraintes indiquées dans le tableau 1.1, dans lequel p_i représente la durée opératoire de la tâche *i.*

Tab.1.1. Données et contraintes relatives à l'exemple de la figure 1.1

Contraintes de données	Contraintes de précédence	Contraintes de ressources
$p_1 = 3$	$t_1 + p_1 \le t_3$	$(t_1 + p_1 \le t_2) OU (t_2 + p_2 \le t_1)$
$p_2 = 2$		
$p_3 = 4$	$t_1 + p_1 \le t_4$	$(t_3 + p_3 \le t_5) OU (t_5 + p_5 \le t_3)$
$p_4 = 1$		$(t_3 + p_3 \le t_7) OU (t_7 + p_7 \le t_3)$
$p_5 = 8$	$t_3 + p_3 \le t_6$	$(t_7 + p_7 \le t_5) OU (t_5 + p_5 \le t_7)$
$p_6 = 3$	$t_4 + p_4 \le t_7$	
$p_7 = 7$	$t_2 + p_2 \le t_5$	$(t_4 + p_4 \le t_6) OU (t_6 + p_6 \le t_4)$

1.2.4.1. Représentation des solutions

La solution d'un problème d'ordonnancement est représentée généralement par le digramme de Gantt. Ce diagramme constitue la plus ancienne et la plus facile des méthodes envisagées pour représenter cette solution.

Le digramme de Gantt peut avoir deux représentations, figure 1.2 :

- la représentation (a), « Produits »,
- la représentation (b), « Machines ».

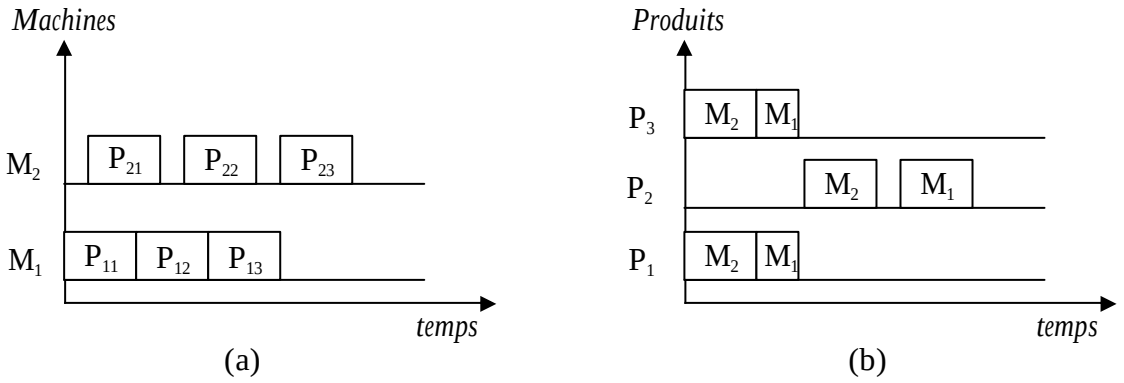


Fig.1. 2. Digrammes de Gantt

1.3. Particularités de l'ordonnancement dans un atelier de production agroalimentaire

1.3.1 Introduction

Les industries agroalimentaires sont très dépendantes de l'évolution des habitudes de consommation dans le monde entier et des différentes réglementations internationales. Les entreprises agroalimentaires doivent elles-mêmes évoluer en terme d'emploi : effectifs, structure de qualification, compétences, etc.

L'évolution de ce type d'entreprise dépend, essentiellement, des facteurs suivants :

- un facteur externe qui est directement lié au contexte sectoriel et qui dépend de l'évolution qualitative et quantitative du marché, des circuits de distribution et des innovations technologiques,
- un facteur interne qui dépend de la stratégie ou de la politique de l'entreprise qui identifie les objectifs souhaités,
- un deuxième facteur interne qui concerne essentiellement les choix de l'entreprise en matière d'organisation de travail et de coordination entre les postes fonctionnels et opérationnels.

Ce dernier facteur a été souvent jugé très important et même décisif pour la majorité des entreprises agroalimentaires [Gar03]. En effet, certaines entreprises ont fait un effort d'automatisation de leurs lignes de fabrication dans le but de mieux organiser et maîtriser les flux de produits.

1.3.2. Les produits

1.3.2.1. Les composants primaires

Un produit agroalimentaire fini P_i à fabriquer est composé par n opérations et caractérisé par :

- r_i : la date de mise en fabrication au plus tôt,
- d_i : la date de fin de fabrication au plus tard,
- g_i : la gamme définissant l'ensemble des opérations à réaliser.

Ce produit est également défini par une nomenclature qui met en évidence les composants primaires nécessaires pour l'obtenir. Ces composants sont souvent représentés par des produits agricoles et des matières crues qui sont périssables et qui ont des durées de vie limitées.

1.3.2.2. Les produits semi-finis

A cause des risques de contamination, les produits semi-finis manipulés dans les industries agroalimentaires ont généralement des durées de vie très courtes. Ces produits doivent être transférés sur les postes de travail dans délais très courts. En effet, la synchronisation entre les différents postes de travail de la ligne de fabrication est nécessaire pour éviter la péremption en ligne.

1.3.2.3. Les produits finis

Contrairement à d'autres produits industriels, les produits agroalimentaires ont une durée de vie (*Dv*) bien déterminée à compter de leurs dates de fin de fabrication (*Dff*). A chaque produit fini, est alors associée une date limite de consommation (*Dlc*). Il est à noter que la vente de ces produits passe souvent par des surfaces de distribution qui imposent certaines conditions aux manufacturiers, le produit étant considéré invendable à partir d'un certain délai de sa date limite de consommation. Ce délai est appelé délai de retour (*Dr*). Le responsable de la production doit alors reprendre son produit.

Exemple (figure 1.3.) :

Considérons un exemple de produit fini ayant une durée de 21 jours. Si son délai de retour est fixé à 5 jours, ce produit doit être repris par le manufacturier 5 jours avant sa date limite de sa consommation [Gar03].

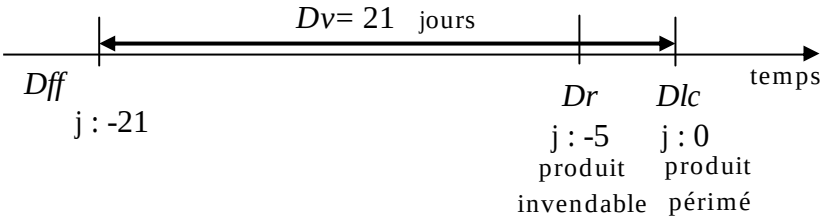


Fig.1.3. Cycle de vie d'un produit agroalimentaire

1.3.3. Spécificités d'un problème d'ordonnancement en industries agroalimentaires

Dans le paragraphe précédent, les particularités des produits manipulés en industries agroalimentaires sont citées. En effet, il faut tenir compte de ces particularités lors de la résolution du problème d'ordonnancement. Contrairement aux autres industries, la succession des opérations dans un même poste de travail dans un atelier de production agroalimentaire est caractérisée, généralement, par des opérations de nettoyage de durées variables qui interrompent le cycle de fonctionnement de la production et peuvent donc provoquer des retards.

Ces opérations, engendrant des temps improductifs assez importants, nécessitent l'optimisation des changements de recettes et des gammes de produits.
D'autres opérations de nettoyage de durées d'exécution plus importantes sont d'autre part imposées par le service qualité pour éviter la contamination des ressources et assurer ainsi une qualité satisfaisante. Dans ce type d'industrie, les produits semi-finis résultants des opérations élémentaires du processus de fabrication peuvent être modifiés et ne sont pas forcément ceux qui ont été planifiés, de ce fait, l'évolution dans le temps d'une opération en cours de réalisation n'est pas toujours déterministe (les caractéristiques des composants : acidité, concentration, etc. ou encore les conditions de production : température, taux d'humidité, etc.). Ces produits peuvent être complètement perdus si aucune action corrective n'est possible. Des décisions d'interruptions des opérations en-cours ou d'engagements d'autres opérations peuvent avoir lieu dans telles conditions. On est donc face à un problème d'ordonnancement dynamique. Un outil réactif d'aide à la décision en temps réel peut être alors la meilleure solution pour ce type de problème.

Ci-après un tableau (Tab.1.2) qui résume les principales différences entre un ordonnancement classique et un ordonnancement en industries agroalimentaires.

Tab.1.2. Comparaison entre un problème d'ordonnancement classique et celui en industries agroalimentaires

<table>
<tr><th>Critères de comparaison</th><th>Ordonnancement classique</th><th>Ordonnancement agroalimentaire</th></tr>
<tr><td>Matières premières</td><td>- non périssables
- discrétisées
- approvisionnement maîtrisé</td><td>- périssables
- non discrétisées
- approvisionnement irrégulier</td></tr>
<tr><td>Produits semi-finis</td><td>- non périssables
- discrétisés</td><td>- durée de vie courte
- non discrétisés</td></tr>
<tr><td>Produits finis</td><td>- non périssables
- prix de vente fixe et invariable</td><td>- périssables
- prix de vente variable
- coût de stockage élevé</td></tr>
<tr><td>Durées opératoires</td><td>- fixes
- temps d'attente connus</td><td>- variables
- temps d'attente entre opérations, variable</td></tr>
<tr><td rowspan="2">Critères</td><td></td><td>- minimisation des produits périmés
- minimisation du discount de distribution</td></tr>
<tr><td colspan="2">- minimisation du makespan
- minimisation du retard
- respect des dates d'échéances clients
- équilibre des charges de ressources
- minimisation des changements d'outils</td></tr>
<tr><td>Contraintes</td><td colspan="2">- contraintes de disponibilité des matières premières
- contraintes de disponibilité du personnel
- contraintes de gamme (de précédence)
- contraintes disjonctives et cumulatives</td></tr>
<tr><td>Demande</td><td>demande prévisible</td><td>demande variable</td></tr>
</table>

1.4. Les problèmes d'ordonnancement à une machine

1.4.1. Introduction

Les problèmes classiques à une machine, en se limitant à une approche monocritère, peuvent être décomposés en deux catégories principales : les problèmes de type « min max » où la fonction objectif consiste à minimiser la plus grande valeur du critère considéré et les problèmes de type « min sum » où la fonction objectif consiste à minimiser la somme des critères. Plusieurs travaux de recherche ont été consacrés à ce type de problème [Alb93], [Bea92], [Wig94], [Bru98], [Cha98a], [Cha98b], [Now94], [Pan92], [Cha96], etc.

Dans cette partie, un état de l'art sur les problèmes monocritères à une machine, où la fonction objectif consiste à minimiser une fonction de coût, est présenté. Ces problèmes sont de type « min-max » puisque l'objectif est de minimiser la plus grande valeur du coût. Ils sont dits aussi problèmes de type disjonctif ou de capacité unitaire, c'est-à-dire que la capacité de la ressource est 1, elle ne peut exécuter qu'une seule opération à la fois.

1.4.2. Présentation du problème à une machine

Dans un problème à une machine, le but est d'ordonnancer un ensemble E de n opérations sur une machine unique. Chaque opération O_i, occupant la ressource pendant un temps donné, est caractérisée par :

- sa date de début au plus tôt r_i,
- sa date de début effective t_i,
- sa durée opératoire p_i,
- sa date de fin effective C_i,
- sa date de fin au plus tard d_i.

Quant une opération O_i est soumise à une date au plus tôt r_i et une date de fin au plus tard d_i, l'intervalle $[r_i, d_i]$ représente la fenêtre de réalisation de cette

opération, où elle doit impérativement être exécutée, l'objectif étant de minimiser une fonction objectif dont les plus courantes sont : la date de fin d'exécution de l'ordonnancement $C_{\max}$ (le makespan), le plus grand retard algébrique $L_{\max}$. Ils existent des relations qui lient deux opérations : les contraintes de précédence. Ces contraintes sont de deux types : le premier type indique que, pour qu'un ordonnancement soit réalisable, il faut qu'une opération O_i s'exécute avant l'opération O_j. Le deuxième ajoute, en plus de cette condition, qu'entre le début de l'opération O_i et le début de l'opération O_j, un intervalle de temps doit être respecté.

Deux types de problèmes sont distingués : les problèmes *préemptifs* et les problèmes *non-préemptifs*. Dans le premier cas, une opération peut être interrompue pendant son exécution tandis que pour le second un tel procédé est interdit [Mau04].

1.4.3. Les problèmes à une machine non préemptifs

Suivant la taille et la complexité du problème, les problèmes à une machine non préemptifs sont de deux types : les problèmes polynomiaux et ceux NP-difficiles.

1.4.3.1. Les problèmes polynomiaux

* *Exemples*
  - Problème : $1 \mid r_i \mid C_{\max}$

C'est un problème à une machine où, chaque opération $O_i \in E$ de durée opératoire p_i, ne peut être exécutée avant sa date de début au plus tôt r_i ; l'objectif étant de minimiser la durée totale de l'ordonnancement : le makespan. Ce problème peut être résolu par l'algorithme ERT (Earliest Release Time), qui donne, en général, une solution optimale [Mau04]. Cet algorithme consiste à ordonnancer les opérations en les triant par dates de début au plus tôt croissantes.

- Problème : $1|.|L_{\max}$

L'objectif de ce problème est de minimiser le plus grand retard algébrique $L_{\max}$, c'est à dire de minimiser le plus grand retard associé à une opération $O_i \in E$. Si toutes les opérations sont terminées à temps, il s'agit de maximiser la plus petite avance. L'algorithme qui peut résoudre ce problème est l'algorithme EDD (Earliest Due Date) qui consiste à placer par ordre croissant les opérations des dates de fin souhaitées.

1.4.3.2. Les problèmes NP-difficiles

∗ *Exemple*

- Problème : $1|r_i|\max(C_i + q_i)$

La complexité de ce problème a été démontrée qu'elle est NP-difficile au sens fort [Len77] [Gar79]. Ce problème a fait l'objet de plusieurs travaux de recherche dans les années soixante-dix et les années quatre-vingt [Bra73] [Mah75] [Pot80] [Car82], etc.

Bien que ce type de problème ait une complexité assez forte, une méthode de Séparation et d'Evaluation Progressive (PSE), proposée par Carlier [Car82], a été efficace pour le résoudre. La méthode possède la particularité de décrire un ordonnancement complet à chaque nœud et repose sur le principe que l'algorithme de Schrage [Sch71] peut toujours fournir la solution optimale du problème à une machine, si les données de ce problème ont été modifiées pour certaines opérations (voir section 2.3.4).

1.4.4. Les problèmes à une machine préemptifs

14.4.1. Les problèmes polynomiaux

* *Exemple*

- Problème : $1|prmt,\ r_i|\max(C_i+q_i)$

Pour ce type de problème, chaque opération $O_i \in E$ de durée opératoire p_i ne peut être exécutée avant sa date de début au plus tôt r_i et est effectivement achevée après une durée de latence q_i. De plus la péremption est autorisée.

Ce problème est équivalent au $1|prmt,\ r_i|L_{\max}$. Il peut être résolu, en un temps polynomial et en donnant une solution optimale, par l'algorithme de JPS (Jackson's Preemptive Schedule) [Jac55]. Cet algorithme consiste à placer en priorité l'opération disponible de plus grand q_i. Il peut être programmé en un temps *O (n log n)* [Car82], où *n* représente le nombre d'opérations à ordonnancer.

14.4.2. Les problèmes NP-difficiles

* *Exemple*

- Problème : $1|prmt, prec,\ r_i|\max(C_i+q_i)$

C'est le même problème que le précédent avec, dans ce cas, des contraintes de précédence généralisées ajoutées. De ce fait, ce problème est classé NP-difficile au sens fort [Bal95] pour lequel il n'existe, jusqu'à présent, à notre connaissance, aucun algorithme pour le résoudre.

1.4.5. Les problèmes à une machine en temps réel

Les premiers problèmes importants traités en temps réel ont concerné le pilotage de systèmes complexes (avion, centrale nucléaire,…). Ils sont concernés également l'ordonnancement de radars [Orm98], ou de processeurs en informatique [Chr95]. Pour ces problèmes, il est nécessaire de fournir rapidement une solution sans pouvoir attendre les informations sur les demandes à long terme.

L'ordonnancement en temps réel permet aussi de fournir un délai de fabrication précis au client, à l'instant où il passe sa commande [Fia98]. Aujourd'hui, les applications de l'ordonnancement en temps réel se développent parce qu'elles permettent de réagir dynamiquement aux aléas imprévisibles qui peuvent intervenir à chaque instant.

Les applications en temps réel impliquent de développer des techniques rapides, et de les appliquer, de préférence pour tous les cas de problèmes rencontrés. La rapidité souhaitée d'obtention des résultats dépend du problème traité : pour une entreprise, il peut s'agir de quelques minutes, pour un radar, de quelques millisecondes [Dur02]. Le terme temps réel signifiant que l'application considérée est capable de fournir une solution à un problème avant que d'autres évènements ne viennent modifier l'état du système. La rapidité d'un algorithme en temps réel est fonction de sa « complexité », c'est-à-dire l'évaluation du nombre d'opérations élémentaires effectuées par l'algorithme dans le cas extrême. Elle est donnée en fonction de la taille des problèmes traités [Gau87]. Pour garantir que le temps d'exécution d'un algorithme ne dégénère pas dans des cas d'application en temps réel, il est souhaitable que sa complexité soit polynomiale.
Au contraire, les problèmes NP-difficiles au sens fort forment une classe pour laquelle on ne connaît pas d'algorithme polynomial pour les résoudre de manière optimale ; les seules méthodes connues consistent à explorer explicitement ou implicitement toutes les solutions, ce qui n'est possible que pour des exemples de taille très limitée. Ainsi, pour les applications réelles et lorsque la recherche de la solution optimale devient «combinatoire », des heuristiques qui fournissent, au mieux, une solution proche de la solution optimale, sont utilisées.

1.5. Problèmes d'optimisation et méthodes de résolution

1.5.1. Introduction

Les problèmes d'ordonnancement sont, en général, des problèmes d'optimisation puisque leur objectif est de minimiser (ou maximiser) une fonction objectif tout en respectant certains critères.

1.5.2. Problèmes d'optimisation

Résoudre un problème d'optimisation consiste à trouver la (ou les) meilleure(s) solution(s), vérifiant un ensemble de contraintes et d'objectifs définis par l'utilisateur [Bar03]. Pour déterminer si une solution est meilleure qu'une autre, il est nécessaire que le problème introduise un critère de comparaison.

1.5.2.1. Formulations

∗ ***La fonction objectif***

C'est la fonction de coût, notée f qu'il s'agit d'optimiser.

∗ ***Les variables de décision***

Elles sont regroupées dans un vecteur x. C'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction f [Col02].

∗ ***Types de minima***

Il existe deux types de minima : les minima locaux et les minima globaux, figure 1.4.

- **Minimum global**

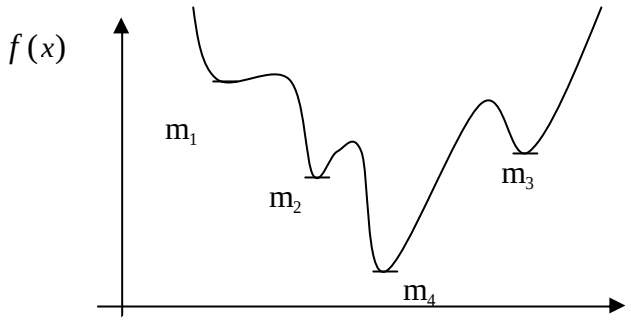
Un point x^* est un minimum global de la fonction f, si on a :

$f(x^*) < f(x) \quad \forall x$ tel que $x \neq x^*$

- **Minimum local**

Un point x^* est un minimum local de la fonction f si on a :

$f(x^*) < f(x) \quad \forall x \in V(x^*)$ et $x \neq x^*$ où $V(x^*)$ définit un voisinage de x^*



m_1, m_2, m_3 sont des minima locaux

m_4 est un minimum global

Fig.1.4. Les deux types de minima

1.5.2.2. Problèmes mono-objectifs.

D'un point de vue mathématique, un problème d'optimisation mono-objectif est formulé de la façon suivante :

$$\begin{cases} \text{minimiser } f(\vec{x}) \text{ (fonction à optimiser)} & \vec{x} \in \mathbb{R}^n,\ f(\vec{x}) \in \mathbb{R} \\ \text{avec } g(\vec{x}) \le 0 \quad \text{(m contraintes d'inégalités)} & \vec{g}(\vec{x}) \in \mathbb{R}^m \\ \text{et } h(\vec{x}) = 0 \quad \text{(n contraintes d'égalités)} & \vec{h}(\vec{x}) \in \mathbb{R}^n \end{cases}$$

1.5.2.3. Problèmes multi-objectifs

Lorsqu'on modélise un problème, il est souvent nécessaire de satisfaire plusieurs objectifs [Bou06b]. Dans ce cas, on parle d'optimisation multi-objectifs.

D'un point de vue mathématique, un problème d'optimisation multi-objectifs, se présente, dans le cas où le vecteur $\vec{f}$ regroupe *k* fonctions objectif, de la façon suivante [Col02] :

$$\begin{cases} \text{minimiser } \vec{f}(\vec{x}) & \vec{x} \in \Re^m,\ \vec{f}(\vec{x}) \in \Re^k \\ \text{avec } \vec{g}(\vec{x}) \le 0 & \vec{g}(\vec{x}) \in \Re^m \\ \text{et } \vec{h}(\vec{x}) = 0 & \vec{h}(\vec{x}) \in \Re^p \end{cases}$$

1.5.2.4. Notion de dominance

En ordonnancement, le concept de dominance d'un sous-ensemble de solutions est important afin de limiter la complexité algorithmique liée à la recherche d'une solution optimale au sein d'un grand ensemble de solutions.

