République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université des sciences et de la Technologie Houari BOUMEDIENE (USTHB)

FACULTE GENI-ELECTRIQUE

DEPARTEMENT D'INFORMATIQUE

THESE

Présentée à l'USTHB pour l'obtention du grade de
MAGISTER EN INFORMATIQUE

PAR
M. Hakim Malek

Metaheuristiques Pour l'Optimisation Multicritères :

Etude Des Problèmes D'ordonnancement

M. A.Khelladi	*Professeur USTHB*	*Président*
M. M.Rahoual	*Chargé de cours USTHB*	*Directeur de thèse*
M. M.Ahmed Nacer	*Maître de conférence USTHB*	*Examinateur*
M. M.Aider	*Maître de conférence USTHB*	*Examinateur*
M R.Saad	*Maître de conférence USTHB*	*Examinateur*
M. E-G.Talbi	*Professeur U.Lille1*	*Examinateur*
M. V.Bachelet	*Maître de conférence USTHB*	*Examinateur*

Vilfredo Pareto

Remerciements

Mes remerciements sont adressés à mon promoteur *Malek Rahoual* pour son soutient pendant ces deux années de recherche. Je tiens à lui témoigner ma gratitude pour sa présence continuelle et ses aides précieuses. *A Mr Talbi*, je tiens à témoigner mon estime quant à sa participation à cette thèse et sa présence pendant mes stages à l'université de Lille.

Je tiens à présenter mes remerciements *à Mr Vincent Bachlet* et *M^{me} Clarisse Dhaenens* pour leur aide et leur soutient. Mes remerciements s'adressent aussi à l'ensemble de l'équipe *OPAC* du laboratoire *LIFL*.

Je saisis cette occasion pour exprimer ma reconnaissance à M. A.Kelladi pour avoir bien voulu présider le jury. Que l'ensemble des membres du jury qui ont accépté d'évaluer ce travail, trouvent ici l'expression de mes remerciements les plus sincères.

Résumé

Les problèmes d'ordonnancement ont reçu une grande attention, mais la plupart des travaux dans ce sens les traitent dans leur forme monocritère. Dans cette thèse, nous nous sommes intéressés à l'étude de ces problèmes dans leur version multicritère en utilisant les méta-heuristiques. Une étude détaillée des problèmes d'optimisation combinatoire mono et multicrière est menée. L'accent est mis sur les problèmes d'ordonnancement: Flow Shop, Job Shop, emploi du temps, routage de véhicules et satisfaction de contraintes.

Les techniques de recherche tabou, de recuit simulé et d'algorithmes génétiques sont discutées, leurs principes fondamentaux décrits.et les mécanismes d'hybridation et de parallélisation étudiées. Pour étayer cette étude, sont présentées différentes manières d'application de ces méthodes pour la résolution des problèmes d'ordonnancement cités ci-dessus. Les variantes multicritères de ces méthodes sont aussi présentées.

Un intérêt particulier est porté aux algorithmes génétiques(AG), vu leur grande efficacité face aux problèmes d'optimisation combinatoire. L'extension de cette méthode au cas multicritère est particulièrement exposée.

Un algorithme génétique multicritère est proposé pour le problème de Flow Shop bi-critères avec pour objectifs, la minimisation du temps de terminaison total et de la somme des retards enregistrés. Pour cela les variantes suivantes sont implémentées:

- ✓ Six stratégies de sélection: sélection par somme pondérée des objectifs, sélection parallèle, sélection NSGA, sélection NDS, sélection WAR et sélection NDS élitiste.
- ✓ Trois stratégies de maintien de la diversité: sharing génotypique, sharing phénotypique et le sharing combiné.
- ✓ Une hybridation d'algorithme génétique multicritère avec la recherche locale.
- ✓ Deux modèles d'AG parallèle, l'un adoptant une approche centralisée et l'autre distribuée.
- ✓ Deux méthodes de génération de la population initiale, l'une se basant sur une approche aléatoire et l'autre adoptant une version probabiliste de l'heuristique NEH.
- ✓ Un gé nérateur d'instance de Flow Shop bi-critères est développé, fournissant un cadre de test et de comparaison idéal pour les méthodes d'optimisation.

Mots clés: Optimisation combinatoire, Optimisation multicritère, Méta-heuristiques, Algorithmes Génétiques, Maintien de la diversité, Parallélisme, Hybridation.

Introduction Générale

Le monde réel regorge de situations où on est face à un grand ensemble d'alternatives dont il s'agit de choisir la meilleure possible. Le parcours exhaustif de ces alternatives étant impossible, il est alors nécessaire d'élaborer des stratégies de recherche au sein de cet ensemble. Cette recherche consiste en un parcours partiel des solutions potentielles du problème tout en garantissant l'atteinte ou l'approximation de la meilleure solution. Toute recherche partielle, implique un compromis entre les tendances d'exploration et d'exploitation, et face à des problèmes multi-modales la qualité de cette balance se définit comme critère déterminant de la qualité de la recherche.

La pratique nous fait observer que les problèmes d'optimisation prennent rarement l'aspect d'une optimisation monocritère. Souvent le problème est formuler sous forme d'un ensemble d'objectifs qui peuvent éventuellement être contradictoires. Chaque objectif est modélisé par une fonction scalaire dont il s'agit de minimiser ou maximiser la valeur. Le domaine d'ordonnancement ne fait pas exception à cette règle[Bil 99][T'ki 99]. La plupart de ces problèmes regroupent plusieurs objectifs liés principalement au désir d'accomplir les tâches le plus tôt et d'éliminer les retards de traitement ou ceux dus aux délais d'attente et de stockage. La difficulté supplémentaire induite par le caractère multi-objectifs du problème apparaît dans l'ambiguïté qui entoure la notion d'optimalité. En l'absence d'une relation d'ordre total permettant la comparaison entre des solutions différentes, la notion d'optimalité *Pareto* a jouit d'une certaine unanimité au sein de la communauté d'optimisation multicritère. Cette notion est liée à la relation de dominance qui décrit une relation d'ordre partiel entre les solutions. L'objectif d'une optimisation multicritère revient alors à la génération d'un ensemble de solutions non dominées.

Selon le type d'interactions établi entre l'algorithme de recherche et le décideur, selon l'objectif de la recherche : exacte ou approchée et selon la définition donnée à la notion d'optimalité, plusieurs approches de résolution ont vu le jour[Tal 99][Zit 99]. Les méta-heuristiques se sont avérées comme de performantes méthodes dans le cadre de l'optimisation monocritère[Pre 95]. Un fait, qui a poussé les chercheurs à proposer des extensions multicritère à ces techniques[Tal 99]. Pour augmenter les performances de telles méthodes, des mécanismes supplémentaires ont été introduits. Ces derniers serviront à l'amélioration de la diversification[Oba 98] et de l'intensification[Mab 97][Ros 95] de la recherche et à l'amélioration des temps de recherche via le parallélisme et l'hybridation[Tal 99b] de plusieurs techniques.

De toutes les méta-heuristiques, les Algorithmes Génétiques ont été les plus exposés à ce phénomène d'adaptation[Coe 96] [Zit 99] [Vel 99] dû au fait que la méthode travail sur une population de solutions en parallèle. Ce qui répond au besoin de produire un ensemble de solutions efficaces en une seule exécution. La principale différence entre ces extensions réside dans le mécanisme de rangement (*ranking*) des solutions appartenant à la population.

Le problème de flow shop[Mar 98] qui se présente comme un cas particulier des problèmes d'atelier, servira, lors de ce travail de champs d'application et d'investigation des approches de résolution multicritère.

Le travail réalisé est organisé en quatre parties. Dans la première, nous introduisons les notions liées aux problèmes d'optimisation combinatoire. Nous donnons les différentes raisons de leurs difficultés ainsi que les méthodes de modélisation et de résolution proposées. Au cours de cette étude, les techniques de recherche Branch & Bound[Bra 91], programmation dynamique[Sim 82], programmation linéaire[Kuf 98], algorithme A*[Far 85] seront décrites. Ces méthodes se présentent généralement comme des heuristiques de recherche arborescente où il s'agit de trouver la solution par amélioration ou construction itérative. Ces méthodes se distinguent aussi par le caractère exact ou approché de la solution proposée. Vu qu'un problème d'optimisation est souvent doublé d'un problème de satisfaction de contraintes, nous présentons un état de l'art sur les approches proposées pour sa résolution. Nous formulons à la fin de cette partie une étude approfondie des problèmes d'ordonnancements. Les problèmes de Flow Shop, de Job Shop, d'emploi du temps et de routage de véhicules sont décrits et nous donnons pour chacun l'ensemble des critères d'optimisation qui peuvent être considérés ainsi que les différentes méthodes de résolution appliquées.

La deuxième partie, est consacrée à l'étude des méta-heuristiques. Les mécanismes de base régissants cette classe d'approches sont développés. Puis une étude détaillée de trois méta-heuristiques: Algorithmes Génétique, Recherche Tabou et Recuit Simulé est proposée. Pour chacune de ces méthodes, nous mettons l'accent sur leurs mécanismes de diversification et de parallélisation [Tal 95][Tal 98a][Ram 96] ainsi que leurs application pour la résolution des problèmes d'ordonnancement cités ci-dessus. Les différentes stratégies d'hybridation de ces techniques sont par la suite présentées.

Dans la troisième partie, nous introduisons les notions liées à l'optimisation multicritère[Coe 96] tel que l'optimalité *Pareto*, la dominance et l'optimalité Min-Max[Coe 98]. Nous mettons l'accent sur la nature des problèmes d'optimisation multicritère et sur les méthodes de résolution proposées pour ces problèmes. Nous présentons les versions multicritère des techniques de recherche Branch & Bound, de programmation dynamique et d'algorithme A*. Nous passons en revu tout particulièrement l'extension des méta-heuristiques au cas multicritère. Un intérêt particulier sera porté sur l'étude des algorithmes génétiques multicritère.