Pour trouver une solution optimale aux problèmes d'optimisation multi-objectifs, constituant un ensemble de points, il s'avère nécessaire de définir une relation d'ordre entre ces éléments, dite relation de dominance, pour identifier les meilleurs compromis. La règle de dominance est une contrainte qui peut être ajoutée au problème initial sans changer la valeur de l'optimum [Jou02b]. La plus utilisée est celle définie au « sens de Pareto » [Bar03].

Ainsi, la résolution d'un problème d'optimisation multi-objectifs conduit généralement à une multitude de solutions. Seul un nombre restreint de ces solutions est intéressant. Une solution est considérée intéressante s'il existe une relation de dominance entre cette solution et les autres solutions [Col02].

On dit que le vecteur $\vec{x}_1$ domine le vecteur $\vec{x}_2$ si :

- $\vec{x}_1$ est au moins aussi bon que $\vec{x}_2$ dans tous les objectifs,
- $\vec{x}_1$ est strictement meilleur que $\vec{x}_2$ dans au moins un objectif.

Les solutions qui dominent les autres mais ne se dominent pas entre elles, sont appelées solutions optimales au sens de Pareto.

1.5.3. Méthodes de résolution

Les problèmes d'optimisation sont, en général, difficiles à résoudre. Plusieurs méthodes sont utilisées afin de trouver une réponse satisfaisante à ces problèmes. On distingue les méthodes exactes et les méthodes approchées [Bar03].

1.5.3.1. Méthodes exactes

Les problèmes d'ordonnancement sont traités par des méthodes d'optimisation en considérant les données du problème comme des contraintes à satisfaire et en proposant une solution optimale et admissible. L'optimalité des données est mesurée par rapport aux critères et aux objectifs établis par le niveau hiérarchique supérieur de décision [Gar03].

On distingue :

- la méthode de Séparation et d'Evaluation Progressive ou « branch & bound »,
- la programmation linéaire,
- la programmation dynamique.

Ces méthodes exactes, examinent, d'une manière implicite, la totalité de l'espace de recherche et produisent, en principe, une solution optimale. Lorsque le temps de calcul nécessaire pour atteindre cette solution est excessif, les méthodes approchées peuvent fournir une solution quasi-optimale au bout d'un temps de calcul raisonnable.

1.5.3.2. Méthodes approchées

Ces méthodes sont considérées pour les problèmes d'ordonnancement dans lesquels on ne trouve pas de solution optimale en un temps raisonnable [Lio98].

On distingue :

- les méthodes basées sur des heuristiques, ensembles des connaissances fruits de l'expérience, se présentant comme un ensemble de règles simples EDD (Earliest Due Date), FIRO (First In Random Out), SPT (Shorter Processing Time), etc.

- les méthodes basées sur les métaheuristiques, telles que les méthodes de recherche par voisinage, la recherche tabou [Glo97], les algorithmes génétiques [Hol75], les algorithmes de colonies de fourmis [Col91], [Col94], le recuit simulé[Kir83],etc.

* **Les heuristiques**

Les heuristiques sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables. Elles se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de cette catégorie de méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimal tout en essayant d'avoir un temps de calcul raisonnable [Bel01], [Bou91], [Kac03].

Exemples d'heuristiques :

– EDF (Earliest Deadline First) ou EDD : cet algorithme choisit parmi les tâches exécutables celle dont le délai est échu le plus tôt. Si aucune tâche n'est disponible, alors un temps libre est généré [Wan99] [Mok83].

– LLF ou LL (Last Laxity First) : cet algorithme choisit parmi les tâches exécutables celle dont la distance entre la fin de son exécution et son délai est le plus court. Si aucune tâche n'est disponible, alors un temps libre est généré [Hak99].

– LRF (Last Release Time First) : cet algorithme choisit parmi les tâches exécutables celle dont le temps écoulé depuis son apparition est le plus grand. Si aucune tâche n'est disponible, alors un temps libre est généré [Sch96].

– HPF (High Priority First) : cet algorithme choisit parmi les tâches exécutables celle dont la priorité est la plus grande. Si aucune tâche n'est disponible, alors un temps libre est généré.

Il a été démontré que l'algorithme EDF[Wan99] [Mok83] qui ordonnance les tâches dans l'ordre croissant de leur délai d'exécution, et l'algorithme LL (Last Laxity) [Cha00], sont optimaux dans le cas de tâches indépendantes et préemptives exécutées sur une seule ressource.

* **Les métaheuristiques**

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

- ***Les méthodes par voisinage***

Les méthodes par voisinage suivent la démarche suivante [Car88] :

- construire une solution initiale x,
- déterminer parmi les solutions voisines y de x celle de $f(y)$ minimale,
- si $f(y) \prec f(x)$ alors remplacer x par y et aller à la deuxième étape,
  sinon retenir x comme meilleure solution.

Ces méthodes généralement plus rapides que les méthodes exactes, donnent de bons résultats et ont un coût assez réduit. Un autre avantage réside dans la possibilité de contrôler le temps de calcul [Hao99]. En effet, la qualité de la solution trouvée tend à s'améliorer progressivement au cours du temps et l'utilisateur est libre d'arrêter l'exécution au moment qu'il aura choisi.

- ***La recherche tabou***

Cette méthode a pour but d'améliorer à chaque étape, la valeur de la fonction objectif. Elle utilise une mémoire afin de conserver les informations sur les solutions déjà visitées. Cette mémoire constitue la liste tabou qui va servir à interdire l'accès aux dernières solutions visitées. Lorsqu'un optimum local est atteint, il y'a interdiction de revenir sur le même chemin [Glo89].

- ***Le recuit simulé***

Inspiré du recuit physique, ce processus est utilisé en métallurgie pour améliorer la qualité d'un solide et cherche un état d'énergie minimale qui correspond à une structure stable du solide. Ainsi, pour qu'un métal retrouve une structure proche du cristal parfait, on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement [Kir83]. En optimisation [Col02], [Hao99], le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. Ainsi, si une amélioration du critère est constatée, le nouvel état est retenu, sinon une diminution ΔE du critère est calculée et le nouvel état est retenu avec une probabilité $p = \exp(-\Delta E / T)$, figure 1.5.

Algorithme [Mes99]

T : paramètre qui diminue lorsque le nombre d'itérations augmente (T : Température)

E : énergie du système

ΔE : variation de l'énergie

e : température seuil à fixer par l'utilisateur

- Initialisation des paramètres de l'algorithme(T, e),
- modification élémentaire (variation d'énergieΔE),
- application de la règle d'acceptation de Metropolis [Car88]

  si $\Delta E < 0$ alors la modification est acceptée,

  si $\Delta E > 0$ alors la modification est acceptée avec la probabilité $p = \exp(-\Delta E / T)$,

  si $T > e$ alors diminuer lentement T et retourner à la deuxième étape,

  si $T \leq e$ ou (amélioration non observée) alors l'état actuel est la solution recherchée,

- fin.

Fig.1.5. Algorithme général du recuit simulé

- ***Les algorithmes génétiques***

Les Algorithmes Génétiques [Hol75] sont des algorithmes itératifs dont le but est d'optimiser une fonction prédéfinie, appelée « fitness ». Pour réaliser cet objectif, l'algorithme, figure 1.6., travaille sur un ensemble de points, appelés population d'individus. Chaque individu ou chromosome (chaîne binaire de longueur finie) représente une solution possible du problème donné. Il est constitué d'éléments appelés gènes, pouvant prendre plusieurs valeurs, appelées allèles [Mes99].

Le principe des algorithmes génétiques repose sur une analogie entre un individu dans une population et la solution d'un problème parmi un ensemble de solutions potentielles : un individu (une solution) est caractérisé par une structure génétique (codage des solutions du problème). Selon les lois de survie de Darwin, seuls les individus les plus forts (les meilleures solutions) survivront et pourront donner une descendance. Les opérateurs de croisement et de mutation (recombinaison et mutation des codages des solutions) permettent de se déplacer dans l'espace des solutions du problème. A partir d'une population initiale et après un certain nombre de générations, on obtient une population d'individus forts, c'est à dire de bonnes solutions du problème considéré.

Les opérateurs utilisés par les algorithmes génétiques sont les opérateurs de sélection, les opérateurs de croisement et les opérateurs de mutation :

- l'opérateur de sélection a pour objectif de choisir les individus qui vont pouvoir survivre et / ou se reproduire pour transmettre leurs caractéristiques à la génération suivante. La sélection se base sur le principe de conservation des individus les mieux adaptés et d'élimination des moins adaptés [Hao99].
- l'opérateur de croisement, assurant le brassage et la recombinaison des gènes parentaux, permet de former des descendants aux potentialités nouvelles.
  Étant aléatoire, cet opérateur agit selon une probabilité *Pc* fixée par l'utilisateur en fonction du problème à optimiser [Mes99].

- l'opérateur de mutation consiste à changer, aléatoirement, la valeur de certains allèles dans un individu. Sans elle, on aura une population uniforme, incapable d'évoluer. La mutation classique consiste à changer un 0 par un 1 et inversement.

Les travaux de recherche sur les algorithmes génétiques a pour souci principal l'amélioration de la robustesse, l'équilibre entre la performance et le coût nécessaire à la survie dans des environnements nombreux et différents [Gol94].

1. Calcul de la population initiale
   Construire une population de solutions réalisables
2. Calcul des nouveaux individus
   Accroître la population en ajoutant des individus issus de mutations ou de croisements d'individus de la génération précédente
3. Sélection des individus
   Sélectionner les meilleurs individus pour revenir à la taille de la population initiale
4. Condition d'arrêt
   Recommencer les croisements et les mutations jusqu'à vérifier une condition d'arrêt

Fig. 1.6. Algorithme génétique simple

- ***Les algorithmes de colonies de fourmis***

Les algorithmes d'Optimisation par Colonies de Fourmis (OCF), figure 1.7., sont proposés pour résoudre des problèmes NP-difficiles. Ces algorithmes sont nés à la suite de constatations faites sur le comportement des fourmis qui sont capables de trouver le chemin le plus court du nid à une source de nourriture et de s'adapter aux changements de l'environnement et ceci grâce à la phéromone (substance leur permettant de laisser une trace sur leur chemin). Ces algorithmes ont donc pour but de reproduire le comportement naturel des fourmis pour trouver la meilleure solution possible à plusieurs types de problèmes [Gag01].

Algorithme général

Tant que la condition d'arrêt n'est pas atteinte faire

{Initialisation}

1. Générer et activer la colonie de fourmis
2. Initialiser la phéromone

{Début de l'optimisation}

Chercher le chemin de chaque fourmi

Mettre à jour la phéromone

{Fin de l'optimisation}

3. Retenir la meilleure solution

Fin Tant que

Fig.1.7. Algorithme de l'OCF

1.5.3.3. Méthodes de relaxation des contraintes

Les méthodes de relaxation ont un double rôle dans la résolution de problèmes d'optimisation combinatoire [Geo74]. Elles donnent de bonnes bornes inférieures pour augmenter l'efficacité de méthodes « branch and bound » d'une part, et sont à la base de méthodes de résolutions comme la relaxation lagrangienne [Fis76] [Fis81], d'autre part. Le principe de ces méthodes et de relaxer un certain nombre de contraintes de façon à rendre plus facile la résolution du problème. Dans le cas du job-shop, une relaxation des contraintes de ressources et ensuite une relaxation des contraintes de gammes opératoires, peut être envisageable [Fis83]. Cette méthode donne de bonnes solutions pour des problèmes de petite taille mais n'a pas été testée pour des tailles plus importantes.

1.5.3.4. Méthodes de décomposition

Les méthodes de décomposition ont pour principe de diminuer la complexité du problème en décomposant le problème initial en plusieurs sous problèmes plus faciles à résoudre. L'objectif d'une telle approche est de regrouper les jobs utilisant le même ensemble de machines. Le but est alors de rendre les plus indépendants possibles les groupes de jobs [Pen94]. Pour cela, il faut qu'un job n'utilise que des machines liées à son groupe, et qu'une machine ne traite que des jobs d'un même groupe. Pour décomposer un problème de ce type, des méthodes comme la technologie de groupe peuvent être utilisées [Dri87] [Por88]. Quand le problème est décomposable, deux cas se présentent : la décomposition est dite parfaite lorsque les sous problèmes sont complètement indépendants, la résolution de chacun des sous-problèmes séparément donnant une solution globale de l'ordonnancement. Si la décomposition n'est pas parfaite lorsque certains jobs passent sur des machines liées à des groupes différents, la résolution de chaque sous problème ne donnant pas une solution réalisable du problème global [Leo91].

1.6. Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les problèmes d'ordonnancement d'ateliers et les approches utilisées pour résoudre ce type de problème, combinatoire. Ces approches, qu'elles soient exactes ou approchées, restent incapables de résoudre tous les problèmes réels d'ordonnancement. De ce fait, les méthodes hybrides donnent, en général, de meilleurs résultats.

Les particularités d'un problème ordonnancement dans un atelier de production agroalimentaire, sont présentées. Pour réussir à trouver la meilleure solution à ce type de problème, il faut d'abord, définir les critères et les contraintes spécifiques à ce type d'industrie, à savoir : la périssabilité des produits, le discount de distribution et le coût de stock. Pour remédier à ces problèmes, un ordonnancement temps réel et donc, dynamique, reste le mieux adapté.

Dans le chapitre suivant, la méthode «branch & bound », qui est une méthode exacte, est développée et appliquée pour optimiser les critères retenus.

Chapitre 2

Optimisation de la fonction de coût par la méthode « branch & bound »

2.1. Introduction

La méthode « branch & bound » ou Procédure de Séparation et d'Évaluation progressive (PSE), est une méthode exacte d'optimisation basée sur une énumération contrôlée qui fournit, en général, une solution optimale. Elle consiste à construire une arborescence dont la racine correspond à l'espace des solutions du problème initial. Dans le cas d'un problème de minimisation, un majorant ou une borne supérieure d'une solution réalisable de la valeur de la fonction objectif, est avant tout calculé en utilisant une méthode approchée ou une heuristique. Elle constitue une des méthodes pour résoudre le problème de satisfaction de contraintes.

2.2. Problème de satisfaction de contraintes

2.2.1. Introduction

Le problème de satisfaction de contraintes (Contraints Satisfaction Problem, CSP) [Dec92] [Bap98] a été résolu par plusieurs méthodes. Sa résolution, s'effectue, en général, par l'intermédiaire d'un arbre de recherche, en plusieurs étapes : d'abord l'étiquetage, ensuite le retour arrière (backtracking), et enfin la propagation.

Cette approche a fait ses preuves dans la résolution des problèmes d'ordonnancement. Plusieurs applications industrielles ont confirmé l'efficacité de cet outil de résolution.

2.2.2. Définitions

- *Définition 2.1*

Un problème de satisfaction de contraintes (CSP) est défini par un ensemble de variables et un ensemble de contraintes. Chaque variable peut prendre une valeur choisie dans le domaine qui lui est associé. Les contraintes, quant à elles, décrivent les combinaisons autorisées de valeurs pour les variables. L'objectif est d'attribuer une valeur à chaque variable de sorte que toutes les contraintes soient satisfaites.

D'une manière plus explicite [Mon74] :

Une instance CSP, notée *P*, est un quadruplet (*X*, *D*, *C*, *R*) avec :

1. *X* : un ensemble $\{x_1, x_2, ..., x_n\}$ de variables ;
2. *D* : un ensemble de domaines finis $\{d_{x1}, d_{x2}, ..., d_{xn}\}$, d_{xi} étant associé à la variable x_i de *X* et contenant les valeurs que peut prendre la variable x_i ;
3. *C* : un ensemble $\{c_1, c_2, ..., c_m\}$ de contraintes, chaque contrainte c_i de *C* est définie par l'ensemble de variables sur lesquelles elle porte ;
4. *R* : un ensemble $\{r_{c1}, r_{c2}, ..., r_{cm}\}$ de relations, r_{ci} étant relative à la contrainte c_i et définie sur $\prod_{x \in c_i} d_x$. Cette relation représente les affectations compatibles entre les variables contraintes par c_i.

- *Définition 2.2*

Étant donné $Y \subseteq X$, une *instanciation* des variables de *Y*, Y = $\{y_1,..., y_k\}$, est un k-uplet $(v_1,..., v_k)$ de $(d_{y1} \times \cdots \times d_{yk})$. Une instanciation est dite complète si elle porte sur toutes les variables de *X*, partielle sinon. Le terme *affectation* est employé aussi au lieu *d'instanciation.*

- *Définition 2.3*

Une affectation A satisfait une contrainte c de C si $A[c] \in r_c$. Sinon, A viole c. Une instanciation A est qualifiée de *consistante* si $\forall c \in C, c \subseteq X_A, A[c] \in r_c$; elle est dite *inconsistante* sinon. Autrement dit, une instanciation est consistante si elle ne viole aucune contrainte. Dans la littérature, le terme *cohérent* est souvent employé (resp. *incohérent*).

- *Définition 2.4*

Une *solution* est une instanciation complète *consistante*. Autrement dit, une solution est une affectation de toutes les variables de X avec des valeurs prises dans leurs domaines respectifs, qui satisfait toutes les contraintes. Une instance P est dite *consistante* si P possède au moins une solution. Elle est qualifiée *d'inconsistante* sinon.

Les algorithmes d'établissement de la consistance sont issus des techniques de graphes. Un CSP est représenté par un graphe de contraintes où les nœuds correspondent aux variables et les arcs sont étiquetés par les contraintes. Ceci implique que le CSP doit être ramené à un CSP binaire, c'est-à-dire à un CSP n'ayant que des contraintes binaires.

2.2.3. Principe de résolution d'un CSP

Le principe de résolution d'un problème de satisfaction de contraintes est constitué, généralement, par un étiquetage, un « backtracking » et un arbre de recherche.

2.2.3.1. L'étiquetage

Lors de la recherche d'une solution, il faut étiqueter une ou plusieurs variables. C'est l'étape d'étiquetage ou « labelling » qui consiste à attribuer aux variables des valeurs temporaires, qui doivent être prises dans leurs domaines respectifs.

2.2.3.2. Le retour arrière « backtracking »

Le « backtracking » consiste à construire progressivement une solution en affectant les variables une par une et en revenant en arrière à chaque échec rencontré. Lorsque toutes les variables concernant une contrainte sont instanciées, la validité de cette contrainte est testée. Si la solution partielle viole une des contraintes du problème, un retour arrière est effectué jusqu'à la dernière instanciation partielle qui ne viole pas les contraintes et qui représente une solution alternative à celle qui vient d'échouer.

Plus explicitement, l'algorithme de backtracking, noté BT, élimine les sous-espaces des domaines de variables qui pourraient être engendrés à partir de la solution partielle qui viole une ou plusieurs contraintes.
Ainsi, le backtracking ne parcourt pas toutes les solutions et explore donc moins de solutions que l'algorithme général, puisqu'il élimine les branches qui donnent des mauvaises solutions vis-à-vis du critère à optimiser. Cependant sa complexité, dans le cas extrême reste exponentielle.

Plus formellement, étant donnée une affectation consistante A, l'algorithme BT tente d'étendre de façon consistante A. Dans ce but, il choisit une variable x parmi les variables non instanciées et lui attribue une valeur v issue de d_x. L'ordre d'instanciation des variables peut alors être soit prédéfini, soit calculé grâce à une heuristique. Il en est de même pour l'ordre d'instanciation des valeurs. Ainsi, BT étend l'affectation A en $A \cup \{x \leftarrow v\}$. Il évalue alors la consistance de l'affectation en vérifiant que $A \cup \{x \leftarrow v\}$ ne viole aucune contrainte liant x et une variable affectée dans A. Si $A \cup \{x \leftarrow v\}$ est inconsistante, BT essaie d'étendre A avec une nouvelle valeur pour x. S'il n'existe plus de valeur possible dans le domaine de x, alors il faut revenir en arrière sur la variable instanciée juste avant x. Si l'affectation $A \cup \{x \leftarrow v\}$ est consistante, BT cherche à l'étendre comme précédemment.

La recherche se poursuit soit jusqu'à ce que toutes les variables aient été instanciées de façon consistante (c'est-à-dire jusqu'à l'obtention d'une solution), soit jusqu'à ce que toutes les possibilités aient été envisagées. L'algorithme BT est décrit à la figure 2.1. *A* représente l'affectation courante et *V* l'ensemble des variables non instanciées.

Initialement $A = \emptyset$; et $V = X$. La complexité en temps de cet algorithme est exponentielle.

BT (A, V)

1. $V \neq \emptyset$
2. Choisir $x \in V$
3. $d \leftarrow d_x$
4. *Consistance* ←Faux
5. Tant que $d \neq \emptyset$; et ¬*Consistance* Faire
6. Choisir v dans d
7. $d \leftarrow d \setminus \{v\}$
8. Si $\nexists$c ∈ C telle que c viole $A \cup \{x \leftarrow v\}$.
9. Alors *Consistance* ← BT ($A \cup \{x \leftarrow v\}$, $V \setminus \{x\}$)
10. Fin Si
11. Fin Tant que
12. Return *Consistance*
13. Fin Si

Fig.2.1. Algorithme « backtracking » [Ter02]

2.2.3.3. Arbre de recherche

La recherche d'une solution passe par le parcours d'un arbre de recherche. Soit un exemple simple avec deux variables *x* et *y* (sans tenir compte des contraintes qui les relient). Au premier point de choix, on peut choisir entre les valeurs de *x*. Si on choisit une valeur de *x*, on doit effectuer un autre choix pour *y*. Si une certaine valeur de y échoue lors de la vérification des contraintes, on effectue un retour arrière jusqu'au plus proche point de choix possédant une branche non encore explorée, et ainsi de suite jusqu'à ne plus avoir de point de choix disponible ou jusqu'à ce qu'une solution est rencontrée.

* **Exemple**

Etant donné un problème d'ordonnancement de type job-shop formé de trois machines M_1, M_2, M_3 et deux jobs J_1 et J_2. Le but est de déterminer l'ordonnancement des jobs sur ces trois machines tout en respectant les contraintes de problème dont l'objectif est d'optimiser la date de fin de l'ordonnancement le C_{max}. L'arbre de recherche binaire, figure 2.4, dans lequel une disjonction est fixée dans chacun des nœuds, peut être employé pour résoudre ce problème. L'optimum est 20 (la feuille cerclée dans la figure 2.2). Supposons qu'une solution heuristique a été trouvée (avec une valeur de 30, cette valeur représente une borne supérieure BS du problème considéré) mais aucune borne inférieure n'a été calculée. La recherche commencera à partir du noeud marqué 120 dans l'arbre. Ce noeud dépasse la borne supérieure, il n'est pas donc une solution acceptable. Depuis la borne supérieure a été respectée dans son nœud père, on peut conclure que le choix de $J_1 \longrightarrow J_2$ n'est pas satisfaisant pour la machine 3.

Le prochain noeud exploré est ainsi celui dont la valeur est 80. La borne supérieure est une fois de plus est dépassée. Pour la même raison comme ci-dessus, elle est en raison de choix $J_2 \longrightarrow J_1$ sur la machine 3. Il n'y a aucune autre possibilité à être examinée.

Mais dans le cas où J_1 est ordonnancé en premier puis J_2 sur la machine 1 ainsi que sur la machine 3. Cette solution excède aussi la borne supérieure.

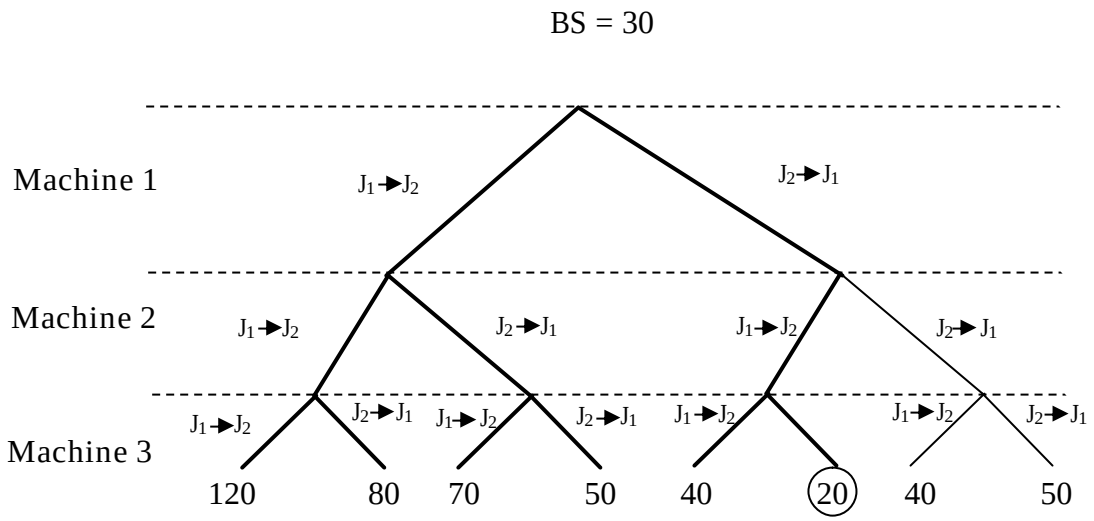


Fig. 2.2. Arbre de recherche binaire pour le jop-shop 2x 3

2.2.4. Principe de la propagation de contraintes

Pour résoudre le problème de satisfaction de contraintes, la programmation de contraintes est appliquée. Un CSP est décrit par un ensemble de variables, un ensemble de valeurs possibles (domaine) pour chaque variable, et un ensemble de contraintes liant ces variables. Le but est de trouver une attribution des valeurs de ces variables, de manière à satisfaire les contraintes. Ces contraintes peuvent être décrites implicitement, par exemple par une formule arithmétique ; ou explicitement, chaque contrainte est un ensemble de valeurs satisfaisant cette contrainte. La résolution se fait le plus souvent à travers un algorithme de recherche se basant sur un arbre de recherche où, à chaque étape, certaines valeurs sont instanciées par des valeurs. Une technique de retour en arrière permet de contrôler le développement de l'arborescence pour éviter son explosion, figure 2.3.

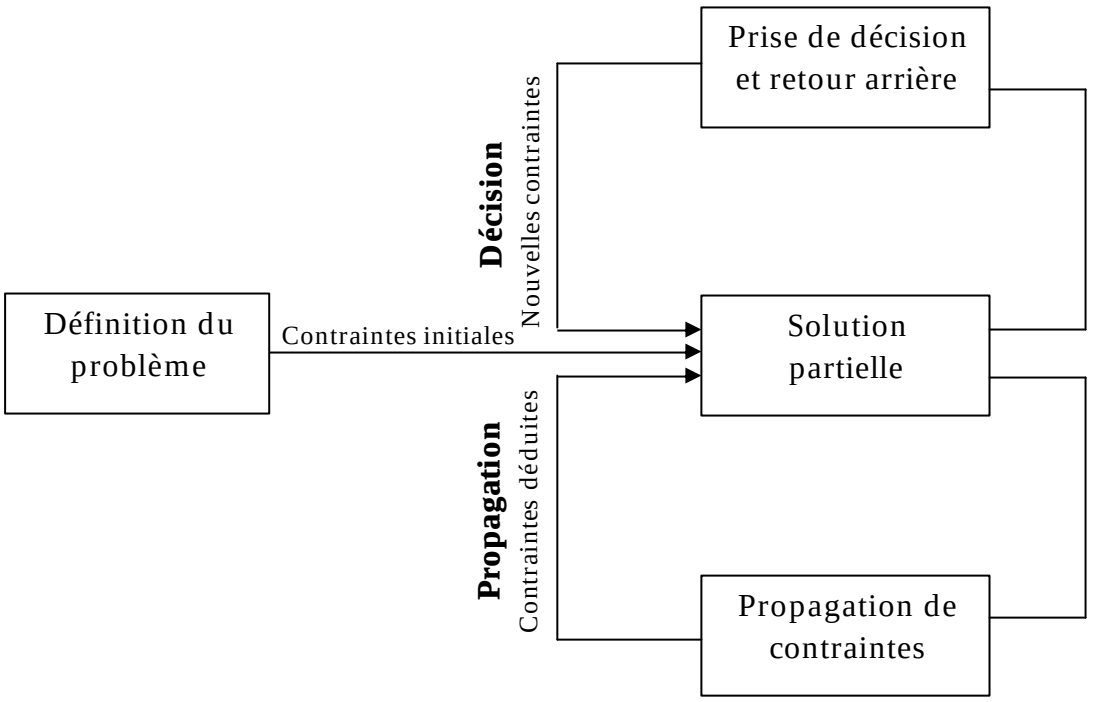


Fig. 2.3. Comportement d'un système de propagation de contraintes [Bap98]

∗ *Exemple*

Considérons le CSP binaire P = (X, D, C, R), avec :

- X = {x1, x2, x3, x4, x5},
- D = {d1, d2, d3, d4, d5} avec d1 = d2 = d3 = {1, 2, 3} et d4 = d5 = {1, 2},
- C = {c12, c13, c15, c23, c24, c35} avec cij = {xi, xj},
- R = {r12, r13, r15, r23, r24, r35}.

Les relations qui relient les variables sont les suivantes :

- r12 : $x1 \neq x2$,
- r13 : $x1 \neq x3$,
- r15 : $x1 \leq x5$,
- r23 : $x2 \neq x3$,
- r24 : $x2 \leq x4$,
- r35 : $x3 \leq x5$.

r12	
x1	x2
1	2
1	3
2	1
2	3
3	1
3	2

r13	
x1	x3
1	2
1	3
2	1
2	3
3	1
3	2

r15	
x1	x5
1	1
1	2
2	2

r23	
x2	x3
1	2
1	3
2	1
2	3
3	1
3	2

r24	
x2	x4
1	1
1	2
2	2

r35	
x3	x5
1	1
1	2
2	2

Le graphe de contraintes (X, C) associé à P est représenté dans la figure 2.4 suivante :

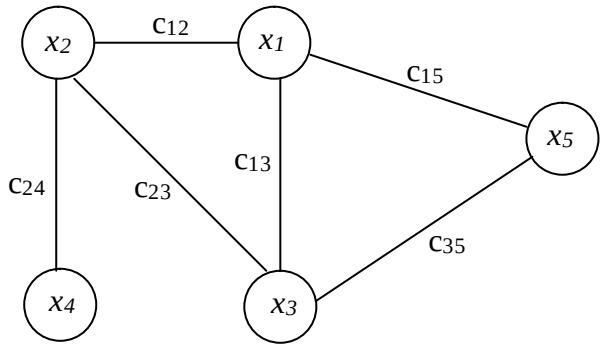


Fig.2.4. Graphe de contraintes associé à P

2.2.5. Propagation de contraintes et ordonnancement

Dans le problème d'ordonnancement disjonctif non préemptif, par exemple, si deux opérations O_i et O_j exigent la même ressource R, d'après la contrainte disjonctive, ces deux opérations ne peuvent pas être exécutées au même temps. Dans ce cas, elles sont liées par la contrainte de précédence suivante :
O_i précède O_j ou O_j précède O_i. Si n opérations requièrent la même ressource R alors il y aura $n * (n - 1) /$ (explicite ou implicite) contraintes disjonctives. Quant aux contraintes de temps , plusieurs variantes existent dans la littérature [Bap98]. Ce type de contrainte peut s'écrire :

$$\text{fin}(O_i) \leq \text{début}(O_j) \quad \text{ou} \quad \text{fin}(O_j) \leq \text{début}(O_i)$$

En effet, quand la plus petite date de fin possible de O_i dépasse la plus grande date de début possible de O_j, O_i ne peut pas précéder O_j et donc O_j doit précéder O_i.

De ce fait, les dates de début et de fin effectives de ces deux opérations doivent être mises à jour suivant cette contrainte. De la même façon, si la plus petite date de fin possible de O_j dépasse la plus grande date de début possible de O_i, O_j ne peut pas précéder O_i et donc O_i doit précéder O_j. Dans le cas où aucune des deux opérations ne peut précéder l'autre, une contradiction est détectée.

- Exemple

Pour cet exemple, deux opérations requièrent la même ressource et ne peuvent pas s'exécuter en même temps. L'opération O_1 précède l'opération O_2 ou inversement. La date de fin au plus tôt de chaque opération ne doit pas dépasser sa date au début au plus tard. La propagation de contrainte disjonctive impose que la date effective de fin de O_1 soit inférieure ou égale à la date effective de début de O_2, tableau 2.1.

Tab. 2.1. Contraintes temporelles

Avant propagation	r_i	d_i	p_i
O_1	0	4	2
O_2	1	5	2
propagation	r_i	d_i	p_i
O_1	0	3	2
O_2	2	5	2

où r_i, d_i et p_i représentant respectivement la date de début au plus tôt, la date de fin au plus tard et la durée opératoire de l'opération O_i.

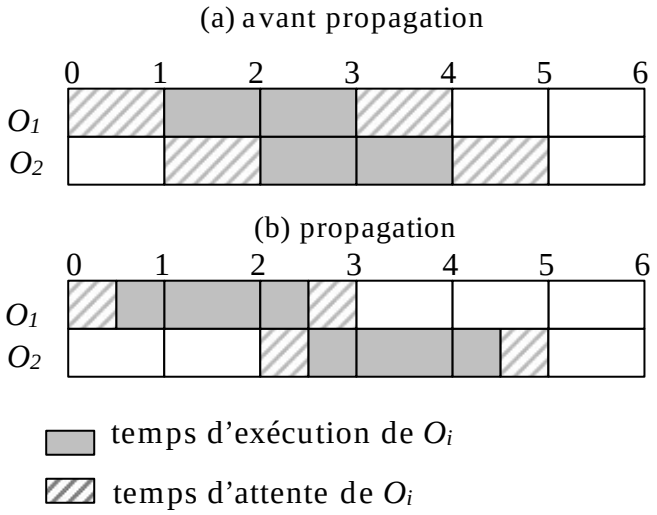


Fig. 2.5. Propagation de contrainte disjonctive

La figure 2.5 montre que, pour le cas (a), cette solution n'est pas envisageable car elle ne respecte pas la contrainte disjonctive (l'opération O_1 et l'opération O_2 s'exécutent au même temps entre les instants 2 et 3). Ce problème est résolu par la propagation de cette contrainte en mettant à jour les dates de début et de fin des opérations, cas (b).

2.3. Méthode « branch & bound »

2.3.1. Principe

C'est l'une des méthodes exactes d'optimisation qui peut résoudre le Problème de Satisfaction de Contraintes (CSP). Elle est basée sur un arbre de recherche. Le principe de la méthode «branch & bound » se traduit par deux concepts : le branchement, ou encore la séparation, et l'évaluation.

2.3.1.1. La séparation

La séparation consiste à décomposer un sommet représentant l'espace de solutions en sous-ensembles, cette séparation exige de ne pas perdre ni ajouter des solutions. En d'autres termes, cette séparation est basée sur le partage, tout en considérant le critère à optimiser, entre l'ensemble des solutions admissibles contenues dans un même sommet (nœud) de l'arborescence et des solutions irréalisables ou moins intéressantes par rapport aux solutions déjà obtenues.

2.3.1.2. L'évaluation

L'évaluation d'un sommet consiste à minorer ou à majorer les solutions associées afin d'éviter les branches inutiles c'est-à-dire ne conduisant pas à une solution. Ainsi, cette exploration intelligente de l'espace de recherche est réalisée grâce à des évaluations des branches et à des comparaisons avec une borne inférieure (un minorant) du critère à optimiser. Dans les problèmes de grande taille, il sera nécessaire d'affiner la valeur du minorant (majorant) pour éviter l'explosion de l'arbre de recherche.

2.3.2. Branch & bound et ordonnancement

Dans la littérature, la méthode de séparation et d'évaluation progressive, « branch & bound », a été appliquée dans plusieurs domaines et par beaucoup de chercheurs. Barker et McMahon [Bar85] et Carlier [Car89], entre autres, ont

contribué au progrès des approches exactes, qui sont principalement basées sur la méthode « branch & bound ».

Ils ont généralement envisagé des problèmes d'ordonnancement et ont comme repère des benchmarks comme « défi informatique » pour démontrer l'efficacité de leurs algorithmes, et la meilleure solution connue pour un problème a été améliorée. En 1989, Carlier et Pinson ont réussi à résoudre un problème d'ordonnancement de façon optimale par une méthode « branch & bound » [Car89]. Depuis lors, Brucker et al. [Bru94] [Bru97], Martin et Shmoys [Mar96], et Carlier [Car94] ont amélioré l'exécution et l'efficacité des approches exactes pour les problèmes NP-difficiles. Plus récemment, Rivereau [Riv99] a travaillé sur le problème flow-shop ; son travail a été inspiré de [Pot80] et [Car96], et Guéret et al. [Gué00], ont travaillé sur le problème, open-shop.

2.3.3. Algorithme général

Parmi les méthodes de résolution exactes de problèmes d'optimisation combinatoire et en particulier de problèmes d'ordonnancement, la procédure de séparation et d'évaluation progressive est la plus utilisée.

L'algorithme général correspondant est le suivant [Zri05]:

- diviser l'espace de recherche en sous-espaces (branches),
- chercher une borne supérieure (inférieure) d'une fonction objectif relative à chaque sous-espace de recherche,
- éliminer les « mauvais » sous-espaces (suivant le(s) critère(s) à optimiser),
- reproduire les étapes précédentes jusqu'à obtenir l'optimum global.

Un exemple d'algorithme pour un problème de minimisation est donné dans la figure 2.6.

Étape 1

Initialisation

Calculer une borne supérieure *BS*

Étape 2

Séparation

Sélectionner le sommet (nœud) à séparer et créer ses fils

Étape 3

Évaluation

Pour tous les sommets *S* créés faire

Si *S* représente une solution complète alors

calculer sa valeur pour le critère d'optimisation et mettre à jour *BS*

Sinon calculer la borne inférieure *BI*

Fin Si

Fin Pour

Étape 4

Élimination

Éliminer tout sommet tel que *BI* > *BS*

Étape 5

Critère d'arrêt

Si tous les sommets ont été éliminés alors arrêter

Sinon aller à l'étape 2

Fig.2.6. Algorithme général d'une PSE

2.3.4. Exemple d'algorithme

L'algorithme de la figure 2.7, proposé par Carlier [Car82], permet de résoudre de manière exacte des instances de problèmes de type $1|r_i|\max(C_i + q_i)$ relatif au problème à une machine avec des dates de début au plus tôt et dont l'objectif est de minimiser le makespan. La méthode possède la particularité de décrire un ordonnancement complet à chaque nœud et repose sur le principe que l'algorithme de Schrage [Sch71] il peut toujours fournir la solution optimale du problème à une machine si on modifie les données de ce dernier pour certaines opérations. Cet algorithme va donc permettre dans un premier temps de calculer la borne supérieure du problème et dans un second de générer l'ordonnancement d'un nœud. La séparation consiste à augmenter une contrainte sur une opération.

Notations

t_i **:** date effective de début d'exécution de l'opération i

r_i : date de début au plus tôt de l'opération i

p_i : durée opératoire de l'opération i

q_i : durée de latence de l'opération i

m : borne supérieure relative au makespan $C_{\max}$ (critère à optimiser)

C_i : date de fin d'exécution de l'opération i

Un problème d'ordonnancement à une machine, noté P_0, est défini par $\{r_i ; p_i ; q_i\}$;

1. Initialiser $P_a = P_0$ et $m = \infty$.
2. Appliquer une heuristique* *à* P_a et obtenir l'ordonnancement de Schrage [Sch71] et son makespan $C_{\max}(P_a)$, l'opération critique** c et l'ensemble critique*** J.
3. Si $C_{\max}(P_a) < m$, maintenir alors P_a comme meilleur ordonnancement, $P_b = P_a$ et mettre à jour $m = C_{\max}(P_a)$.
4. Générer P_l et P_r définis par $\{r_i^l, p_i^l, q_i^l\}$ *et* $\{r_i^r, p_i^r, q_i^r\}$ en utilisant respectivement les équations (2.1) et (2.2).
5. Calculer $l(P_l)$, la borne inférieure de P_l, comme $l(P_l) = \max\{C_{\max}(P_a), h_l(J), h_l(J \cup c)\}$ où h_l est calculé par (2.3) avec $h_l = h$ et $\{r_i, p_i, q_i\} = \{r_i^l, p_i^l, q_i^l\}$. Calculer $l(P_r)$ de la même manière que $l(P_l)$.
6. Ajouter le nouveau noeud P_l à l'arbre de recherche représentant un nœud enfant pour P_a si $l(P_l) < \mu$ et ajouter P_r si $l(P_r) < \mu$.
7. Mettre à jour P_a comme un noeud de la plus faible borne parmi des noeuds qui ne sont pas encore été visités.
8. Répéter de l'étape 2 à l'étape 7 jusqu'à ce qu'il n'y ait aucun noeud qui n'est pas encore visité.
9. P_b représente un ordonnancement optimal pour P_0.

Fig. 2.7. Algorithme PSE [Car82]

* l'heuristique MRW (Most Remaining Work) qui consiste à placer en priorité l'opération de plus grand q_i.

** une opération critique c est définie comme la dernière opération de l'ensemble critique J tel que $q_c < q_{j_n}$,

*** un ensemble critique J, $J = (j_1, j_2, ..., j_n)$, choisi de telle sorte qu'aucune opération ne s'exécute pendant l'intervalle de temps $[t_{j_1} - 1, t_{j_1}]$.

$$q_c = \max\left(q_c, \sum_{k \in J} \left(p_k^r + q_k^l \right) \right) \quad 2.1$$

$$r_c = \max\left(r_c, \min_{k \in J} r_k^r + \sum_{k \in J} p_k^r \right) \quad 2.2$$

$$h\left(I_1\right) = \min_{i \in I_1} r_i + \sum_{i \in I_1} p_i + \min_{i \in I_1} q_i \quad 2.3$$

2.4. Optimisation de la fonction de coût- Position du problème

2.4.1. Présentation

Les systèmes d'ordonnancement, utilisés jusqu'à maintenant, construisent, en règle générale, des ordonnancements imprécis à cause d'un manque de connaissance sur les particularités des ateliers et des lots de fabrication qui y accèdent [Vac00]. Le problème d'ordonnancement en industries agroalimentaires n'échappe pas à ces difficultés, bien au contraire il est plus complexe et plus difficile à résoudre. En effet, les principaux problèmes que peut rencontrer tout atelier de fabrication sont : le problème de la gestion des en-cours de fabrication, la répartition des charges des diverses ressources de l'atelier, que ce soit des ressources humaines ou matérielles et la réduction du retard ou de l'avance durant le processus de fabrication. Il s'agit donc de développer un outil d'aide à l'ordonnancement dynamique adapté à l'industrie agroalimentaire, afin de tenir compte des différentes contraintes et des fréquents aléas dus à la nature des produits manipulés à savoir les produits agroalimentaires. Ces derniers ont la particularité d'avoir des dates limites de validité assez courtes.

2.4.2. Système d'ordonnancement réactif en temps réel

Un système d'ordonnancement réactif est un système capable de rétablir un ordonnancement rapidement et efficacement en réponse à un évènement inattendu [Lio98]. En effet, plusieurs problèmes d'ordonnancement se caractérisent par des perturbations de fonctionnement. Ces aléas sont généralement détectables lors de la phase de la mise en œuvre d'une solution d'ordonnancement.