Dans la dernière partie, nous proposons la mise en œuvre d'un algorithme à base d'algorithmes génétiques pour la résolution du problème de flow shop bicritère F/permu,d_i/(C_{max}, T). Il s'agit dans ce cas de minimiser le temps d'accomplissement final des tâches (*makespan*) ainsi que de la somme des retards enregistrée. Différents mécanismes de *ranking*, de *sharing*, d'hybridation, de parallélisation et de génération de la population initiale seront implémentés. Dans un premier temps, 6 stratégies de sélection sont mises en œuvre et leurs performances comparées. Par la suite, 3 variantes pour le maintien de la diversité basées sur la stratégie de niches écologiques sont testées. L'hybridation de l'algorithme développé avec la recherche locale est proposée, ainsi que deux modèles de parallèlisation. Une adaptation probabiliste de l'heuristique *NEH* et enfin exposée servant à la génération de la population initiale. Un générateur d'instances de problème de flow shop bi-critères est développé, palliant de ce fait au manque de benchmarks dans ce cadre. Le générateur se présente comme extension des problèmes de *Taillard* monocritère[Tai 93] très référencé dans la littérature[Ben 98][Now 99]. Ce générateur permettra l'évaluation des performances des résultats trouvés ainsi que leurs interprétation et comparaison.

Partie I: Problèmes d'optimisation Combinatoire : Pourquoi est-ce aussi difficile ?

I.1 Introduction

La vie courante nous offre un vaste éventail de problèmes où on se trouve en face d'un ensemble important de solutions potentielles dans lequel il s'agit de trouver la meilleure solution qui soit. Le nombre important de ces choix rend impossible leur parcours exhaustif en vue de recenser le choix le plus adéquat.

De tels problèmes sont très fréquents dans tous les domaines professionnels. Dans un atelier[Bil 99] de production de pièces, celles-ci doivent passer par un certain nombre de traitements réalisés sur différents types de machines. La séquence d'usinage n'étant pas nécessairement la même pour toutes les catégories de pièces, un bon plan d'usinage est alors nécessaire afin de garantir une productivité maximale.

Le monde informatique lui-même ne manque pas de problèmes de ce genre. Par exemple une application sensée être porté sur un environnement réparti est composée de plusieurs tâches élémentaires régies par des contraintes et communicantes par envoi de messages[Mun 91]. Le système d'exploitation doit répartir l'ensemble de ces tâches sur les différents processeurs afin d'assurer un temps de réponse rapide.

L'objectif de cette section est de présenter une étude sur les particularités de tels problèmes, les différents types de modélisation possibles et les raisons de leur difficulté. Les méthodes appliquées pour la résolution de ces problèmes sont ainsi exposées tout en formulant une analyse critique quant à leur efficacité. Notre attention sera particulièrement portée sur les problèmes d'ordonnancement.

I.2 Problèmes Combinatoires

Un problème combinatoire[Pap 95] est un problème où les solutions se présentent comme une collection de paramètres, d'attributs ou de prédicats dont il s'agit de trouver la meilleure combinaison possible. Un tel problème peut être soit un problème de décision, un problème de recherche ou un problème d'optimisation[Car 88], selon la question à la quelle on est censé répondre.

I.2.1 Problème de décision

C'est un problème où la résolution se limite à la réponse par "oui" ou par "non" à la question de savoir s'il existe une solution au problème. Par conséquent, il n'est pas nécessaire de trouver la solution proprement dite.

I.2.2 Problème de recherche

Dans ce cas précis, la résolution du problème ne s'arrête plus au point de savoir si le problème admet ou non une solution. L'algorithme doit être en mesure de fournir la solution si celle-ci existe. Par conséquent, tout problème de décision peut être étendu à un problème de recherche.

I.2.3 Problème d'optimisation

Un problème d'optimisation[Pap 95][Car 88] consiste à trouver parmi un ensemble de solutions potentielles celle qui répond le mieux à certains critères décrits sous forme d'une fonction objectif à maximiser ou minimiser. Ces critères peuvent être par exemple la réduction du coût, la réduction du temps ou la maximisation du gain.

Un problème d'optimisation est généralement doublé d'un problème de satisfaction de contraintes (PSC), particulièrement pour les problèmes d'ordonnancement. Dans ce cas, il n'est plus suffisant que la solution trouvée réponde le mieux aux critères d'optimisation, elle doit aussi satisfaire un ensemble de contraintes[Cau 95]. Un tel problème est mathématiquement modélisé par : *Max* ou *Min f(x) sous la contrainte g(x*) où *g* désigne l'ensemble des contraintes. Dans le cas où les fonctions *f* et *g* sont linéaires, le problème revient à un problème d'optimisation linéaire susceptible d'être résolue facilement.

Avant d'entamer notre étude il est nécessaire de délimiter la classe de problèmes d'optimisation qui fera l'objet de cette étude. Premièrement, nous éliminons tous les problèmes pour lesquels il existe des méthodes de résolution exactes donnant la solution en un temps raisonnable(polynomial). Ce qui est le cas de nombreux problèmes d'optimisation notamment en mathématiques. Deuxièmement, nous éliminons tous les problèmes dont on ne dispose pas de modélisation ou de description formelle. Ainsi les problèmes dont l'expertise humaine ne peut être reprise de manière fidèle, généralement, en raison du fait que l'expert lui-même a une connaissance vague de la manière dont il procède, ne rentrent pas dans le cadre de notre étude. Reste aussi à éliminer tous les problèmes dont nous ne disposons pas de méthodes d'estimation objectives et quantitatives de la qualité d'une solution en un nombre fini d'étapes.