D'autre part et dans un cadre multicritère, d'autres aléas peuvent également avoir lieu. Ce sont les incertitudes concernant l'expression des choix et des préférences donnés soit par un décideur soit par un utilisateur [Kac03]. L'ordonnancement temps réel est alors très complexe vu le nombre d'informations à traiter et l'influence de ces perturbations et aléas au cours de la production. C'est pour ces raisons que l'intervention humaine à ce niveau est cruciale. Il s'ensuit qu'une interaction suffisante et efficace est nécessaire pour pouvoir agir devant deux problèmes importants à la fois, à savoir l'optimisation et l'évaluation. La figure 2.7 suivante présente un système d'ordonnancement temps réel.

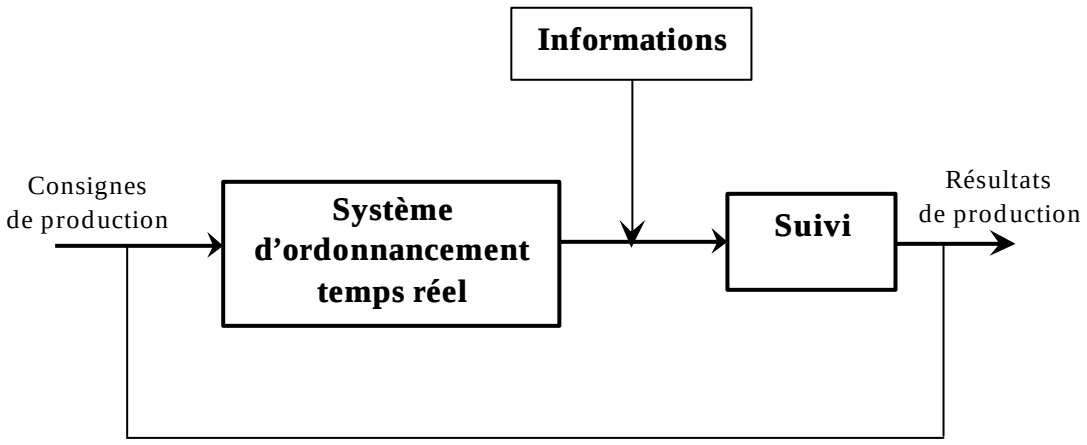


Fig. 2.7. Comportement d'un système d'ordonnancement temps réel

2.5. Application de la méthode « branch & bound » dans les industries agroalimentaires

2.5.1. Introduction

Actuellement, toute entreprise se doit d'être performante par rapport aux entreprises concurrentes que se soit sur le plan technique soit sur le plan économique. L'amélioration de la gestion de production est à la base de toute recherche dans le domaine de la productivité à accroître. La productivité est la clef de la survie et du développement des entreprises industrielles.

La loi du marché nécessite une remise en cause des produits fabriqués. Les transformations et les améliorations de produits, sont de plus en plus fréquentes et la notion de flexibilité est fortement liée à cette évolution. Cette flexibilité est nécessaire surtout dans les industries agroalimentaires. La concurrence exige que l'entreprise de production agroalimentaire, en particulier, se dote d'un système de production efficace qui réagisse rapidement aux contraintes et à son environnement et en particulier aux exigences et aux évolutions du marché.

Deux types d'évolutions non antagonistes, bien au contraire, semblent émerger pour tenter de répondre au problème de la survie de l'entreprise [Vac00]:

3. l'intégration cohérente de la gestion de production avec une importance particulière pour la transmission des informations et de la communication ;
4. l'automatisation de la fabrication pour l'amélioration de l'exécution des tâches et de l'ensemble des flux physiques.

Parmi les différentes fonctions de la gestion de production, nous nous intéressons plus particulièrement à *l'ordonnancement dynamique dans les industries agroalimentaires.*

2.5.2. Description du problème d'ordonnancement dynamique

Etant donné un ensemble *E* de *n* opérations à ordonnancer sur une même machine avec pour but d'optimiser certain(s) critère(s), l'algorithme « branch and bound » est établi en construisant dynamiquement un arbre de recherche. La racine de l'arbre, constituant le niveau 0, est un noeud vide ne possédant aucune opération. De ce nœud, il y a *k* noeuds enfants (branches) représentant tous les ordonnancements admissibles. Au niveau *j* (*j* > 1), quand la prochaine opération est considérée, la relation de dominance entre les opérations est examinée, l'opération dominante étant celle qui satisfait au mieux le(s) critère(s) considéré(s). Si la relation de dominance est satisfaite, la recherche continue dans cette branche. Si celle-ci n'est pas satisfaite, la recherche est avortée pour cette branche (branche élaguée). En général, les relations de dominance améliorent l'efficacité d'un algorithme « branch-and-bound » en contraignant l'espace de recherche. Pendant la recherche, n'importe quelle branche inachevée, qui a une borne inférieure qui est supérieure ou égale à la borne supérieure courante, est sondée. La recherche suit une première stratégie en profondeur puis horizontalement [Tan06c].

2.5.3. Optimisation de la fonction « objectif »

2.5.3.1. Notion de dominance

En ordonnancement, le concept de dominance d'un sous-ensemble de solutions est important afin de limiter la complexité algorithmique liée à la recherche d'une solution optimale au sein d'un grand ensemble de solutions. Pour trouver une solution optimale aux problèmes d'optimisation multi-objectifs, constituant un ensemble de points, il s'avère nécessaire de définir une relation d'ordre entre ces éléments dite relation de dominance, pour identifier les meilleurs compromis. La règle de dominance est une contrainte qui peut être ajoutée au problème initial sans changer la valeur de l'optimum [Jou02]. La plus utilisée est celle définie au « sens de Pareto » [Bar03].

2.5.3.2. Position du problème

Il s'agit de construire un ordonnancement multicritère adapté à l'industrie agroalimentaire. Parmi les contraintes et les critères spécifiques à cette industrie, on distingue la périssabilité des produits et le discount de distribution. L'objectif est alors de sélectionner parmi l'ensemble des opérations candidates à ordonnancer, celle qui présente le meilleur compromis entre les différents critères en réduisant et en filtrant l'espace de recherche initial. La décision d'éliminer ou de maintenir une opération permet d'éviter la péremption de certains composants et entraîne la diminution des coûts de ces composants périmés ainsi que du coût de discount de distribution.

2.5.3.3. Schéma de séparation

Le schéma de séparation utilisé pour notre application concernant l'optimisation de la fonction de coût pour un atelier de production agroalimentaire, est inspiré de [Riv99]. La procédure arborescente tente de construire la meilleure permutation qui donne la séquence optimale «S_i» en fixant progressivement des opérations depuis le début et la fin de la séquence. Ainsi, tout nœud i de l'arbre de recherche est identifié par un triplet(t,g,Ω) où t est la date indiquant le début de la séquence, g la fin de l'ordonnancement et Ω l'ensemble des opérations non encore ordonnancées.

Soit $BI(t,g,\Omega)$ une borne inférieure sur le nœud i (t,g,Ω). Les ensembles E et S représentent des nœuds d'entrées et de sorties associées à (t,g,Ω). Ces nœuds sont définis par :

$$E(t,g,\Omega) = \{j \in \Omega \ / BI(t(j), \Omega \setminus \{j\}, g) = BS\}$$

$$S(t,g,\Omega) = \{j \in \Omega \ / BI(t, \Omega \setminus \{j\}, g(j)) = BS\}$$

où BS représente une borne supérieure du critère à optimiser.

Une fois ces deux ensembles calculés, un nouveau nœud est établi en fixant soit une nouvelle entrée si le cardinal de *E* est inférieur à celui de *S*, soit une nouvelle sortie dans le cas contraire.

2.5.3.4. Application des règles de dominance des opérations à l'ordonnancement

❖ **Formulation du problème**

Soit un ensemble *E* de n opérations à ordonnancer entre deux séquences P et A d'opérations déjà ordonnancées. Pour un couple d'opérations O_i et O_j de l'ensemble *E* des opérations candidates à l'ordonnancement, le problème est de déterminer laquelle de ces opérations est à ordonnancer en premier, c'est à dire l'opération dominante, dont le but est de minimiser :

- le coût des produits périmés, engendré par la péremption de certains composants primaires nécessaires pour la fabrication du produit fini P ;
- le coût du discount de distribution représentant la perte sur le prix de vente du produit fini, proportionnel à la durée du stockage du produit fini avant sa livraison.

Il s'agit donc, de filtrer l'espace de recherche pour construire un ordonnancement qui satisfait ces critères tout en respectant les contraintes [Gar01].

❖ **Calcul des coûts relatifs aux produits périmés et au discount de distribution**

Dans le cas général, le coût des produits périmés $K_1(S)$ et celui du discount de distribution $K_2(S)$ s'écrivent comme suit [Tan06b] :

$$K_1(S) = \sum_i a_i \sum_j \sum_k P_{ijk}^{rev} \left(\frac{\max\left(0, t_{ij} - v_{ijk}\right)}{\left(t_{ij} - v_{ijk}\right)} \right) \qquad 2.4$$

$$K_2(S) = \sum_i b_i \max\left(0, d_{P_i}^{liv} - C_{P_i}\right) \times \left(\frac{P_{P_i}^{ven}}{Dv_{P_i} - Dr_{P_i}} + C_{P_i}^{stk} \right) \qquad 2.5$$

La fonction de coût, représentant le coût total, est la somme de ces deux coûts :

$$K^{tot}(S) = K_1(S) + K_2(S) \qquad 2.6$$

Remarque : Les coefficients a_i et b_i favorisent une gamme par rapport à une autre ; par exemple, dans le cas où une commande urgente est plus intéressante (du point de vue coût) qu'une autre qui est en cours de production, la commande qui est en urgence est alors favorisée et lancée en production alors que la production de l'autre gamme est interrompue.

2.5.3.5. Exploration et étude des différents cas d'ordonnancement

Afin de mettre en œuvre la méthode proposée, est traité l'exemple donné au paragraphe précédent. Pour avoir le meilleur ordonnancement qui optimise les critères cités, tous les cas réalisables sont, dans cette partie, étudiés puis comparés, figure 2.8. Cet ordonnancement est ensuite déterminé en vue de son exploitation.

- *cas 1* : Ordonnancement initial S_1,
- *cas* 2 : Ordonnancement S_2, relatif à l'échange entre l'opération O_{ij} et l'opération O_{ik},
- *cas 3* : Ordonnancement S_3, relatif à l'insertion de l'opération O_{ik} juste après la séquence P,
- *cas 4* : Ordonnancement S_4, relatif à la permutation entre l'opération O_{ij} et l'opération O_{ik} du cas 3

Remarque : pour cet exemple on trouve quatre ces réalisables en respectant la contrainte de précédence qui exige que la séquence d'opérations P est toujours en premier à ordonnancer et que les deux séquences A et P ne doivent pas être successives.

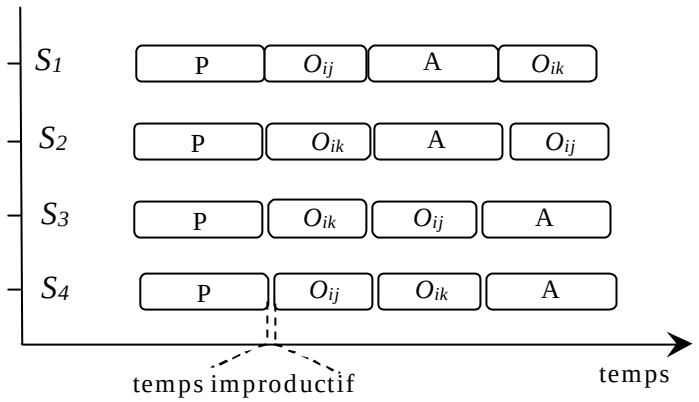


Fig.2.8. Cas d'ordonnancements

2.5.3.6. Mise à jour des dates

- ***1er cas***

Pour l'ordonnancement S_1 où les séquences A et P sont déjà ordonnancées, le coût des produits périmés $K_1(S)$ est formulé par l'expression (2.4) et le coût du discount de distribution $K_2(S)$ est formulé par l'expression (2.5).

- ***2ème cas***

L'échange entre les opérations O_{ij} et O_{ik}, tout en gardant la séquence A, conduit à l'ordonnancement S_2. Dans ce cas, il faut mettre à jour les dates de début et de fin des opérations.

Soient :

t_{ik} , date effective de début de l'opération O_{ik}, $t_{ik} = \max\left(d_{fP}, r_{ik}\right)$

g_{ik} , date de fin de l'opération O_{ik}, $g_{ik} = \max\left(d_{fP}, r_{ik}\right) + p_{ik}$

t_{2A} , date de début effective de la séquence A, $t_{2A} = \max\left(g_{ik}, r_A\right)$

g_{2A} , date de fin effective de la séquence A, $g_{2A} = \max(g_{ik}, r_A) + p_A$

t_{ij} , date effective de début de O_{ij}, $t_{ij} = \max(g_{2A}, r_{ij})$

g_{ij} , date de fin de O_{ij}, $g_{ij} = \max(g_{2A}, r_{ij}) + p_{ij}$

- ***3ème cas***

Pour le cas de l'insertion de l'opération O_{ik} juste après la séquence P, l'ordonnancement S_3 est obtenu et les dates sont mises à jour comme suit :

t_{ik} , date effective de début de l'opération O_{ik}, $t_{ik} = \max(d_{fP}, r_{ik})$

g_{ik} , date de fin effective de l'opération O_{ik}, $g_{ik} = \max(d_{fP}, r_{ik}) + p_{ik}$

t_{ij} , date effective de début de O_{ij}, $t_{ij} = \max(g_{ik}, r_{ij})$

g_{ij} , date de fin de O_{ij} ,$g_{ij} = \max(g_{ik}, r_{ij}) + p_{ij}$

t_{3A} , date effective de début de la séquence A, $t_{3A} = \max(g_{ij}, r_A)$

g_{3A} , date de fin effective de la séquence A, $g_{3A} = \max(g_{ij}, r_A) + p_A$

- ***4ème cas***

Pour le cas de l'insertion de l'opération O_{ik} juste après l'opération O_{ij}, l'ordonnancement S_4 est obtenu et les dates sont mises à jour comme suit :

t_{ij} , date effective de début de l'opération O_{ij}, $t_{ij} = \max(d_{fP}, r_{ij})$

g_{ij} , date de fin effective de l'opération O_{ij}, $g_{ij} = \max(d_{fP}, r_{ij}) + p_{ij}$

t_{ik} , date effective de début de O_{ik}, $t_{ik} = \max(g_{ij}, r_{ik})$

g_{ik} , date de fin effective de O_{ik}, $g_{ik} = \max(g_{ij}, r_{ik}) + p_{ik}$

t_{4A} , date effective de début de la séquence A, $t_{4A} = \max(g_{ik}, r_A)$

g_{4A} , date de fin effective de la séquence A, $g_{4A} = \max(g_{ik}, r_A) + p_A$

2.5.3.7. Application

Considérons l'exemple suivant relatif à une gamme de quatre produits P_i, i= {1,…,4} composé chacun de :

- l'opération O_{i1} est formée de trois composants : c_{i11}, c_{i12} *et* c_{i13} ;
  $O_{i1} = \{c_{i11}, c_{i12}, c_{i13}\}$
- l'opération O_{i2} est formée de deux composants c_{i21} *et* c_{i22} ; $O_{i2} = \{c_{i21}, c_{i22}\}$
- la séquence A est composée de deux opérations O_{i3} et O_{i4}, telles que
  $O_{i3} = \{c_{i31}\}$ et $O_{i4} = \{c_{i41}, c_{i42}\}$
- $a_i = b_i = 1$

Pour cet exemple, le coût des produits périmés peut être formulé comme suit :

$$K_1(S) = \sum_{i=1}^{4}\sum_{k=1}^{3} P_{i1k}\frac{\max(0, t_{i1} - v_{i1k})}{(t_{i1} - v_{i1k})} + \sum_{i=1}^{4}\sum_{k=1}^{2} P_{i2k}\frac{\max(0, t_{i2} - v_{i2k})}{(t_{i2} - v_{i2k})} + \sum_{i=1}^{4} P_{i3k}\frac{\max(0, t_{i3} - v_{i3k})}{(t_{i3} - v_{i3k})} + \sum_{i=1}^{4}\sum_{k=1}^{2} P_{i4k}\frac{\max(0, t_{i4} - v_{i4k})}{(t_{i4} - v_{i4k})}$$

et le coût du discount de distribution par :

$$K_2(S) = \sum_{i=1}^{4} \max\left(0, d_{P_i}^{liv} - C_{P_i}\right) \times \left(\frac{P_{P_i}^{ven}}{Dv_{P_i} - Dr_{P_i}} + C_{P_i}^{stk}\right)$$

Les données relatives à cet exemple sont présentées dans le tableau 2.2.

Tab. 2.2.

	O_1	O_2	O_3	O_4
r_{ij}	1	2	4	4
p_{ij}	2	3	6	6
v_{ij1}	2	1	2	5
v_{ij2}	3	2	-	6
v_{ij3}	4	-	-	-
P_{ij1}^{rev}	2	3	4	3
P_{ij2}^{rev}	1	2	-	5
P_{ij3}^{rev}	1	-	-	-
$P_{P_i}^{ven}$	7	6	8	8
Dv_{P_i}	16	13	12	12
Dr_{P_i}	12	9	10	10
$d_{P_i}^{liv}$	8	13	10	10
$C_{P_i}^{stk}$	5	2	1	1

En appliquant la méthode de résolution proposée, les ordonnancements non réalisables sont alors élimés et les coûts relatifs à chaque ordonnancement calculés. Il vient les résultats expérimentaux consignés dans le tableau 2.3.

Tab. 2.3. Résultats expérimentaux

séquences			K_1	K_2	K^{tot}
O_1	O_2	A	7	61.75	68.75
O_1	A	O_2	9	40.75	49.75
O_2	O_1	A	9	48.25	57.25
O_2	A	O_1	11	28.00	39.00

Ces résultats montrent que la différence entre le coût minimum et le coût maximum est significative. La méthode de résolution proposée a aussi permis d'obtenir un bénéfice intéressant. Ce gain considérable se traduit par l'écart entre le coût minimum, 39, et le coût maximum, 68. 75. la meilleure solution, à retenir, est donc l'ordonnancement : P-O_2-A-O_1.

2.6. Conclusion

L'approche développée dans ce travail a permis de choisir, parmi plusieurs solutions possibles, celle qui représente l'ordonnancement dont la fonction de coût constituée du coût des produits périmés et du coût du discount de distribution, est minimale. Cette solution engendre un gain considérable vis-à-vis des critères considérés du fait de la particularité des produits fabriqués en industries agroalimentaires. Ainsi, moyennant des règles de dominance des opérations et des paramètres nécessaires pour le calcul des coûts et des données de stock, peut être évitée la péremption de certains composants et produits semi-finis, en maintenant dans l'espace de recherche les opérations dont les composants possèdent les dates limites de validité les plus courtes. Au contraire, les opérations dont la durée de vie est suffisamment longue et dont la réalisation peut engendrer un retard de mise en fabrication d'autres opérations, sont éliminées de l'espace de recherche.

Dans le chapitre suivant, il s'agit de développer et d'appliquer deux méthodes approchées : les algorithmes génétiques et les algorithmes d'optimisation par colonie de fourmis, afin de construire un ordonnancement dynamique adapté aux industries agroalimentaires.

Chapitre 3

Optimisation multicritère pour la construction d'un ordonnancement dynamique

3.1. Introduction

Un système d'aide à décision est développé dans cette partie. L'objectif d'un tel processus est de trouver ou de déterminer la solution optimale. Il s'agit de construire un ordonnancement multicritère dynamique adapté aux ateliers de production agroalimentaire. Ce système a pour but de choisir et de maintenir dans l'espace de recherche l'ensemble des « meilleures » opérations au sens des critères retenus. Deux approches ont été appliquées pour atteindre cet objectif. La première approche se base sur les algorithmes génétiques et la deuxième représente les algorithmes d'optimisation par colonie de fourmis.

3.2. Optimisation par les algorithmes génétiques

3.2.1. Introduction

Les Algorithmes Génétiques (AGs) sont des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique.

Ils utilisent à la fois les principes de la survie des structures les mieux adaptées, et les échanges d'informations aléatoires, parfois guidées, pour former un algorithme

d'exploration qui possède certaines des caractéristiques de l'exploration humaine [Gol94]. Ils ont été développés par John Holland [Hol75] à l'Université de Michigan avec deux objectifs à atteindre :

- mettre en évidence et expliquer rigoureusement les processus d'adaptation des systèmes naturels,
- concevoir des systèmes artificiels (en l'occurrence des logiciels) qui possèdent les propriétés importantes des systèmes naturels.

Caractérisés par leur propre théorie et structure, les AGs se sont différenciés des autres méthodes d'optimisation traditionnelles par leur mode d'action. Un AG modélise le processus d'évolution collectif d'une population d'individus pour s'adapter à un environnement [Mes99]. Chaque individu va, non seulement représenter un point de l'espace des solutions du problème, mais aussi contenir la connaissance actuelle de l'individu par rapport à l'environnement. La recherche sur les AGs a pour souci principal l'amélioration de la robustesse, l'équilibre entre la performance et le coût nécessaire à la survie dans des environnements nombreux et différents. Ayant été reconnus comme une approche valide des problèmes nécessitant une exploration performante et économique du point de vue du calcul, les algorithmes génétiques sont maintenant appliqués dans de nombreux domaines : l'ingénierie, la distribution (le problème de voyageur de commerce), l'ordonnancement, la logistique et la recherche scientifique en optimisation en général.

3.2.2. Principe général des algorithmes génétiques

3.2.2.1. Concepts de base

Les AGs constituent une classe de stratégies de recherche réalisant un compromis entre l'exploration et l'exploitation. Ils représentent des méthodes qui utilisent un choix aléatoire comme outil pour guider une exploration hautement intelligente dans l'espace des paramètres codés [Mes99].

Ce sont des algorithmes itératifs de recherche globale dont l'objectif est d'optimiser une fonction prédéfinie, appelée fonction coût ou fonction *fitness*.
Les algorithmes génétiques emploient un vocabulaire emprunté à la génétique naturelle. Ils travaillent sur un ensemble d'individus appelé *population*. Un individu a deux représentations, appelées *phénotype* et *génotype*. Le phénotype représente une solution potentielle du problème à optimiser en utilisant la formulation originale du problème. Le génotype donne une représentation codée d'une solution potentielle sous la forme d'un *chromosome*. Un chromosome est formé de *gènes* disposés en une succession linéaire et chaque gène peut prendre plusieurs valeurs appelées *allèles*.

Par exemple, un chromosome se compose d'une succession de 0 et de 1 (c.-à-d. une chaîne binaire), et la valeur pour une certaine position correspond à *on* (la valeur = 1) ou à *off* (la valeur = 0) d'un certain dispositif. Des formes plus sophistiquées, telles qu'un ensemble de symboles et une permutation des alphabets, sont choisis pour décrire les chromosomes du problème à optimiser.
Chaque individu a une fonction objectif *f*, fonction fitness, qui mesure l'adaptation de l'individu par rapport à son environnement local. La théorie darwinienne indique que, parmi des individus d'une population, celui qui est le mieux adapté à l'environnement local a le plus de chance de survivre et d'avoir un plus grand nombre de descendants : c'est la règle de la « survie du plus fort ». Ainsi, la fonction objectif *f* du problème d'optimisation joue le rôle d'un critère d'adaptation [Yam03]. Un des points les plus importants des algorithmes génétiques est la flexibilité dans la fonction objectif.
Un AG peut être considéré à la base comme un processus aléatoire. Cependant, les informations qui viennent des fonctions objectifs sont toujours utilisées pour paramétrer ce processus. Le travail de recherche commence en plusieurs points dans l'espace des solutions, c'est-à-dire sur une population de points et non pas en un point singulier comme dans la plupart des techniques d'optimisation [Vac00].

3.2.2.2. Les opérateurs

Un algorithme génétique simple utilise les trois opérateurs suivants : la sélection, le croisement et la mutation.

- ***L'opérateur de sélection***

La sélection est un processus dans lequel des individus dans une population sont choisis selon les valeurs de leurs fonctions coût ou « fitness » pour former une nouvelle population. Les individus *évoluent* par des itérations successives de reproduction et de sélection, appelées *générations.* Chaque individu est sélectionné proportionnellement à sa fonction « fitness », donc, un individu avec une fonction « fitness » plus élevée aura plus de chance d'être sélectionné qu'un autre avec une valeur de «fitness » inférieure. Cette fonction peut être envisagée comme une mesure de profit ou de qualité qu'on souhaite maximiser.

Un opérateur simple de sélection est la technique de la roulette pondérée où chaque individu d'une population occupe une surface de la roulette proportionnelle à sa valeur de la fonction «fitness ». Pour la reproduction, les candidats sont sélectionnés avec une probabilité proportionnelle à leurs « fitness ». Pour chaque sélection d'un individu, une simple rotation de la roue donne le candidat sélectionné. Cependant cette sélection n'est pas parfaite. En effet le risque de favoriser un individu ou un petit ensemble d'individus constitue un inconvénient qui risque d'appauvrir la diversité de la population.

- ***L'opérateur de croisement***

Le croisement est un opérateur de recombinaison. Les individus dans une population sont couplés au hasard par paires qui représentent les parents. Chaque paire d'individu subit *le* croisement, opérant sur les génotypes (c'est-à-dire chromosomes) de deux individus appelés parents. Il produit de nouveaux individus (généralement deux) appelés enfants dont les gènes sont hérités de l'un ou / et de l'autre parent.

Ceci peut être fait en dédoublant chacun des deux chromosomes dans des fragments et les recombinant pour former de nouveaux chromosomes.

- ***Le croisement à un point***

Le croisement à un point place un point de croisement au hasard, dans le cas où le génotype est une chaîne binaire de longueur *n*. Un enfant prend une section avant le point de croisement d'un parent et prend l'autre section après le point de croisement de l'autre parent puis recombine les deux sections pour former une nouvelle chaîne binaire. L'autre enfant se construit de façon inverse.

Exemple

Considérons *P1* et *P*2 deux chaînes binaires de longueur $n = 7$ correspondant aux parents :

$$P1 = 0000 \mid \mathbf{001}$$

$$P2 = 1111 \mid \mathbf{110}$$

Le symbole | indique un point de croisement, et dans ce cas il est placé après le quatrième bit.

Le croisement à un point crée les deux nouveaux individus *E1* et *E2* comme suit :

$$E1 = 0000 \mid \mathbf{110}$$

$$E1 = 1111 \mid \mathbf{001}$$

- ***Le croisement à deux points***

Le croisement à deux points place deux points de croisement au hasard, et prend une section entre les points d'un parent et les autres sections en dehors des points de l'autre parent puis les recombine. Dans l'exemple suivant, les deux points de croisement sont placés respectivement après le premier et quatrième bit.

P1 = 0 | **000** | 001

P2 = 1 | **111** | 110

Le croisement à deux points résultant conduit aux deux individus suivants :

E1 = 0 | **111** | 001

E2 = 1 | **000** | 110

- ***Le croisement uniforme***

Ce type de croisement a été proposé par Syswerda [Sys91]. Il consiste à choisir avec la même probabilité un allèle de l'un ou de l'autre parent, pour transmettre sa valeur à la même position, aux enfants.

Exemple

P1 = 0000001

P2 = 1111110

M = **1101001**

E1 = 00**1**0**11**1

E2 = 11**0**1**00**0

M représente le masque de transmission dont le principe est le suivant : si la valeur dans le masque est égale à 1, alors la valeur de l'allèle du parent 1 passe à l'enfant 1 (respectivement du parent 2 passe à l'enfant 2) ; sinon la valeur de l'allèle du parent 1 passe à l'enfant 2 (respectivement du parent 2 passe à l'enfant 1), tout en respectant la même position de l'allèle.

- ***L'opérateur de mutation***

La mutation opère sur le génotype d'un seul individu. Elle correspond à une « erreur » produite quand le chromosome est copié et reproduit ; c'est-à-dire, pour une chaîne binaire, elle consiste, par exemple, à faire pour un allèle un échange

entre le « 0 » et le « 1 ». Si des copies exactes sont toujours garanties, alors le taux de mutation est égal à zéro. Cependant, dans la vie réelle, l'erreur de copie peut se produire dans des circonstances spécifiques comme la présence du bruit. La mutation change des valeurs de certains gènes avec une faible probabilité. Elle n'améliore pas, en général, les solutions mais évite une perte irréparable de la diversité.

Exemple

A = 10**0**00 (avant mutation)

A' = 10**1**00 (après mutation)

Dans cet exemple il y a un échange entre le 0 et le 1 du troisième bit de l'individu A pour obtenir l'individu A'.

3.2.2.3. Schéma de principe et algorithme général

- **Schéma de principe**

Pour un problème d'optimisation donné, un individu représente un point de l'espace de recherche, une solution potentielle, la valeur du critère à optimiser, son *adaptation*, lui est associée. Ensuite, d'une façon itérative, figure 3.1, les processus de sélection, de croisement et de mutation, sont appliqués aux populations d'individus. La sélection a pour but de favoriser les meilleurs éléments de la population pour le critère considéré (les mieux *adaptés*), le croisement et la mutation assurent l'exploration de l'espace de recherche.

Au début, une population aléatoire d'individus est générée. Pour passer d'une génération *i* à la génération *i+1*, les opérations suivantes sont effectuées. Dans un premier temps, la population est reproduite par *sélection* où les « bons » individus se reproduisent mieux que les mauvais, au sens du critère considéré. Ensuite, un *croisement* aux paires d'individus (les parents) d'une certaine proportion de la population (une probabilité P_c généralement autour de 0.6), est appliqué, pour en produire des nouveaux (les enfants). Un opérateur de *mutation* est également

appliqué à une certaine proportion de la population (probabilité P_m, généralement très inférieure à P_c). Enfin, les nouveaux individus sont évalués et intégrés à la population de la génération suivante. Plusieurs critères d'arrêt de l'algorithme sont possibles : le nombre de générations peut être fixé a priori (temps constant) ou l'algorithme peut être arrêté lorsque la population n'évolue plus suffisamment rapidement. Pour utiliser un algorithme génétique sur un problème d'optimisation, un principe de codage des individus, un mécanisme de génération de la population initiale et d'opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche, doit donc être disposés [Bar99].

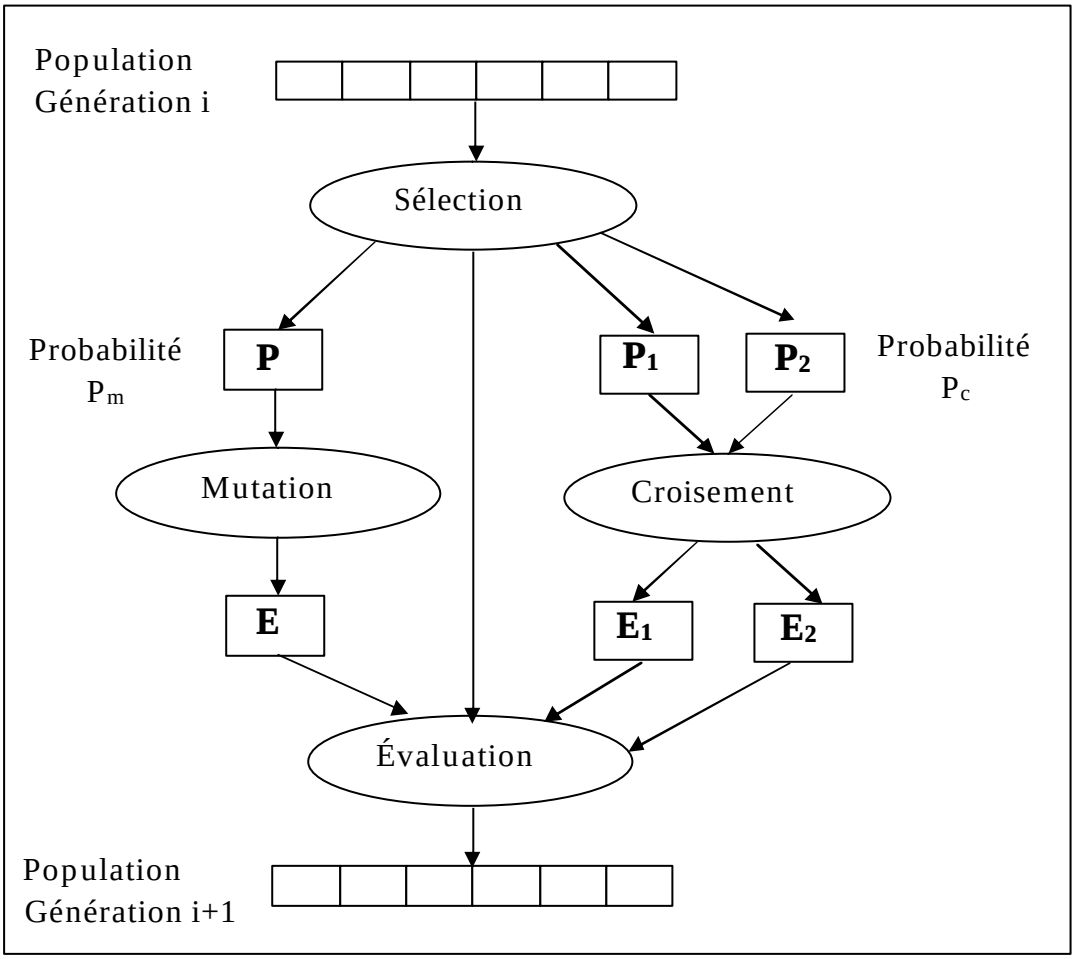


Fig.3.1. Principe général d'un algorithme génétique

3.2.2.4. Algorithme général

L'algorithme général correspondant est le suivant :

Étape 1 : Produire une population initiale d'individus.

Étape 2 : Exécuter les étapes suivantes jusqu'à satisfaction du critère d'arrêt :

1. Assigner une valeur d'aptitude à chaque individu de la population en utilisant la fonction « fitness »
2. Créer une nouvelle population d'individus en appliquant les opérateurs génétiques suivants. Ces opérateurs sont appliqués à des chromosomes choisis de la population avec une probabilité choisie sur l'aptitude :

- *Sélection* : reproduire un individu existant en le copiant dans la nouvelle génération ;
- *Croisement* : créer deux nouveaux individus à partir de deux individus existants par recombinaison génétique de leurs chromosomes par l'opérateur de croisement ;
- *Mutation* : créer un nouvel individu à partir d'un individu existant en subissant une mutation ;

Étape 3: L'individu qui est désigné par la méthode de d'estimation du résultat est retourné comme étant le meilleur chromosome produit (meilleure solution).

3.2.3. Application des algorithmes génétiques en ordonnancement

Cette approche a été largement utilisée ces dernières années [Cab00] [Yam03], [Ash04], etc. L'utilisation des AGs dans des nombreux domaines a fait ses preuves notamment dans des problèmes combinatoires tels que les problèmes d'ordonnancement [Dav85], [Nak91], [Yam92], [Del95]. Les problèmes d'ordonnancement d'un atelier classique de type Jop-Shop (JSP) ont été largement étudiés et résolus par les AGs [Ghe99], [Che96], etc. D'autres algorithmes hybrides

ont été aussi proposés [Cav98] [Jai97]. La difficulté principale dans la résolution de ces types de problèmes résulte dans la façon avec laquelle ils sont représentés sous forme algorithmique. Cette phase représente le point le plus important dans la recherche génétique. Plusieurs approches de représentation et différents types d'opérateurs d'AGs ont été proposés, pour résoudre ces problèmes.

3.2.3.1. Le codage dans les ateliers de type jop-shop

Dans la littérature, plusieurs codages ont été proposés pour le job-shop classique et le jop-shop flexible. Yamada [Yam03], a proposé un codage qui représente les dates de fin des opérations pour chaque tâche. Tamaki [Tam92] à utilisé un codage binaire traduisant la représentation de la solution en graphe disjonctif. Dans le codage de Kobayachi [Kob95], les séquences des opérations par machine sont uniquement représentées. Enfin, Portmann [Por88] propose un codage indirect sous forme d'écriture matricielle de la présence d'un enchaînement entre deux opérations consécutives.

Pour le job-shop flexible, Ghedjati [Ghe94] propose un codage basé sur l'affectation d'une heuristique de choix à chaque ressource. Ces heuristiques permettent à la ressource de choisir la tâche à réaliser. Ainsi, avec cette représentation un chromosome contient autant de gènes que de ressources. Mesghouni [Mes99] propose aussi deux codages représentant des extensions de codages existants. Le premier utilise le codage de Kobayashi en ajoutant pour chaque opération d'une séquence, l'ordre de l'opération dans la gamme et sa date de début d'exécution. Le deuxième, une extension du codage de Yamada introduit la machine à laquelle sont affectées les opérations pour chaque tâche. Enfin Kacem [Kac03], propose trois codages, un codage opérations machines qui donne les dates de début et de fin de chaque opération sur la machine à laquelle est affectée l'opération, le deuxième codage est le codage liste des opérations qui représente l'ordonnancement dans un tableau de trois colonnes, opération, machine capable d'exécuter l'opération et une troisième colonne pour les dates de fin.

Le troisième codage est celui des séquencements des jobs représentant l'ordonnancement en *n* colonnes (où *n* est le nombre maximum d'opérations que peut contenir une tâche). Chaque colonne représente les tâches à ordonnancer sous forme d'une liste de *x* cellules (ou *x* est le nombre de tâches). Chaque cellule est codée par le numéro de la tâche, la machine à laquelle l'opération est affectée, la date de début et de fin de l'opération.

3.2.3.2. Le codage dans les ateliers de type flow-shop

Plusieurs chercheurs se sont intéressés à la résolution des problèmes d'ordonnancement pour les ateliers de type flow-shop [Bou06a].

Dans son article intitulé «A Genetic Algorithm with Sub-indexed Partitioning genes and its application to production scheduling of parallel machines », [Jou05], propose un algorithme génétique dont le codage des gènes est basé sur une indexation dépendante de la génération (GASP). L'ordonnancement cherche la combinaison optimale entre le temps de fabrication le plus court et du temps d'attente minimum des machines. Puisque le problème en général est NP-difficile, les solutions d'ordonnancement obtenues sont pseudo optimales pour des machines du flow-shop parallèle. Cet algorithme a été appliqué pour un système de production dans le domaine électronique. Il a montré que les solutions de GASP sont meilleures que celles des règles heuristiques. Lee et al. [Lee02] considérèrent les jobs avec des contraintes de précédence dans l'ordonnancement. Leur fonction «fitness » inclus les paramètres : dates de fin des jobs et coût de l'approvisionnement.

Leu et al. [Leu02] proposent également une technique de recherche basée sur un algorithme génétique. Cette technique est adoptée dans un système de production du type flow-shop, pour fournir la combinaison optimale ou pseudo-optimale des séquences de production, en considérant les ressources utilisées, et en minimisant le makespan tout en respectant les contraintes de ressources et de la production.

3.2.4. Codage proposé pour le problème à une machine en industries agroalimentaires

3.2.4.1. Introduction

Dans les industries agroalimentaires, il faut tenir compte des produits manipulés car ils ont en général des durées de vie et des dates de mise en fabrication assez courtes. En effet, dans notre application nous avons considéré le critère péremption des produits qui est un critère important et spécifique à l'industrie agroalimentaire. Donc dans un atelier agroalimentaire, une opération est caractérisée aussi bien par sa date de début au plus tôt et sa date de fin au plus tard que par les dates de validités v_{ijk} des composants c_{ijk} nécessaires pour accomplir cette opération. D'autres critères, plus classiques, sont aussi pris en compte.

3.2.4.2. Optimisation en industries agroalimentaires - Position du problème

Il s'agit d'optimiser un ordonnancement multicritère appliqué à l'industrie agroalimentaire. Deux critères considérés sont spécifiques à ce type d'industrie, on distingue la périssabilité des produits et le discount de distribution et un troisième critère qui est plus « classique » à savoir la date de fin de l'ordonnancement. L'objectif est alors de sélectionner parmi l'ensemble des ordonnancements réalisables celui qui présente le meilleur compromis entre les différents critères en réduisant et en filtrant l'espace de recherche initial. La décision d'ordonnancer, en respectant les contraintes du problème, une opération *i* avant une opération *j* a pour objectif de minimiser : les coûts engendrés par la péremption de certains composants, le coût du discount de distribution ainsi que la durée totale de fabrication du produit fini.

Le problème est alors de déterminer le meilleur ordonnancement c'est à dire celui dont la fonction objectif est minimale. Cette fonction objectif $f(.)$ ou « fitness » est définie comme suit :

$$f(.) = \min \sum_i a_i C_i \qquad 3.1$$

avec :

- C_i : $i^{ème}$ critère
- $a_i \in [0,1]$: coefficient d'importance du critère C_i, $\sum_i a_i = 1$

3.2.4.3. Formulation des critères

Dans l'application considérée, optimisation en atelier de production agroalimentaire, les critères suivants sont retenus :

- C_1 : le coût des produits périmés ;
- C_2 : le coût du discount de distribution ;
- C_3 : la date de fin de l'ordonnancement (le $C_{\max}$).

avec :

$$C_1 = \sum_i \sum_j \sum_k P_{ijk}^{rev} \left(\frac{\max(0, t_{ij} - v_{ijk})}{(t_{ij} - v_{ijk})} \right) \qquad 3.2$$

$$C_2 = \sum_i \max\left(0, d_{P_i}^{liv} - C_{P_i}\right) \times \left(\frac{P_{P_i}^{ven}}{Dv_{P_i} - Dr_{P_i}} + C_{P_i}^{stk} \right) \qquad 3.3$$

$$C_3 = \max_{1 \le i \le n} \left(C_{P_i} \right) \qquad 3.4$$

L'objectif, pour ce cas d'application est de construire un ordonnancement qui satisfait ces critères tout en respectant les contraintes du problème qui sont les suivantes :

- la contrainte de précédence, une opération *i* devant être exécuter avant une opération *j* ;
- la contrainte de gamme, représente l'ordre de passage de l'opération dans la gamme ;
- la contrainte de disponibilité de l'opération à ordonnancer selon sa date de début au plus tôt.

3.2.4.4. Formulation des bornes inférieures

Il s'agit de calculer la borne inférieure C_i^b du critère C_i pour i = {1,2} des critères retenus dans cette application.

- ***Proposition 1***

$$C_i \geq 0 \quad \forall i \in \{1,2\} \text{ et } C_i^b = 0 \tag{3.5}$$

- ***Proposition 2***

La borne inférieure du makespan, C_3^b est définie comme suit :

$$C_3^b = \max_i \left(t_i + p_i\right)$$

En effet, on a :

$$C_3 = \max_{1 \leq i \leq n} \left(C_{P_i}\right), \text{ et } t_i \geq r_i \tag{3.6}$$

alors :

$$C_3^b = \max_i \left(t_i + p_i\right) \tag{3.7}$$

3.2.4.5. Codage proposé : CLOO-A pour optimiser les critères considérés

Le codage proposé [Tan06a] dans cette application est le Codage Liste des Opérations Ordonnées appliqué en Agroalimentaire : CLOO-A, Tableau 3.1. Inspiré du codage CLO (Codage Liste des Opérations) [Kac03] et du codage CPM (Codage Parallèle Machines) [Mes99], il consiste à proposer des listes ordonnées d'une gamme de produits en ligne de fabrication.