Exemple I.1: Pour les besoins des explications ultérieures, nous introduisons l'exemple ci-dessous. Soit un problème d'ordonnancement constitué de 4 Jobs $\{J_1,J_2,J_3,J_4\}$ à ordonner sur une machine. Le tableau ci-dessous présente les temps de traitement ainsi que les dates d'achèvement au plus tard des jobs. L'objectif est de planifier le passage des jobs sur la machine afin de réduire au maximum les retards d'usinage.

Job	**Temps de Traitement (d_i)**	**Date de retard (l_i)**
J_1	***10***	***15***
J_2	***7***	***20***
J_3	***5***	***13***
J_4	***3***	***7***

Objectif : Minimiser $f(u) = \sum_{i=1}^{4} \max(0, C_i(u) - l_i)$ (I.1)

$C_i(u)$ correspond au temps d'achèvement du job J_i relativement à l'ordonnancement u.

I.3 Méthodes de représentation des problèmes

La modélisation d'un problème d'optimisation passe inévitablement par

- ✓ Trouver le moyen de représentation d'une solution potentielle du problème: ce qui inclut l'extraction des variables intéressantes entrant dans la définition du problème.

- ✓ Trouver la manière de mesurer l'adéquation d'une telle solution: Cerner les critères qui rentrent dans l'estimation d'une solution donnée.

On distingue principalement deux types de représentation d'un problème[Far 85] qui décrivent du coup la manière de le résoudre.

I.3.1 Représentation par graphe d'états

Dans ce type de représentation[Far 85] on organise l'ensemble des états ou solutions du problème en un graphe orienté. Les sommets du graphe représentent les solutions potentielles, et les arcs la possibilité de passer d'une solution à une autre via une transformation simple(opérateur). Un problème donné peut être représenté de plusieurs manières selon le type d'opérateurs choisis, puisque ce sont les opérateurs qui décrivent les liens de voisinage entre les différentes solutions.

Un état est une description de la situation initiale dans la quelle se trouve le problème (la recherche de la solution optimale débute généralement par une solution de départ non nécessairement réalisable) ou une des solutions intermédiaires rencontrées pendant la résolution. Cette description met en jeux l'ensemble des données qui décrivent le problème et qui peuvent se présenter sous différentes formes (prédicats, relations, variables, vecteurs, tableaux, …, etc.).

Notons U l'ensemble discret non nécessairement fini d'états, qui constituent l'espace de recherche. S l'ensemble des opérateurs (s_1, s_2, s_3, …, s_n), tel que $s_i : U_i \rightarrow U$, pour $U_i \subset U$. De ce fait chaque opérateur s_i offre une méthode de navigation à l'intérieure de l'espace de recherche. Et chaque opérateur a un domaine spécial d'états U_i sur lequel il est applicable. Le couple G= (U,S) définit le graphe d'états.

Un problème d'optimisation dans ce cas est formulé par

- ✓ Une solution initiale d'où débutera la recherche. Ce premier paramètre peut ne pas figurer comme donnée du problème.
- ✓ L'ensemble d'opérateurs S de changement d'états.
- ✓ L'ensemble T d'états terminaux qui représentent l'ensemble des solutions du problème.

I.3.2 Représentation par graphe de sous-problèmes

Ce type de représentation[Far 85] s'appuie sur la décomposition comme moyen de contrôle de la complexité inhérente de certains problèmes. La technique diviser pour régner consiste à décomposer le problème initial en sous problèmes de difficulté moindre. Ces derniers

peuvent, à leur tour, être sujets d'une décomposition. Ce processus est réitéré jusqu'à atteindre des problèmes trivialement résolvables. La technique de programmation dynamique représente ce type d'approches [Sim 82][Car 90].

La décomposition du problème peut aussi se faire par fragmentation successive et hiérarchique de l'espace de recherche. Les éléments d'un ensemble partagent les même valeurs de certaines variables. La méthode de Branch and Bound [Bra 91][Cas 95][Sha 91] prône se type de décomposition. Le problème, dans ce cas, est représenté comme un arbre dont les nœuds correspondent aux sous problèmes ou à des sous ensembles de l'espace de recherche. La figure ci-dessous correspond à une représentation en graphe de décomposition de l'exemple I.1. La présence d'un x dans une chaîne signifie qu'aucun job n'est encore assigné à cette position. Par exemple la chaîne 24xx représente l'ensemble de tous les ordonnancements, planifiant le job J_2 en première position suivi par J_4 en seconde position.