Le codage proposé défini l'ordre, la date effective de début et la date de fin effective des opérations pour chaque gamme de produit. Ces dates sont calculées et mises à jour par l'algorithme de calcul des dates, Tableau 3.2.

Tab.3.1. Codage CLOO-A

1	2	3	4
O_{k2}, t_{k2}, g_{k2}	O_{k1}, t_{k1}, g_{k1}	O_{k4}, t_{k4}, g_{k4}	....

Tab.3.2. Table de calcul des dates

t_{ij} , date effective de début de l'opération O_{ij} , $t_{ij} = \max\left(r_{ij}, g_{ik}\right)$
g_{ij} , date de fin effective de l'opération O_{ij} , $g_{ij} = \max\left(r_{ij}, g_{ik}\right) + p_{ij}$
où g_{ik} représente la date de fin de l'opération O_{ik}
qui précède O_{ij}

3.2.4.6. Opérateurs utilisés dans l'algorithme proposé

Les opérateurs utilisés pour le codage CLOO-A sont l'opérateur de croisement à un point, l'opérateur de croisement à deux points et l'opérateur de mutation.

- *Opérateur de croisement à un point*

Cet opérateur choisit deux individus (parents) pour générer deux autres individus enfants, à partir d'un seul point de croisement choisi aléatoirement, figure 3.2.

Début

Étape 1 : Choisir deux individus (parents) et un point de croisement *i*

Étape 2 : *Si* $p \geq p_c$ *alors*

Parcourir toutes les opérations

Tant que ($j < n$) *faire*

Si ($j < i$) *Alors*

Copier les opérations du parent1 dans l'enfant1

Copier les opérations du parent2 dans l'enfant2

Fin Si

Copier du parent2, à la même position, les opérations non existantes dans l'enfant1

Copier du parent1, à la même position, les opérations non existantes dans l'enfant2

Fin Tant que

Fin Si

Étape 3 : Terminer la construction de l'enfant1 (respectivement de l'enfant2) avec les opérations manquantes (en respectant l'ordre)

Étape 4 : Mettre à jour l'enfant1 et l'enfant2

Étape 5 : Calculer C_1, C_2 et C_3 des deux nouveaux individus.

Fin

Fig.3.2. Algorithme de croisement à un point

Cet algorithme utilise les notations suivantes :

- i : indice de l'opération (point de croisement) ;
- j : indice pour parcourir les opérations ;
- n : nombre total des opérations ;
- p_c : probabilité de croisement.

▪ Exemple

Considérons un problème à une machine et à sept opérations pour réaliser un produit P_1. Un exemple de croisement à un point est représenté dans la figure 3.3.

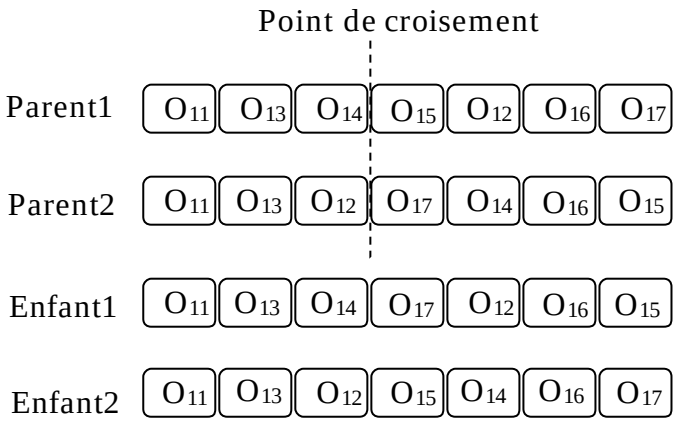


Fig. 3.3. Exemple de croisement à un point

- *Opérateur de croisement à deux points*

Cet opérateur, choisit deux individus parents pour générer deux autres individus enfants à partir de deux points de croisement choisis aléatoirement, figure.3.4.

Début

Étape 1 : Choisir deux individus (parents) et deux points de croisement *i* et *k*

Étape 2 : *Si* $p_i \geq p_c$ et $p_k \geq p_c$ *alors*

Parcourir toutes les opérations

Tant que $(j < n)$ *faire*

Copier les opérations du parent1, qui précèdent le premier point de croisement et qui suivent le second point de croisement, dans l'enfant1

Copier les opérations du parent2, qui précèdent le premier point de croisement et qui suivent le second point de croisement, dans l'enfant2

Copier, à la même position, les opérations non existantes du parent2 dans l'enfant1

Copier, à la même position, les opérations non existantes du parent1 dans l'enfant2

Fin Tant que

Fin Si

Étape 3 : Terminer la construction de l'enfant1 (respectivement de l'enfant2) avec les opérations manquantes (en respectant l'ordre)

Étape 4 : Mettre à jour l'enfant1 et l'enfant2

Étape 5 : Calculer C_1, C_2 et C_3 des deux nouveaux individus.

Fin

Fig.3.4. Algorithme de croisement à deux points

- Exemple

Considérons le même exemple que précédemment, mais dans ce cas il y a deux points de croisement *i* et *k,* figure 3.5.

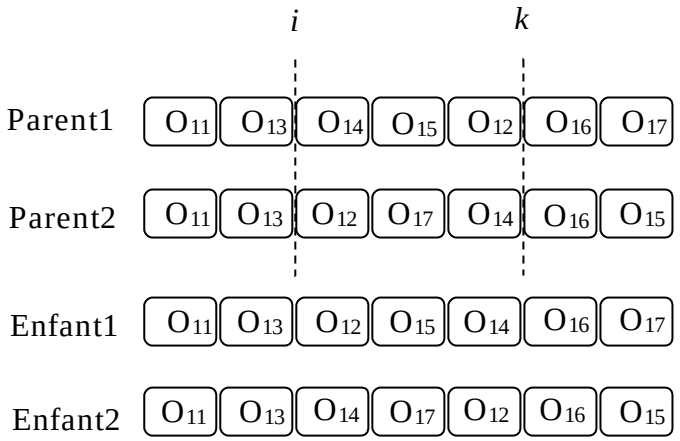


Fig.3.5. Exemple de croisement à deux points

- *Opérateur de mutation*

Cet opérateur choisit aléatoirement deux points d'un même individu (parent), pour générer un autre individu (enfant), figure 3.6.

Début

Étape 1 : Choisir deux positions n_1 et n_2 d'un même individu, à chaque position correspond une opération O_i et une opération O_j (*i* et *j* indices de l'opération) et choisir une valeur p_m

Étape 2 : Si $p \leq p_m$ alors

Permuter entre l'opération O_i et l'opération O_j

Étape 3 : Mettre à jour l'enfant

Étape 4 : Calculer C_1, C_2 et C_3 du nouvel individu

(p : probabilité de l'individu choisi, p_m : probabilité de mutation)

Fin

Fig.3.6. Algorithme de mutation

- Exemple

Dans cet exemple, une mutation est réalisée pour le « parent » par permutation des deux opérations O_{14} et O_{12} pour obtenir un « enfant », figure 3.7.

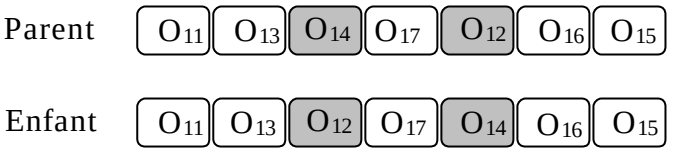


Fig. 3.7. Exemple de mutation

3.2.4.7. Approche d'évaluation multicritère

D'une manière générale, les critères considérés présentent entre eux des relations non linéaires et complexes et n'ont pas forcement la même importance du point de vue de décideur. Ainsi, beaucoup de considérations peuvent être retenues pour tenir compte de toutes ces difficultés. Pour ce faire, une méthode d'évaluation floue est proposée. Cette dernière est basée sur les étapes qui suivent

- Pour chaque fonction objectif, une borne inférieure est calculée, telle que :

$$C_i(x) \geq C_i^b \quad \forall x \in S, \quad 1 \leq i \leq n_c \tag{3.8}$$

où S représente l'espace des solutions réalisables et n_c le nombre de fonctions « objectif ».

- Les valeurs des fonctions «objectif », dans la plupart des cas, peuvent appartenir à différents intervalles d'amplitude variable. En particulier, soit H une heuristique choisie et C_i^h la valeur maximale de la solution donnée par l'heuristique considérée selon la $i^{ème}$ fonction objectif.

La fuzzification est appliquée en utilisant les fonctions d'appartenance décrites sur la figure 3.5.

Un vecteur $C(x)$ est associé à chaque solution réalisable x,

$C(x) \in \left[C_1^b, +\infty\right] \times \ldots \times \left[C_{n_c}^b, +\infty\right]$, avec $C(x) = (C_1(x), \ldots, C_{n_c}(x))^T$. Pour chaque vecteur$C(x)$, une fuzzification de ses composantes $C_i(x)$, proposée selon leurs positions dans les intervalles$\left[C_i^b, C_i^h\right]$, est considérée en deux sous-ensembles flous B^i et M^i, figure 3.8. On a :

$$m_i^B(C_i(x)) = \frac{C_i^h - C_i(x) + e}{C_i^h - C_i^b + e}, \text{ si } C_i(x) \in \left[C_i^b, C_i^h + e\right] \text{ et } m_i^B(C_i(x)) = 0, \text{ sinon} \qquad 3.9$$

De la même façon , il vient :

$$m_i^H(C_i(x)) = \frac{C_i(x) - C_i^h + e}{C_i^b - C_i^h + e}, \text{ si } C_i(x) \in \left[C_i^b, C_i^h + e\right] \text{ et } m_i^H(C_i(x)) = 0, \text{ sinon}$$

avec :

- $m_i^B(C_i(x))$ étant la mesure floue de $C_i(x)$ dans le sous-ensemble B^i ;
- $m_i^H(C_i(x))$ étant la mesure floue de $C_i(x)$ dans le sous-ensemble H^i.

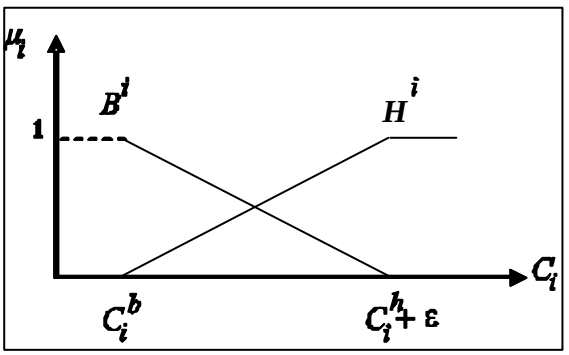


Fig. 3.8. Fuzzification floue dans la résolution du problème d'échelle

Ensuite, la qualité de chaque solution x est caractérisée par le vecteur $C_B(x)$ défini dans (3.10) dont toutes les composantes sont homogènes puisqu'elles appartiennent toutes au même intervalle $[0,1]$ et sont toutes sans dimension :

$$C_B(x) = (a_1, ..., a_{n_c})^T \quad 3.10$$

$$a_i = m_i^B(C_i(x)), \quad \forall i = 1,2,...,n_c \quad 3.11$$

- Pour l'évaluation multicritère, la fonction objectif $C_g(x)$ est réduite à la minimisation de la somme pondérée des critères relative à l'utilisation de l'opérateur d'agrégation OWA [Yag88] :

$$C_g(x) = \sum_{i=1}^{n_c} w_i a_i \quad 3.12$$

Pour aider le décideur quand il ne peut pas clairement donner une préférence particulière à des fonctions objectif, un ensemble de solutions Pareto-optimales est construit sans accorder de privilège à une direction particulière de recherche. Cette approche est basée sur un algorithme dans lequel, la fonction objectif $C_g(.)$, définie dans la relation (3.12), est utilisée pour l'évaluation des solutions. Les pondérations w_i $(1 \le i \le n_c)$ sont calculées en utilisant une règle floue. L'idée est de mesurer la qualité moyenne des solutions selon chaque critère à chaque itération et de calculer les différents poids suivant le degré de cette qualité. Le but est d'étudier les gains et les améliorations possibles des solutions en accordant la priorité à l'optimisation des fonctions objectif dont la moyenne des valeurs est loin de la borne inférieure ; cette approche est appelée approche agrégative avec direction de recherche dynamique.

Soit $\overline{C_i^k}$ la moyenne des solutions de la $i^{ème}$ fonction objectif trouvée à la $k^{ième}$ itération :

$$\bar{C}_i^k = \frac{\sum_{x \in P_k} C_i^k(x)}{card(P_k)} \qquad 3.13$$

P_k étant la population des solutions à cette itération.

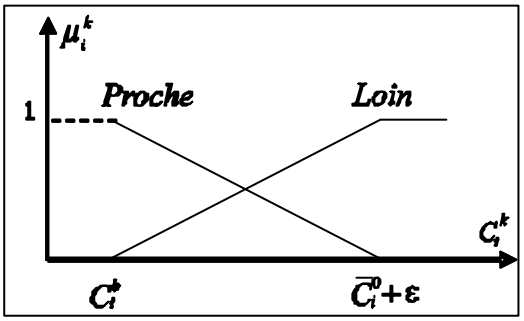


Fig. 3.9**.** Fonction d'appartenance des différentes valeurs des critères

Pour chaque vecteur $C(x)$, une fuzziffication est appliquée sur ces composantes $C_i(x)$ selon leurs positions dans les intervalles $\left[C_i^b, \bar{C}_i^0 + e\right]$ où e est une petite valeur positive introduite pour éviter le problème de la division par zéro ($e = 0.1C_i^b$, si $\bar{C}_i^0 = C_i^b$; $e = 0$ sinon).

L'évaluation de la qualité des solutions se fait en utilisant les fonctions d'appartenance définies dans la figure 3.9., relatives aux deux sous ensembles flous, *Proche* et *Loin* de la borne inférieure.

Les fonctions d'appartenance peuvent ainsi être formulées comme suit :

$$\mu_{ik}^{L}\left(\bar{C}_i^k\right)=\frac{\bar{C}_i^k-C_i^b}{\bar{C}_i^0-C_i^b+\varepsilon}, \text{ si } \bar{C}_i^k\in\left[C_i^b,\ \bar{C}_i^0+\varepsilon\right] \qquad 3.14$$

$$\mu_{ik}^{L}\left(\bar{C}_i^k\right)=1, \text{ sinon}$$

Le calcul des différentes pondérations w_i^{k+1} est effectué en utilisant les deux règles floues suivantes :

- Si (C_i^k est *Proche* de C_i^b) Alors (w_i^{k+1} diminue)
- Si (C_i^k est *Loin* de C_i^b) Alors (w_i^{k+1} augmente)

Ce qui conduisent à l'expression de w_i^k suivante :

$$w_i^k=\frac{\mu_{ik}^{L}\left(\bar{C}_i^k\right)}{\sum_{j=1}^{n_c}\mu_{jk}^{L}\left(\bar{C}_j^k\right)},\ \forall i\ \forall k \text{ tq } 1\le i\le n_c \text{ et } 2\le k\le Q \qquad 3.15$$

w_i^1 correspond à la première itération définie par :

$w_i^1=\frac{1}{n_c}, \forall i$ tel que $1\le i\le n_c$

Q est le nombre total d'itérations et L l'indice relatif au sous-ensemble flou *Loin*.

Les différents vecteurs de pondération $(w^1, w^2, ..., w^Q)$ sont calculés progressivement de la $k^{ème}$ génération P_k à la génération P_{k+1}, selon la distance entre les bornes inférieures et la moyenne des individus de la $k^{ème}$ génération, représentée par un disque noir dans la figure 3.10.

Le but est d'améliorer des solutions en accordant la priorité à l'optimisation des fonctions objectif dont la moyenne des valeurs est loin de la borne inférieure. Il s'en suit qu'en utilisant une règle floue, il est envisageable de contrôler la direction de recherche afin de construire un ensemble final avec des solutions s'approchant le plus possible des valeurs optimales, figure 3.7.

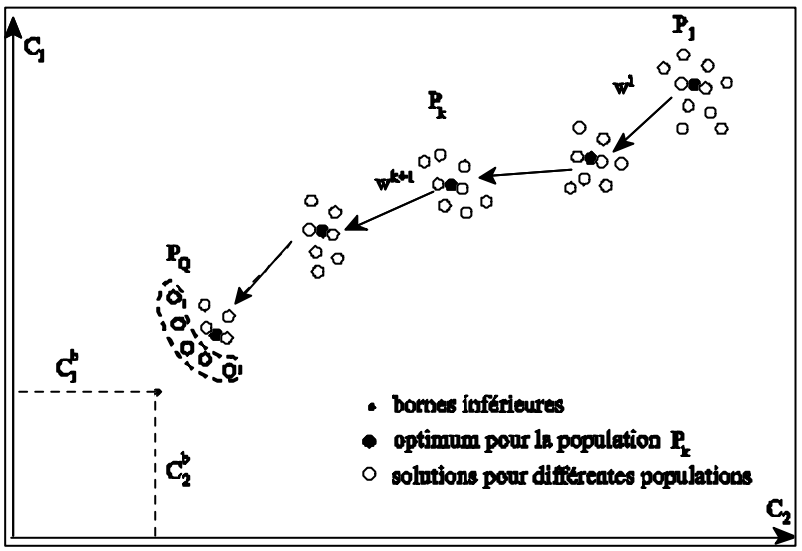


Fig. 3.10. Direction de recherche

Cette méthode, généralement utilisée quand le décideur ne peut pas donner une préférence particulière à une fonction objectif, permet aussi de générer des poids des critères différents d'une itération à une autre de manière dynamique en fonction de la valeur moyenne des solutions.

3.2.4.8. Résultats de simulation pour différentes gammes de produits agroalimentaires

Pour illustrer l'efficacité et la performance de l'approche proposée, six exemples représentatifs basés sur des données pratiques, sont sélectionnés pour la simulation. Ces exemples forment des gammes de produits de cinq opérations à dix opérations. L'approche proposée est ainsi appliquée pour optimiser les trois critères cités dans les relations (3.2, 3.3 et 3.4).

A titre d'exemple, les données relatives de l'exemple du produit P_1 comprenant dix opérations sont données dans le tableau 3.3.

Par application de l'approche proposée, les résultats expérimentaux suivants sont obtenus, tableau 3.4.

Tab.3.3. Données relatives aux 10 opérations (P_1)

	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}	O_{16}	O_{17}	O_{18}	O_{19}	O_{110}
r_{1j}	0	1	2	3	4	1	3	2	1	3
p_{1j}	1	2	4	2	1	2	1	3	2	4
v_{1j1}	13	14	13	13	12	7	7	13	9	9
v_{1j2}	15	14	5	14	13	15	15	16	15	15
v_{1j3}	-	12	13	12	11	14	14	12	14	14
P^{rev}_{1j1}	2	3	4	3	2	1	1	2	2	2
P^{rev}_{1j2}	1	2	2	4	3	2	2	1	4	1
P^{rev}_{1j3}	-	4	3	2	2	1	3	2	3	3
Dv_{P_1}	35	35	35	33	33	33	33	31	31	31
Dr_{P_1}	10	10	10	9	9	9	9	11	11	11
$d^{liv}_{P_1}$	22	22	22	20	20	20	20	24	24	24
$C^{stk}_{P_1}$	3	3	3	4	4	4	4	2	2	2
$P^{ven}_{P_1}$	6	6	6	8	8	8	8	5	5	5

Tab.3.4. Résultats expérimentaux

n	Meilleurs Ordonnancements	C_1	C_2	C_3	$C_g(.)$
10 (P_1)	$O_1O_3O_5O_9O_4O_2O_7O_6O_8O_{10}$	14	4	24	0,915
9 (P_2)	$O_1O_2O_3O_4O_5O_7O_6O_8O_9$	9	4	20	0,95
8 (P_3)	$O_1O_7O_5O_2O_3O_8O_4O_6$	14	1	18	0,963
7 (P_4)	$O_1O_3O_6O_7O_4O_2O_5$	12	9	17	0,752
6 (P_5)	$O_1O_4O_3O_5O_2O_6$	4	10	14	0,977
5 (P_6)	$O_1O_4O_5O_3O_2$	12	10	14	0,53

Les résultats numériques obtenus montrent que, pour les gammes des produits P_1, P_2, P_3 et P_5, ces solutions sont très satisfaisantes car $C_g(.)$ est proche de 1. Par contre, pour les autres produits, les solutions trouvées sont moins satisfaisantes dus au critère C_2 qui représente le coût du discount de distribution.

3.2.4.9. Conclusion

Les résultats obtenus montrent que les solutions calculées sont, généralement, acceptables et satisfaisantes. Les valeurs des différentes fonctions « objectif » montrent l'efficacité de l'approche proposée (tableau 3.4). De plus, la méthode proposée, a permis d'obtenir de bons résultats dans un temps de calcul polynomial. En fait, les diverses valeurs des critères indiqués par la méthode d'optimisation multiobjectif par Pareto-optimalité montrent son efficacité. Les valeurs des critères pour la frontière de Pareto sont à proximité des bornes inférieures. En effet, une telle approche permet de produire des solutions Pareto-optimales de bonne qualité.