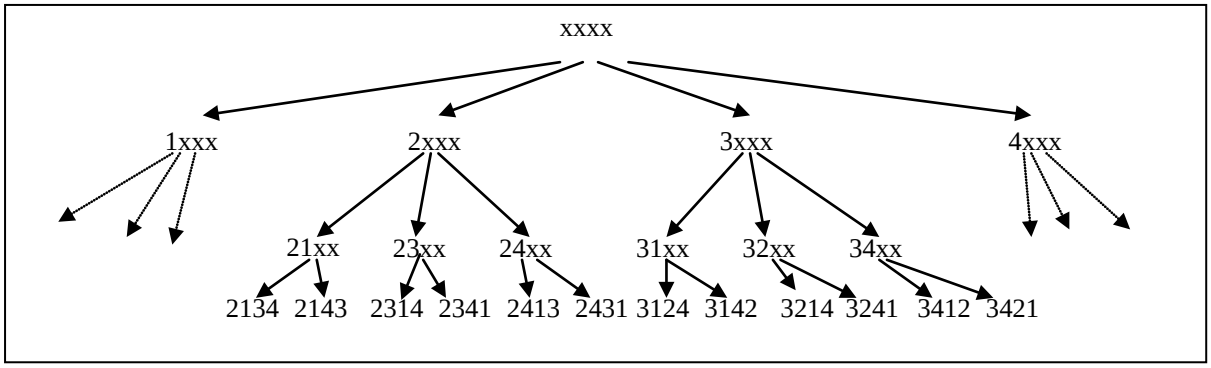


Figure I.1: Représentation par graphe de sous problèmes

I.3.3 Etude comparative des deux représentations

En réalité un problème est aussi bien représentable par un graphe d'états que par un graphe de sous-problèmes. Comme il est possible de passer de l'une à l'autre [Far 85]. Quelques fois les même opérateurs peuvent être interprétés en tant que décomposition dans une représentation ou en tant que simple transformation d'état vers un autre. La différence principale réside dans la façon de construire la solution. Dans une représentation par graphe d'états, la recherche est effectuée en démarrant d'une solution complète non nécessairement optimale. La solution est révisée et améliorée au fur et à mesure que la recherche avance. Dans le cas d'arbre de décomposition, la solution est produite par construction itérative de celle-ci en démarrant de rien ou d'une solution partielle(construction).

Les critères de choix d'une représentation font intervenir

- ✓ La complexité algorithmique de résolution du problème dans la représentation adoptée. En effet, chaque modèle de représentation dénote la manière d'investigation de l'espace de recherche. Une représentation par graphe d'états suggère que la recherche s'effectue en démarrant d'une solution initiale qui sera révisée(améliorée) via des opérateurs jusqu'à arriver à la solution optimale. La modélisation par graphe de sous-problèmes suggère que le problème soit traité en le décomposant en sous problèmes plus simples via des opérateurs de décomposition précis jusqu'à arriver à des problèmes dont on connaît directement la solution. Ce qui signifie que la solution est construite au fur et à mesure de la recherche (complétée).

- ✓ L'aspect intuitif ou naturel de la représentation pour celui qui modélise et la simplicité de la représentation adoptée rend le problème plus maîtrisable et la traduction de l'expertise humaine en heuristiques et règles de recherche plus efficace.

Le réel défit dans les deux types de représentation est d'extraire les opérateurs les plus adaptés au problème traité, que se soit de transformation ou de décomposition.

I.4 Paysage d'un problème

L'utilisation de la métaphore de paysage a été récemment introduite dans le domaine des algorithmes évolutionnistes[Glo 96][Fon 98]. Une définition intuitive de la notion de paysage d'un problème est la suivante : on met l'ensemble des solutions potentielles sur l'axe des abscisses, et l'adéquation des solutions sur l'axe des ordonnées. Un graphe est alors obtenu, composé de plateaux, de pics, de valets, de canions plus ou moins rudes. On imagine alors l'algorithme d'optimisation comme un grimpeur dont le but est d'atteindre le plus haut point. Cependant, ce paysage n'est pas celui vu par l'algorithme, dont la vue est restreinte au voisinage immédiat du point où il se trouve. De ce fait, l'algorithme ne sait pas dans quelle direction se trouve le plus haut point et dans le meilleur des cas il ne peut que le deviner.

Dans le cas d'une représentation par graphe d'états, l'algorithme explore ce paysage en se déplaçant d'un point à un autre moyennant des opérateurs. Le voisinage d'un point correspond aux solutions directement abordables depuis la solution actuelle via un opérateur simple. Par conséquent, nous pouvons dire que le voisinage d'un point est totalement défini par le type d'opérateurs considérés. Cependant, les paysages sur lesquels devra travailler l'algorithme sont de dimensions dépassant largement trois ce qui sort du domaine de l'imagination humaine et sont de ce fait une source de confusion et d'incompréhension[Glo 96].

Ajouter à cela le fait qu'aucun moyen formel à ce jour n'est disponible pouvant nous informer sur les proportions des optimums locaux, sur leurs nombre ni sur leur distribution à travers l'espace de recherche[Fon 98].

I.5 Paysage Nk

L'espace de recherche pour un problème d'optimisation est composé de points chacun représentant une solution potentielle du problème. Un point ou une solution est défini par l'ensemble des caractéristiques intéressantes qu'il présente. L'objectif de l'algorithme d'optimisation est de trouver l'individu qui présente la meilleure combinaison de ces caractéristiques. Un paysage est dit Nk quand la codification des solutions est faite de façon à ce que l'adéquation(*fitness*) d'une caractéristique soit calculée à partir des valeurs de k autres caractéristiques.