3.3. Optimisation par les algorithmes de colonie de fourmis

3.3.1. Le contexte existant

Les algorithmes d'Optimisation par Colonie de Fourmis (OCF, ou Ant Colony Optimization ACO) font partie de la classe des métaheuristiques. L'objectif principal de cette approche est de trouver le chemin le plus court entre un nid et une source de nourriture. C'est en 1992 que Marco Dorigo de L'Université Libre de Bruxelles [Dor92] a formulé cette méthode. A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le Problème du Voyageur de Commerce (PVC) qui a pour objectif de trouver la tournée la plus courte pour un certain nombre de villes. L'avantage de cette méthode réside dans les stratégies de recherche qui réalisent un compromis entre l'exploitation et l'exploration. L'idée initiale provient de l'observation des fourmis. Celles-ci ont la capacité de trouver le chemin le plus court entre leur nid et une source de nourriture en contournant les obstacles qui jonchent leur chemin. Le premier algorithme de ce type a été conçu pour le problème du voyageur de commerce, figure 3.11. D'autres algorithmes ont été développés pour résoudre d'autres problèmes combinatoires tels que le problème de l'affectation quadratique [Stü00], les problèmes de routage de véhicule [Bul99], le problème du sac à dos multidimensionnel [Ala04], les problèmes d'ordonnancement [Shy04] [Lio07], etc.

3.3.2. Les algorithmes de colonie de fourmis : principe général

Les algorithmes de colonie de fourmis reposent sur une analogie avec des phénomènes naturels et s'appuient sur le comportement collectif des fourmis pour organiser la recherche de nourriture. Les fourmis explorent leur environnement en laissant derrière elles des traces volatiles, appelées traces de *phéromone*. Elles se servent de ces traces pour se guider et tendent naturellement à les suivre.

En l'absence de traces de phéromone, leur exploration devient complètement aléatoire.

Ces fourmis possèdent donc une vision très limitée de leur environnement. Une fois la nourriture trouvée, elles se servent des traces qu'elles viennent de déposer pour retrouver le chemin du retour vers le nid. Durant le trajet, elles laissent de nouveau sur leur passage des traces, en quantité proportionnelle à l'intérêt de la source de nourriture. Les fourmis ayant choisi la route la plus courte effectuant un plus grand nombre d'allers-retours, la quantité de phéromone déposée y sera plus importante et cette route deviendra éventuellement empruntée par toutes les fourmis. Le chemin, est donc fortement imprégné de phéromones et constitue une piste de choix pour la colonie de fourmis. Plus la source de nourriture a été jugée intéressante, plus les fourmis auront tendance à suivre ce chemin. Ainsi, peu à peu, les traces vers les sources de nourriture sont de plus en plus marquées. Goss et al. [Gos89] ont observé qu'en présence de deux routes possibles vers la nourriture, les fourmis adoptent rapidement la plus courte. En effet, la bonne route se trouve favorisée par une plus grande concentration de trace de phéromone. Le biais est ensuite renforcé par le phénomène d'évaporation des phéromones, qui tend à effacer progressivement les traces les moins fréquentées. Colorni et al. [Col92] ont adapté ce principe au domaine de l'optimisation combinatoire. Pour ce faire, ils ont associé le voisinage du nid à l'espace des solutions. Chaque solution s'apparente à une source de nourriture dont la qualité est fournie par la fonction d'évaluation. Chaque fourmi est assimilée à un processus répétitif de construction de solutions. La construction est biaisée par un ensemble de données globales figurant les traces de phéromone. Cet ensemble est une mémoire sur la qualité des solutions, régulièrement mise à jour par les processus de construction et par un mécanisme simulant l'évaporation de la phéromone.

- Créer un ensemble d'agents (fourmis) qui coopèrent par l'intermédiaire de traces de phéromone pour trouver le chemin le plus court du PVC.
- Les traces de phéromone représentent la quantité de phéromone déposée par les fourmis sur le trajet reliant la ville *i* à la ville *j*.
- Le comportement des fourmis est modélisé ainsi :
  - La fourmi se place initialement sur une ville choisie au hasard.
  - Elle choisie la ville suivante parmi les villes non encore visitées en suivant le chemin le plus marqué par les traces de phéromone.
  - Elle dépose une quantité de phéromone sur le chemin qu'elle a emprunté (règle locale de mise à jour des traces). L'évaporation de la phéromone est prise en compte (pour ne pas tomber dans des solutions sous optimales) ce qui donne une loi qui a la forme suivante :

$$Pheromone[i, j] = (1 - \alpha)\, Pheromone[i, j] + \alpha * K$$

K étant une constante. La valeur de K recommandée par Dorigo et Gambardella est de $1 / (N * L_{mv})$, avec N le nombre de villes et L_{mv} la longueur du trajet calculée par la méthode du meilleur voisin ou par une approximation.

Fig. 3.11. Principe général de l'algorithme du PVC

3.3.3. Représentation du problème d'optimisation

Le problème considéré est un problème d'ordonnancement d'un ensemble d'opérations pour la fabrication de certains produits sur une machine unique. Le problème traité dans ce travail est représenté par un ensemble de solutions et une fonction « *objectif* », qui représente la fonction de coût à minimiser. Cette fonction prend une valeur à chaque solution et un ensemble de contraintes est considéré. L'objectif est de trouver l'optimum global en respectant les contraintes du problème. Les différents états du problème sont représentés par une séquence d'opérations. Dans cette représentation, les fourmis construisent des solutions en se déplaçant dans un graphe *G= (N, L),* où les nœuds représentent les composants de *N* et où l'ensemble *L* définit les arcs qui lient les composants de *N*. Les contraintes du problème sont directement implémentées dans les règles de déplacement des fourmis. Elles sont mises en œuvre en empêchant les fourmis de prendre les chemins qui violent ces contraintes.

3.3.4. Algorithme d'optimisation par colonie de fourmis proposé

3.3.4.1. Présentation

Dans le problème d'ordonnancement dans les industries agroalimentaires, considéré, l'objectif est de déterminer le meilleur ordonnancement d'un ensemble *E* de *n* opérations candidates à l'ordonnancement caractérisées par leurs dates de début au plus tôt de mise en fabrication, r_i, leurs dates de fin fabrication au plus tard, d_i et leurs durées opératoires p_i. La formulation du problème est inspirée du celle du Problème du Voyageur de Commerce (PVC)**.** Chaque opération à choisir pour l'ordonnancement est représentée par un nœud dans un graphe $G=(N,L)$, où *N* représente le nombre des nœuds ($N = n$) et *L* le nombre d'arcs qui lient les opérations. Quand une fourmi se déplace d'un nœud *i* à un nœud *j* elle laisse une trace de phéromone sur l'arc (a_{ij}).

Cette trace enregistre l'information sur l'utilisation de l'arc (a_{ij}) de sorte que plus cette utilisation a été importante dans le passé, plus la probabilité que cet arc soit utilisé à nouveau est élevée. Pour le problème d'ordonnancement, l'information contenue dans la trace de phéromone est basée sur les données (la date de validité des composants formant l'opération entre autres) et les contraintes du problème. Suivant ces informations heuristiques, la fourmi choisit la prochaine opération *i* à mettre en œuvre. En effet, au temps *t*, à partir d'une séquence partielle d'ordonnancement déjà construite, chaque fourmi *k* choisit la prochaine opération à ajouter à la séquence en utilisant une règle probabiliste basée sur un compromis entre h_{ij} et l'intensité de la trace de phéromone $t_{ij}(t)$. Les coefficients *a et b* sont des paramètres qui permettent de contrôler l'importance ces deux éléments.

Cette probabilité, pour la fourmi *k* de choisir l'arc (*ij*), notée $P_{ij}^{k}(t)$, est calculée par l'expression (3.16) suivante:

$$P_{ij}^{k}(t) = \frac{\left[t_{ij}(t)\right]^{a} * \left[h_{ij}\right]^{b}}{\sum_{l \notin tabou_k} \left[t_{il}(t)\right]^{a} * \left[h_{il}\right]^{b}} \qquad 3.16$$

Dans le problème du PVC, la visibilité h_{ij} est définie par la matrice de distance qui prend en considération la distance entre les paires de villes *i* et *j*, *D* = ($1/d_{ij}$). Pour le problème d'ordonnancement considéré, la matrice correspondante contient, pour chaque paire d'opérations, de l'information sur chacun des objectifs à minimiser de façon à guider la recherche (3. 17).

À l'initialisation de l'algorithme, l'intensité de la trace de phéromone pour toutes les paires d'opérations (*i,j*) est fixée à une petite valeur positive t_0. Les paramètres a, b et g sont utilisés afin de déterminer l'importance relative de l'intensité de la trace et des objectifs dans la construction d'une solution.

De plus, une liste taboue est maintenue pour garantir qu'une opération ayant déjà été affectée à la séquence en cours de construction ne soit pas sélectionnée une autre fois. Chaque fourmi *k* va donc posséder sa propre liste taboue, « $tabou_k$ », qui gardera en mémoire les opérations déjà sélectionnées.

$$\begin{cases} P_{ij}^{k}(t) = \dfrac{\left[t_{ij}(t)\right]^{a} * \left[f_{ij}\right]^{b} * \left[y_{ij}\right]^{g}}{\sum_{h \notin tabou_k} \left[t_{ih}(t)\right]^{a} * \left[f_{ih}\right]^{b} * \left[y_{ih}\right]^{g}} & si \quad j \in \Omega \\ 0 & sinon \end{cases} \qquad 3.17$$

Ω étant l'ensemble des opérations disponibles, $f_{ij} = \dfrac{1}{v_{ijk}}$ et $y_{ij} = p_{ij}$, v_{ijk} et p_{ij} représentant respectivement la date de validité du composant c_{ijk} relatif à l'opération *i* et la durée d'exécution (normalisées), qui ont une influence directe sur la fonction objectif.

Durant une itération de l'algorithme, plusieurs fourmis construisent, à tour de rôle, une solution c'est-à-dire une séquence d'opérations. A la fin de chaque cycle, chacune des *m* fourmis laisse une certaine quantité de phéromone $\Delta t_{ij}^{k}(t)$ définie dans (3.18) sur son trajet, cette quantité dépend de la qualité de la solution trouvée.

Soit $C_k^{tot}(t)$ l'évaluation de l'objectif à optimiser de la solution trouvée par la $k^{ème}$ fourmi.

La contribution sur la mise à jour de la trace de la fourmi k est alors calculée de la façon suivante :

$$\Delta t_{ij}^{k}(t) = C_{\min}^{tot} / C_k^{tot}(t) \qquad 3.18$$

où $C_k^{tot}(t)$ est le coût total associé à la fourmi k (S_k) quand elle a effectué son trajet à l'instant t et $C_{\min}^{tot}$ représente le coût minimum obtenu après que les m fourmis aient effectué leurs trajets.

L'algorithme effectue une mise à jour globale de l'intensité de la trace de phéromone t_{ij}^{k} à la fin de chaque cycle pour éviter une convergence prématurée de l'algorithme (3.19). Cette mise à jour est influencée par un facteur d'évaporation r , $0 < r \leq 1$, qui diminue la quantité de phéromone présente sur tous les arcs (ij) [Gra00], [Dor00], [Col91]. Cette évaporation ne doit pas intervenir sur les tests, c'est-à-dire les solutions de la liste tabou.

$$t_{ij}^{k} = r * t_{ij}^{k} + \sum_{l=1}^{m} \Delta t_{ij}^{l} \qquad 3.19$$

3.3.4.2. Algorithme d'optimisation par colonie de fourmis

La figure 3.12 présente l'algorithme proposé pour l'optimisation de la fonction de coût qui représente la somme du coût des produits périmés et du coût du discount de distribution.

1. Initialisation

$N_C = 0$; N_{Cmax}

Initialiser la matrice de phéromone $t_{ij}^{k}(0) = t_0$

Initialiser la liste taboue

2. Réalisation d'un cycle

Tant que ($N_C < N_{Cmax}$) **faire**

Pour chaque opération de $\{1...n\}$

Pour chaque fourmi k dans $\{1...m\}$ **faire**

Choisir aléatoirement une opération i

Mettre à jour l'ensemble des opérations disponibles

Ajouter l'opération i à la séquence d'opérations S_k

Insérer la solution S_k dans la liste taboue

Sélectionner l'opération suivante selon P_{ij}^{k} (3.17)

Mettre à jour la matrice Δt_{ij}^{k} (3.18) selon la solution S_k

3. Mise à jour globale de la trace de phéromone

Évaluer la solution S_k suivant la fonction objectif

Mettre à jour la matrice t_{ij}^{k} (3.19)

Fin pour

Fin Tant que

$N_C = N_C + 1$

Fig. 3.12. Algorithme d'optimisation par colonie de fourmis

3.3.4.3. Application à l'ordonnancement d'un atelier de production agroalimentaire

Considérons un ensemble d'opérations à ordonnancer dans un atelier de production agroalimentaire. Ces opérations sont caractérisées par des composants qui ont des dates de validité assez courtes. L'objectif est alors de minimiser la péremption de certains de ces composants en assurant leur mise en fabrication avant leurs dates de péremption, tout en respectant certaines contraintes : la contrainte de précédence la contrainte de disponibilité et la contrainte de gamme. La complexité du problème s'accroît en présence d'autres contraintes spécifiques aux industries agroalimentaires.

La fonction de coût à optimiser est la somme de deux coûts à savoir :

- le coût des produits périmés, noté K_1,
- le coût de discount de distribution, noté K_2.

Par application de l'algorithme OCF proposé, nous avons obtenu les résultats suivants des figures 3.13 (a et b), relatifs à au comportement de la fonction de coût à optimiser par variation des différents paramètres, r , a , b *et* g , de l'algorithme.

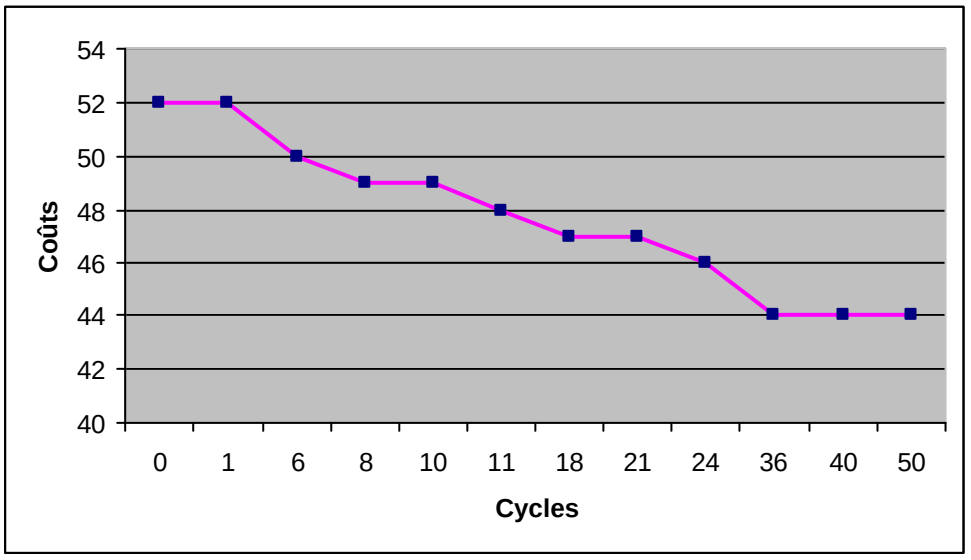


Fig. 3.13. a

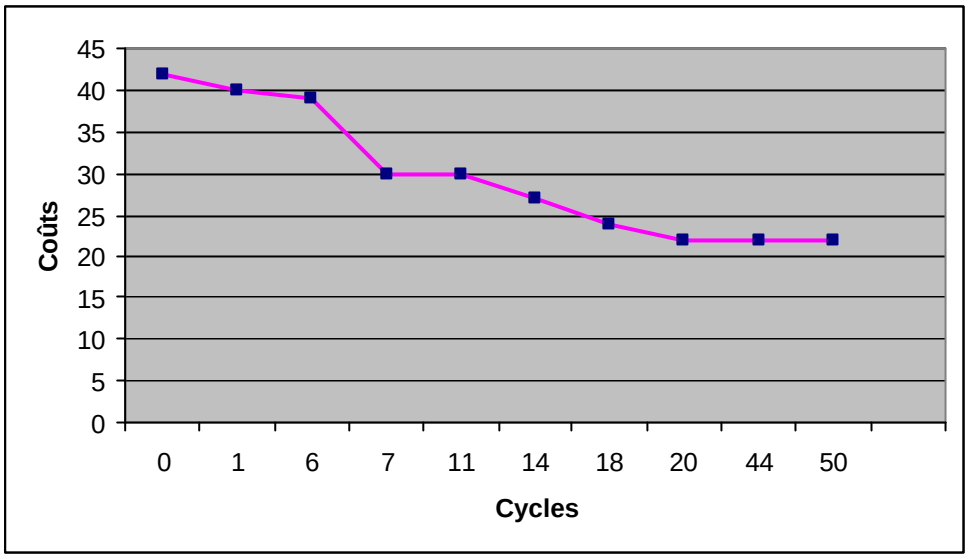


Fig. 3.13. b

Fig. 3.13. Variation du coût en fonction des cycles

Les deux graphiques, montrent que les résultats de la figure 3.13.(b) sont meilleurs puisque le coût optimal, atteint en un temps mois important que celui de la figure 3.13.(a), est plus intéressant. Il est à constater que les paramètres de l'algorithme d'optimisation par colonie de fourmis qui varient d'un problème à un autre suivant l'objectif de l'étude, influent sur les résultats. Il faut donc bien les choisir.

3.3.4.4. Conclusion

La métaheuristique d'optimisation par colonie de fourmis (OCF) présente la particularité d'utiliser une procédure de construction de solutions distincte de celle des autres métaheuristiques. Ces dernières utilisent l'exploration d'un voisinage ou l'échange d'informations entre les solutions. C'est un processus stochastique qui construit une solution, en ajoutant des composants aux solutions partielles. Néanmoins, la qualité des résultats fournis par cette métaheuristique dépend très fortement de l'importance accordée aux traces de phéromone. Lorsque la pondération est trop élevée, le mécanisme de construction tend à favoriser la génération des meilleures solutions trouvées dans les itérations précédentes.

À l'inverse, lorsqu'il est trop faible, il tend à produire des solutions au hasard. En effet, chaque fourmi construit une solution pour le problème à optimiser en choisissant le chemin le plus court entre son nid et la source de nourriture en suivant la trace de phéromone. La procédure de construction des solutions de l'OCF a ainsi exigé un ajustement particulier pour diriger la recherche dans la direction souhaitée et assurer son efficacité. En raison de sa simplicité, de sa flexibilité et de sa performance, cette méthode de résolution constitue cependant un outil d'aide à la décision pertinent dans un contexte d'application en industrie ou le compromis entre le temps de calcul et la qualité des solutions est important.

3.4. Conclusion

Dans ce chapitre, on s'est intéressé à l'optimisation multicritère afin de construire un ordonnancement dynamique appliqué à un atelier de production agroalimentaire pour minimiser la fonction de coût. Deux métaheuristiques ont été appliquées pour atteindre cet objectif. La première méthode concerne les algorithmes génétiques, hybridés avec l'approche Pareto-optimale ; cette hybridation augmente la probabilité de trouver un optimum de bonne qualité. La deuxième métaheuristique se base sur les algorithmes d'optimisation par colonie de fourmis. Cette dernière s'avère performante de point de vue temps de calcul et qualités de solutions par rapport à l'approche basée sur les algorithmes génétiques. Néanmoins, l'inconvénient de l'approche d'optimisation par colonie de fourmis réside dans le choix du paramétrage de l'algorithme correspondant. Cependant, la qualité des solutions trouvées dépend de ce choix.

Conclusion générale

Le travail développé dans cette thèse porte sur la problématique de l'ordonnancement pour le problème à une machine. Notre travail a été appliqué pour un atelier de production en industries agroalimentaires dont l'objectif est de trouver un ordonnancement dynamique adapté à ce type d'industrie. En effet, ce dernier présente la particularité des produits fabriqués et manipulés qui se caractérisent par des courtes durées de vie et la spécificité des critères à retenir : la minimisation des produits périmés à savoir les composants primaires, les produits semi-finis et les produits finis et la minimisation du discount de distribution tout en respectant la date de livraison ou au moins en réduisant la différence entre la date effective de fin de fabrication du produit fini et sa date de livraison. Un autre critère plus classique : la date de fin d'ordonnancement, le C_{max}, est par ailleurs considéré.

Pour atteindre ces objectifs, la méthode « branch and bound » a été appliquée, pour le filtrage de l'espace de recherche basé sur les critères retenus par le décideur. Il s'agit d'éliminer certaines opérations et de maintenir d'autres, en appliquant la technique d'« edge finding » et en se basant sur les règles de dominance des opérations permettant de mettre en fabrication les opérations prioritaires, la priorité étant évaluée par rapport aux critères retenus.
Cette approche exacte développée a permis de choisir, parmi plusieurs solutions possibles, celle qui représente l'ordonnancement dont la fonction coût représentée par la somme du coût des produits périmés et du coût du discount de distribution, est minimale.

Cette solution engendre un gain considérable vis-à-vis des critères considérés du fait de la particularité des produits fabriqués en industries agroalimentaires.
Ainsi, moyennant des règles de dominance des opérations et des paramètres nécessaires pour le calcul des coûts et des données de stock, peut être évitée la péremption de certains composants et produits semi-finis, en maintenant dans l'espace de recherche les opérations dont les composants possèdent les dates de validité limites les plus courtes. Au contraire, les opérations dont la durée de vie est suffisamment longue et dont la réalisation peut engendrer un retard de mise en fabrication d'autres opérations, sont éliminées de l'espace de recherche considéré.