Pour illustrer cette notion, prenons l'exemple du problème d'emploi du temps dans un établissement éducatif. L'espace de recherche dans ce cas est l'ensemble de tous les emplois du temps imaginables(réalisables ou pas) codés par des vecteurs associant une entrée à chaque cours décrivant la période à la quelle il se déroule. L'adéquation de la planification d'un cours à une certaine période est en fonction des autres affectations faites aux cours concernants la même classe(vérification des conflits sur les classes). Comme conséquence de cette remarque, la complexité du problème d'emploi du temps augmente proportionnellement aux nombre de cours de chaque classe.

Pour le problème de maximisation de la fonction X^2, un nombre entier est représenté par son code binaire. L'adéquation de la valeur d'un bit n'est dépendante d'aucun autre bit. En effet, une valeur 1 d'un bit est toujours plus adapté qu'un 0 qu'elle que soit la valeur des autres bits.

Les dépendances entre attributs d'une solution sont largement dues aux contraintes auxquelles devra répondre celle-ci. Par conséquent, la complexité d'un problème est mesurable par considération de trois points essentiels:

- ✓ Le nombre d'attributs pris en compte dans la codification d'une solution donnée.
- ✓ Le nombre de contraintes auxquelles doit satisfaire la solution.
- ✓ Le nombre d'attributs mis en jeux dans chaque contrainte.

C'est en effet, ces deux derniers points qui déterminent le degré de complexité du paysage. La difficulté du problème apparaît alors dans: premièrement la satisfaction des contraintes, qui est à lui seul un problème Np-complet[Fri 94]. Deuxièmement, la difficulté de choisir les bons opérateurs qui assurent qu'après une transformation sur une solution celle-ci ne sera pas sujette à de grands bouleversements dans son adéquation(les attributs étant très interdépendant rend chaque modification sur un attribut d'un impact imprévisible sur la convenance des autres attributs).

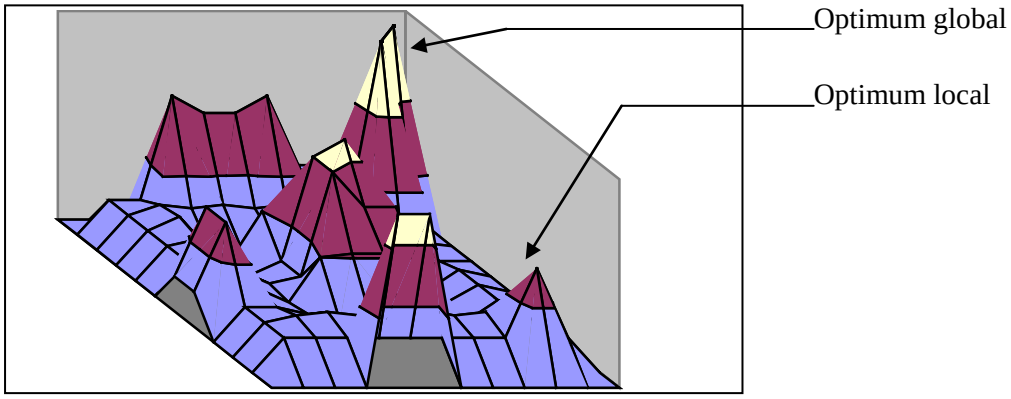


Figure I.2: Paysage à plusieurs optimums

Les paysages auxquels devra faire face l'algorithme sont souvent des paysages Multi-modales, ce qui signifie que plusieurs pics peuvent se présenter(Figure I.2). L'algorithme n'a souvent pas le moyen de déduire si le pic en lequel il se trouve est le plus haut parmi les autres ou non(risque de tomber sur un optimum local).

I.6 Complexité des problèmes d'optimisation combinatoire

La difficulté liée aux problèmes d'optimisation a rendu impossible le développement d'algorithmes efficaces donc constructifs pouvant les résoudre. Du terme efficace nous sous-entendons que la réponse doit se faire en un temps raisonnable donc polynomial[Pap 95]. Un algorithme est dit de complexité polynomiale quand la fonction décrivant le temps de réponse de l'algorithme en fonction de la taille du problème est bornée par un polynôme. De ce fait il n'est plus suffisant de trouver une solution algorithmique à un problème, encore faut-il prouver qu'elle n'est pas de complexité exponentielle.

Les problèmes dont on ne connaît pas d'algorithmes polynomiaux pouvant les résoudre sont dits problèmes Np-complet[Pap 95]. Malheureusement la plupart des problèmes

d'optimisation combinatoire rencontrées dans la vie réelle, particulièrement ceux d'ordonnancement semblent être au sommet des problèmes Np-complet de point de vue complexité.

I.7 Heuristiques de recherche

Les méthodes de résolution algorithmiques ne sont applicables que pour des cas très précis et spécifiques de problèmes combinatoires. De ce fait, de telles approches ne peuvent être généralisées à la résolution des problèmes en leurs généralités.

En l'absence de telles méthodes, les solutions adoptées actuellement sont des solutions de recherche itérative. Dans le sens qu'on opère par amélioration ou construction itérative de la solution. Selon le type de représentation du problème adopté (graphe d'états ou de décomposition), nous serons en face d'un arbre ou d'un graphe. Les heuristiques sont donc un moyen de guidage de cette recherche à l'intérieure de l'arbre(graphe). L'algorithme opère par itération, chaque itération repose sur deux étapes clés[Far 85] :

- ✓ L'étape de développement des alternatives.
- ✓ L'étape de sélection d'une alternative.

I.7.1 L'étape de développement

L'étape de développement consiste à déterminer l'ensemble des opérateurs applicables à partir de la situation actuelle. Pour un graphe d'états, cela voudra dire la détermination de l'ensemble des états accessibles(voisins) à partir de l'état présent. Alors que dans un graphe de décomposition, cela reviendrait à recenser l'ensemble des décompositions possibles du sous-problème actuel.

I.7.2 L'étape de choix

L'existence de plusieurs alternatives possibles implique un choix non déterministe. Le non-déterminisme dans ce cas correspond à l'absence d'un moyen de choix irrévocable[Far 85] (à valeur sûre). Par conséquent, il faut développer une stratégie qui garantisse les meilleurs choix et dans le cas contraire la possibilité de les remettre en cause.

L'exploration du graphe en tenant compte de la possibilité des retours arrières implique souvent une explosion combinatoire. Par exemple, pour un graphe ayant en moyenne K alternatives valides à chaque étape de résolution et nécessitant n itérations, le temps mis pour l'exploration de toutes les possibilités sera de l'ordre de $O(K^n)$. Pour K égal à 5, n à 10 et le temps d'une itération à 25 ms ceci, demanderait trois jours de temps machine.

Le rôle d'un algorithme de recherche est de guider la recherche lors du choix des alternatives tout en gérant les retours arrière sur ces choix et en évitant l'explosion combinatoire. Tout cela en développant une stratégie donc une heuristique de recherche basée sur des considérations précises et judicieuses qui réduise l'ensemble des alternatives à explorer. La qualité d'un tel algorithme serait estimée sur la base des trois critères suivants :

- ✓ La terminaison de l'algorithme.
- ✓ L'admissibilité de l'algorithme: ce qui désigne le degré de satisfaction ou du rapprochement de la solution trouvée de l'objectif voulu.

- ✓ La complexité de l'algorithme.

I.8 Approche de recherche exacte:

Le terme "exact" sous-entend que l'objectif de la recherche est d'atteindre la solution optimale est non simplement de l'approcher. On distingue différentes manières d'organiser la recherche selon la façon dont est réalisée l'étape de développement ou celle du choix. Lors de l'étape de développement des alternatives, on peut choisir soit (a)- de développer la totalité des alternatives avant de choisir l'une d'elle, soit (b)- de ne développer qu'une seule alternative en maintenant une information indiquant que l'état n'est pas totalement développé en vue d'un retour éventuel.

Quand plusieurs alternatives se présentent on peut choisir de ne tenir compte que de l'ordre dans lequel les alternatives sont rencontrées ou d'associer un critère de coût (priorité) à chaque alternative possible. Selon ces distinctions on dénote trois types d'organisation des alternatives(Figure I.3):

- ✓ Dernier rencontré premier exploré (exploration par largeur d'abord)
- ✓ Premier rencontré premier exploré (exploration par profondeur d'abord)
- ✓ Meilleur rencontré premier exploré (recherche heuristique)

C'est bien la dernière stratégie d'organisation qui est intéressante. L'heuristique adoptée apparaît alors dans la façon dont les alternatives sont ordonnées et le critère pris en compte pour réaliser cet ordre. Ce critère est souvent un coût associé à chaque alternative estimant l'espoir porté sur elle en vue d'atteindre une meilleure solution en un temps plus réduit.