Une deuxième approche basée sur l'utilisation des algorithmes génétiques, appartenant à la classe des métaheuristiques, a donné des résultats encourageants concernant l'optimisation des critères cités. En effet, les solutions obtenues sont, généralement, acceptables et satisfaisantes. Les valeurs des différentes fonctions « objectif » montrent l'efficacité de cette approche. De plus, cette méthode proposée a permis d'aboutir à des bons résultats. En fait, les diverses valeurs des critères indiqués par la méthode d'optimisation multiobjectif, par Pareto-optimalité, montrent son efficacité. Les valeurs des critères pour la frontière de Pareto sont à proximité des bornes inférieures donc bonne qualité.

Par ailleurs, la métaheuristique d'optimisation par colonie de fourmis présente la particularité d'utiliser une procédure de construction de solutions distincte de celle des autres métaheuristiques. Cette procédure utilise l'exploration d'un voisinage ou l'échange d'informations entre les solutions. C'est un processus stochastique qui construit une solution, en ajoutant des composants aux solutions partielles.

Néanmoins, la qualité des résultats fournis par cette approche dépend très fortement des paramètres de l'algorithme. La procédure de construction des solutions de l'algorithme d'optimisation par colonie de fourmis a ainsi exigé un ajustement particulier pour diriger la recherche dans la direction souhaitée et assurer son efficacité. En raison de sa simplicité, de sa flexibilité et de sa performance, cette méthode de résolution constitue cependant un outil d'aide à la décision pertinent dans un contexte d'application en industrie ou le compromis entre le temps de calcul et la qualité des solutions est important.

En perspectives, nous envisageons de :

- intégrer la gestion de la flotte de livraison pour maîtriser le discount de distribution,
- étudier la possibilité d'insertion d'opérations non initialement prévues, cas fréquent en industries agroalimentaires,
- prendre en considération d'autres contraintes et d'autres critères,
- appliquer d'autres méthodes pour le même problème,
- généraliser et étendre les approches proposées à d'autres industries qui présentent les mêmes types de contraintes telles que les industries chimiques ou pharmaceutiques.
- étudier et résoudre le problème de jop-shop ou le jop-shop flexible.

Bibliographie

[Alb93] Albers S., P. Brucker, « The complexity of one-machine batching problems », Discrete Applied Mathematics, Vol. 47, pp. 87-107, 1993.

[Ash04] Ashish G., D. Satchidananda, « Evolutionary Algorithms for Multi-Criterion Optimization: a Survey », International Journal of Computing & Information Sciences, Vol. 2, n° 1, pp. 38-57, 2004.

[Bal95] Balas E., J.K. Lenstra, A. Vazacopoulos, « The one machine problem with delayed precedence constraints and its use in job-shop scheduling », Management Science, Vol. 41, n° 1, pp. 94 -109, 1995.

[Bal85] Balas E., « On the facial structure of scheduling polyhedral », Mathematical Programming Studies, Vol. 24, 1985.

[Bar03] Barichard, V., « Approches hybrides pour les problèmes multi-objectifs », Thèse de Doctorat, Université d'Angers, 2003.

[Bar99] Barnier N., P. Brisset, « Optimisation par algorithme génétique sous contraintes », Technique et Science Informatique, Vol. 18, pp. 1-29, 1999.

[Bar85] Barker J.R., G.B. McMahon, « Scheduling the general job-shop», Management Science, Vol. 31, n°. 5, pp. 594–598, 1985.

[Bap98] Baptiste J.P., « Une étude théorique et expérimentale de la propagation des contraintes de ressources », Thèse de Doctorat, Université de Compiègne, 1998.

[Bea92] Beauquier D., J. Bersyel, P. Chrétienne, « Éléments d'algorithm-ique », Éditions Masson, Paris, 1992.

[Bén90] Bénassy J., « La gestion de production », Éditions Hermès, Paris, 1990.

[Bla96] Blazewicz J., K.H. Ecker, E. Pesch, G. Schmidt, J. Weglarz, « Scheduling Computer and Manufacturing Processes », Springer Berlin, 1996.

[Bou06a] Boukef H., F. Tangour, M. Benrejeb, P. Borne, « Nouveau codage pour la résolution de problèmes d'ordonnancement d'ateliers de type flow-shop par les algorithmes génétiques », STA'06, pp.1-14, 2006.

[Bou06b] Boukef H., F. Tangour, M. Benrejeb, P. Borne, « Sur la formulation d'un problème d'ordonnancement de type flow-shop d'ateliers de production en industries pharmaceutiques », JTEA'06, Hammamet 12-14 mai, 2006.

[Bra73] Bratley P., M. Floran, P. Robillard, « On sequencing with earliest starts and due dates with application to computing bounds for the $m \mid n \mid g \mid f_{max}$ problem », Naval Research Logistics Quarterly, Vol.20, n° 1, pp. 57-67, 1973.

[Bru98] Brucker P., S. Knust, « Complexity results for single-machine problems with positive finish-start time-lags », Osnabruecker Schriftenzur, Mathematik, Reihe P, n° 202, 1998.

[Bru97] Brucker P., J. Hurink, B. Jurish, B. Wöstman, «A branch and bound algorithm for the open-shop problem», Discrete Applied Mathematics, Vol. 76, pp. 43-59, 1997.

[Bru94] Brucker P., B. Jurisch, B. Sievers, « A branch & bound algorithm for the job-shop scheduling problem », Discrete Applied Mathematics, Vol. 49, pp. 107–127, 1994.

[Cab00] Cabarbaye A., « Outil générique d'optimisation par Algorithmes Génétiques et Simplexe », 8 èmes Journées Nationales du groupe Mode, Mathématique de l'Optimisation et de la Décision, de la SMAI, Toulouse, 23 - 25 mars, 2000.

[Car96] Carlier J., I. Rebaï, « Two exact methods for solving the permutation flow-shop problem », European Journal of Operational Research, pp. 238–251, 1996.

[Car94] Carlier J., E. Pinson, « Adjustment of heads and tails for the job-shop problem », European Journal of Operational Research, Vol. 78, pp. 146–161, 1994.

[Car89] Carlier J., E. Pinson, « An algorithm for solving the job-shop problem », Management Science, Vol. 35, n° 2, pp. 164–176, 1989.

[Car88] Carlier J., P. Chrétienne, « Problèmes d'ordonnancement, Modélisation, Complexité, Algorithmes », Éditions Masson, Paris, 1988.

[Car82] Carlier J., « The one-machine sequencing problem », European Journal of Operations Research, Vol.11, pp. 42–47, 1982.

[Car84a] Carlier J., « Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité », Thèse de Doctorat, Université Paris VI, 1984.

[Car84b] Carlier J., P. Chrétienne, C. Girault, « Modeling scheduling problems with timed Petri nets », Advances Studies in Petri Nets, Lecture Notes in Comp. Sci., September 1984, Spinger Verlag.

[Cav98] Cavalieri S., P. Gaiardelli, « Hybrid genetic algorithms for a multiple-objective scheduling problem», Journal of Intelligent Manufacturing, Vol. 9, pp.361-367, 1998.

[Cha00] Chauvet F., E. Levner, J.-M. Proth, « On-line part scheduling in a surface treatment system », Journal of Operational Research, IJOR, Vol. 120, n° 2, January, 2000.

[Cha96] Charon I., A. Germa, O. Hudry, « Méthodes d'optimisation combinatoire », Éditions Masson, Paris, 1996.

[Cha98a] Chauvet F. , J.-M. Proth « Scheduling heuristics for minimizing the makespan », 9th SIAM-Discrete Mathematics'98, Toronto, 1998.

[Cha98b] Chauvet F., J.-M. Proth, Y. Wardi, « On-line scheduling with wip regulation », Proceedings of the Rensselaer International Conference on Agile, Intelligent, and Computer-Integrated Manufacturing, Troy, New York, 1998.

[Che96] Cheng R., M. Gen, Y. Tsujimura, « A tutorial survey of job-shop scheduling problems using genetic algorithms », Computers Industrial Engineering, Vol. 30, n° 4, pp. 983-997, 1996.

[Chr95] Chrétienne P., E.G. Coffmann, J.K. Lenstra, Z. Liu, « Scheduling theory and its applications », Wiley, New York, 1995.

[Chr83] Chrétienne P., « Les réseaux de Pétri temporisés », Thèse de Doctorat, Université de Paris VI, Paris, 1983.

[Col02] Collette Y., P. Siarry, « Optimisation Multiobjectif », Éditions Eyrolles, Paris, 2002.

[Coo71] Cook S.A., « The complexity of theorem proving procedures », Association of Computing Machinery, Editor, Proceedings of the third annual ACM Symposium on the Theory of Computing, pp. 151-158, 1971.

[Dav85] Davis L., « Job shop scheduling with genetic algorithm», Proceedings of the first International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pp.136-140, 1985.

[Dec02] Dechter R., D. Frost, « Buckjump-Based backtracking for Constraint Satisfaction Problems », Artificial Intelligence, Vol. 136, pp. 147-188, 2002.

[Dec92] Dechter R., « Constraint Networks », In Encyclopaedia of Artificial Intelligence, Vol.1, pp. 276-285, Second Edition, 1992.

[Del95] Della Croce F., R. Tadei, G.Volta, « A genetic algorithm for the job shop problem », Computer and Operations Research, Vol. 22, n° 1, pp. 15-24, 1995.

[Dri87] Dridi N., J. M. Proth, « Ordonnancement des tâches : une méthode basée sur la technologie de groupe », Proceedings of the second International Conference on Production Systems, 1987.

[Dur02] Duron C., « Ordonnancement en temps réel des activités des radars », Thèse de Doctorat, Université de Metz, décembre 2002.

[Fia98] Fiat A., G. Woeginger, J. Sgall, « Lectures Notes in Computer Science : On-line algorithms, the state of the art, chapter 9 », Springer-Verlag, pp. 196-231, 1998.

[Fis83] Fisher M. L., B. J. Lageweg, J. K. Lenstra, A. H. G. Rinnooy Kan, « Surrogate duality relaxation for job-shop scheduling », Discrete Applied Mathematics, Vol. 5, 1983.

[Fis81] Fisher M. L., « The lagrangian relaxation method for solving integer programming problems », Management Science, Vol. 27, 1981.

[Fis76] Fisher M. L., « A dual algorithm for the one-machine scheduling problem », Mathematical Programming, Vol. 11, pp. 23-37, 1976.

[Gag01] Gagnié C., M. Gravel et W.L. Price, « Optimisation par colonies de fourmis pour un problème d'ordonnancement industriel avec temps de réglage dépendant de la séquence », 3ème Conférence Francophone de MOdélisation et de SIMulation, MOSIM'01, Troyes, 2001.

[Gar79] Garey M., D. Johnson, « Computer and Intractability : a Guide to the Theory of NP- Completeness », Freeman W-H, San Francisco, 1979.

[Gar78] Garey M., R.L. Graham, D.S. Johnson, « Performances guarantees for scheduling algorithms », Operations Research, Vol. 26, pp. 3-21, 1978.

[Gar03] Gargouri E., « Ordonnancement coopératif en industries agro-alimentaires », Thèse de Doctorat, Université des Sciences et Technologies de Lille 1, 2003.

[Gar01] Gargouri, E., S. Hammadi, P. Borne, 2001, « New constraints of agro-food industry scheduling problem », Acte de IFDICON'2001, European Workshop on Intelligent Forecasting, DIagnosis and CONtrol, Santorin, pp. 73-80, 2001.

[Gau87] Gaudel M.C., M. Soria, C. Froidevaux, « Types de données et algorithmes », Volume 2 , « recherche, tri, algorithmes sur les graphes », Collection Didactique, D-004, Inria, Rocquencourt, 1987.

[Geo74] Geoffrion A. M., « Lagrangian relaxation for integer programming», Mathematical Programming Studies, Vol. 3, 1974.

[Ghe99] Ghedjati F., « Genetic algorithms for the job-shop scheduling problem with unrelated paralleled constraints: Heuristics mixing method machines and precedence », Computers and Industrial Engineering, Vol. 73, pp. 39-42, 1999.

[Ghe94] Ghedjati F., « Résolution par des heuristiques dynamiques et des algorithmes génétiques du problème d'ordonnancement de type job-shop généralisé », Thèse de Doctorat, Université de Paris VI, 1994.

[Glo97] Glover F., M. Laguna, « Tabu search », Kluwer Academic Publishers, 1997.

[Glo89] Glover F., « Tabu search », ORSA, Journal of Computing, Vol. 1, n° 2, pp.190-206, 1989.

[Gol94] Goldberg G.E., « Algorithmes génétiques », Éditions Addison-Wesley, Paris, 1994.

[Got93] Gotha, « Les problèmes d'ordonnancements », RAIRO-Recherche Opérationnelle, Vol. 27, pp. 77-150, 1993.

[Gro88] Grotschel M., L. Lovasz, A. Schrijver, « Geometric Algorithms and Combinatorial Optimization », Springer-Verlag, Berlin Heidelberg, 1988.

[Gué00] Guéret C., N. Jussien, C. Prins, « Using intelligent backtracking to improve branch-and-bound methods: An application to Open-Shop problems », European Journal of Operation Research, Vol. 127, pp. 344-354, 2000.

[Gza01] Gzara, M., « Méthode coopérative d'aide multicritère à l'ordonnancement flou », Thèse de Doctorat, Université des Sciences et Technologies de Lille 1, 2001.

[Hak99] Hakan A. R., G. Melhem, D. Mosse, P. Mejia-Alvarez, « Optimal reward-based scheduling of periodic real-time tasks », In IEEE Real-Time Systems Symposium, pp. 79-89, 1999.

[Hol75] Holland J.H., « Adaptation in natural and artificial systems », PhD, Thesis Michigan Press University, Ann Arbor, Michigan.

[Hao99] Hao J.K., P. Galinier, M. Habib, « Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes », Revue d'Intelligence Artificielle, Vol. 18, n°1, pp. 63-82, 1999.

[Jac55] Jackson J.R., « Scheduling a production line to minimize maximum tardiness », Research Report 43, Management Science Research Report, University of California, Los Angeles, 1955.

[Jai99] Jain A.S., S. Meeran, « Deterministic job-shop Scheduling : past, present and future », European Journal of Operational Research, Vol. 113, pp. 390-434, 1999.

[Jai97] Jain A. K., H. A. El Maraghy, «Single process plan scheduling with genetic algorithm», Production Planning and Control, Vol. 8, n° 4, pp. 363-376, 1997.

[Jou02] Jouglet A., P. Baptiste, J. Carlier, « Exact procedures for single machine total cost scheduling », IEEE Transactions on Systems, Man and Cybernetics, SMC-WA2K1, 2002.

[Kac03] Kacem I., « Ordonnancement multicritère des job-shops flexibles : formulation, bornes inférieures et approche évolutionniste coopérative », Thèse de Doctorat, Université de Lille 1, 2003.

[Kir83] Kirkpatrick S., C.D Gelatt, M.P. Vecchi, « Optimisation by simulated annealing », Science, Vol. 220, pp. 671-680, 1983.

[Kob95] Kobayashi S., I. Ono, M. Yamamura., « An efficient genetic algorithm for job-shop scheduling problems », Proceeding of 6th International Conference on Genetic Algorithms, pp. 506–511, 1995.

[Law87] Lawler E. L., J. K. Lenstra , A. H. G. Rinnooy Kan, D. Shmoys, « The Travelling Salesman Problem : A Guided Tour of Combinatorial Optimization », Editions John Wiley & Sons, 1987.

[Lem91] Lemonias H., « Ordonnancement d'Atelier à Tâches : Une Approche par Décomposition », Thèse de Doctorat, Institut National Polytechnique de Grenoble, 1991.

[Len77] Lenstra J.K., A.H.G. Rinnooy Kan, P. Bruker, « Complexity of machine scheduling problems », Annals of Discrete Mathematics, Vol. 1, pp. 343-362, 1977.

[Lio07] Liouane N., I. Saad, S. Hammadi, P. Borne, « Ant systems & Local Search Optimization for flexible Jop shop Scheduling Production », International Journal of Computers, Communications & Control, Vol. 2, n°2, pp. 174-184, 2007.

[Lio98] Liouane N., « Contribution à l'élaboration d'un outil d'aide à la décision pour l'ordonnancement de production en environnement incertain », Thèse de Doctorat, Université des Sciences et Technologies de Lille1, 1998.

[Lop01] Lopez P., F. Roubellat, « Ordonnancement de la production », Hermès Sciences, IC2 Productique, 2001.

[Lop99] Lopez P., P. Esquirol, « L'ordonnancement », Éditions Économia, 1999.

[Mah75] McMahon G.B., M. Florian, « On scheduling with ready times and due dates to minimize maximum lateness », Operations Research, Vol. 23, n° 3, pp. 475-482, 1975.

[Mar96] Martin P., D.B. Shmoys, « A new approach to computing optimal schedules for the job-shop scheduling problem », In the 5th International IPCO Conference, pp. 389–403, 1996.

[Mau04] Mauguière P., « Étude de problèmes d'ordonnancement disjonctifs avec contraintes de disponibilité des ressources et de préparation », Thèse de Doctorat, Université de Tours, 2004.

[Mes99] Mesghouni K., « Application des algorithmes évolutionnistes dans les problèmes d'optimisation en ordonnancement de la production », Thèse de Doctorat, Université des Sciences et Technologies de Lille 1, 1999.

[Mok83] Mok A., « Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment », PhD Thesis, MIT Laboratory for Computer Sciences, 1983.

[Mon74] Montanari U., « Networks of Constraints: Fundamental Properties and Applications to Picture Processing », Artificial Intelligence, Vol. 7, pp. 95-132, 1974.

[Nak91] Nakano R., T. Yamada, « Conventional genetic algorithm for job shop problems », Proceedings of the fourth International Conference on Genetic Algorithms, University of California, pp. 474 -479, 1991.

[Now94] Nowicki E., « An approximation algorithm for a single-machine scheduling problem with release times, delivery times and controllable processing times », European Journal of Operational Research, Vol. 72, n° 1, pp. 74-82, 1994.

[Orm98] Orman A. J., « Modelling for the control of a complex radar system », Computers Operations Research, Vol. 25, n° 3, pp. 239-249, 1998.

[Pan92] Panwalkar S.S., R. Rajagopalan, « Single-machine scheduling with controllable processing times », European Journal of Operational Research, Vol. 59, pp. 298-302, 1992.

[Pas05] Paschos V., « Optimisation combinatoire, concepts fondamentaux », Tome 1, Éditions Hermès Science, Paris, 2005.

[Pen94] B. Penz, « Constructions agrégatives d'ordonnancements pour des job-shops statiques, dynamiques et réactifs », Thèse de Doctorat, Université Joseph Fourier - Grenoble 1, 1994.

[Por88] Portmann M.C., « Méthodes de décompositions spatiale et temporelle en ordonnancement de la production », APII, Vol. 22, 1988.

[Pot80] Potts C.N., « Analysis of a heuristic for one machine sequencing with release dates and delivery times », Operations Research, Vol. 28, n° 6, pp. 1436-1441, 1980.

[Pot80a] Potts C. N., « An adaptative branching rule for the permutation flow-shop problem », European Journal of Operational Research, Vol. 5, pp.19–25, 1980.

[Roy70] Roy B., « Algèbre moderne et théorie de graphe », Éditions Dunod, Paris, 1970.

[Sch96] Schwiegelshohn U., « Preemptive weighted completion time scheduling of parallel jobs », In European Symposium on Algorithms, pp. 39-51, 1996.

[Sch71] Scharge L., « Obtaining optimal solutions to resource constrained net-work scheduling problems», 1971.

[Tam92] Tamaki H., « Maintenance of diversity in a genetic algorithm and application to the job shop scheduling », In Proceedings IMACS/SICE Int Symp On MRP2, pp. 869–869, 1992.

[Tan06a] Tangour F., I. Saad, « Multiobjective Optimization Scheduling Problems by Pareto-optimality in Agro-alimentary Workshop », International Journal of Computers Communications & Control, IJCCC**,** Vol. 1, n° 3, pp. 71-83, 2006.

[Tan06b] Tangour F., P. Borne, « Ordonnancement des opérations dans un atelier de production agroalimentaire en minimisant le coût », Revue e-STA, Vol. 3, n° 1, 2006.

[Tan06c] Tangour F., S. Hammadi, P. Borne, M. Benrejeb, « Cost optimal scheduling in an agro-food production workshop », Journal of Systems Science and Systems Engineering, JSSSE, à paraître.

[Ter02] Terrioux C., « Approches structurelles et coopératives pour la résolution des problèmes de satisfaction de contraintes », Thèse de Doctorat, Université d'Aix-Marseille I, 2002.

[Vac00] Vacher P., « Un système adaptatif par agents avec utilisation des algorithmes génétiques multi-objectifs : Application à l'ordonnancement d'atelier de type job-shop », Thèse de Doctorat, Université du Havre, 2000.

[Wan99] Chung Wang Y., L. Kwei-Jay, « Implementing a general real-time scheduling framework in the RED-Linux real-time kernel », In IEEE Real-Time Systems Symposium, pp. 246–255, 1999.

[Wig94] Wigderson A. , S. Ben-Davi, A. Borodin, M. Karp, G. Tardos, « On the power of randomization in on-line algorithms », Algorithmica, Vol. 11, pp. 2-14, 1994.

[Yam03] Yamada T., « Studies on Metaheuristics for Job-shop and Flow-shop Scheduling Problems », PhD. Thesis, Kyoto University, Kyoto, 2003.

[Yam92] Yamada T., R. Nakano, « A genetic algorithm applicable to large-scale job-shop problem », Parallel problem solving from nature, Editeur R.Männer and B. Manderick, Amesterdam, Vol. 2, pp. 281-290,1992.

[Zri05] Zribi N., « Ordonnancement des job-shops flexibles sous contraintes de disponibilité des machines », Thèse de Doctorat, Université des Sciences et Technologies de Lille, 2005.