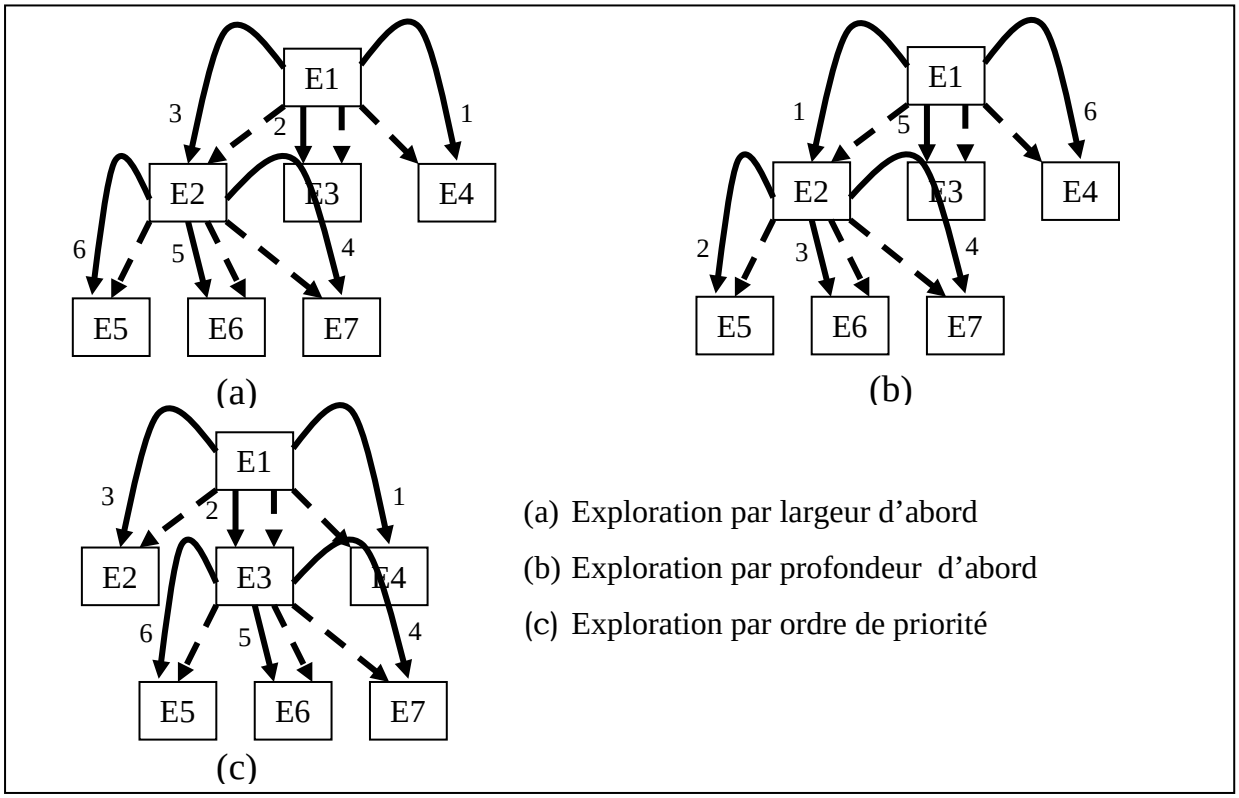


Figure I.3 : Différentes méthodes d'organisation de la recherche

I.8.1 Recherche heuristique :

Une valeur est associée à chaque arc du graphe correspondant au coût estimé de l'application de l'opérateur représenté par l'arc en question. Soit *k**, *g**, *h**, *f** des applications tel que :

- *k**(u, v) = minimum du coût des chemins reliant u à v. *K**(u, v) = +∞ s'il n'existe aucun chemin reliant u à v.

- *g**(u) = *k**(u_0, u) représentant le coût de la recherche ayant aboutit à la solution intermédiaire u.

- *h**(u) = min { *k**(u, v)/ v∈T} où T représente l'ensemble des états terminaux. Ce qui représente l'estimation dans les meilleurs cas (au minimum) du coût induit par la recherche dans le sens du sommet u.

- *f**(u) = *g**(u)+*h**(u).

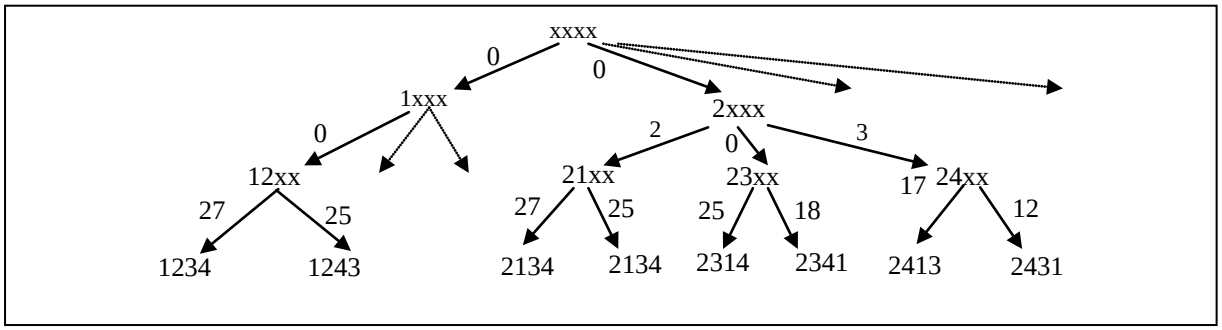


Figure I.4: la fonction k(u, v)

Pour l'exempleI.1, chaque arc du graphe de représentation (Figue I.1) se verra associé un coût égal à l'augmentation induite par le remplacement dans la chaîne d'un *x* par une valeur entière {1,2,3,4}. Par exemple (figure I.4) l'arc reliant le sommet (xxxx) au sommet (1xxx) vaut 0 puisque aucun retard n'est observé suite à la planification de J_1 en première position. L'arc liant (1xxx) à (12xx) vaut aussi 0 pour la même raison. Par contre, l'arc de transition de (12xx) à (1234) vaut 27 puisque suite à cette affectation le job J_3 subit un retard de 9 unités de temps et le job J_4 un temps de retard équivalant à 18.

Alors que la valeur des fonctions *k** et *g** est connue, la valeur de la fonction *h** ne peut quant à elle qu'être approchée par une heuristique *h*. Sauf au niveau des sommets terminaux où *h**(u)=0.

I.8.1.1 Algorithmes de développement complet

Cette famille d'algorithmes se partage le fait de s'appuyer sur une stratégie de développement total de l'état courant.

*I.8.1.1.1 L'algorithme A**

On trouve de nombreuses variantes de cette méthode dans la littérature[Far 87][Ans 97]. La méthode procède en prenant à chaque itération de la recherche l'élément de tête de la pile des alternatives non explorées. Les éléments de la pile étant rangés par ordre croissant de *f* puis

par ordre décroissant de *g* en cas d'égalité de la fonction *f*. Une fois que l'état actuel ait été complètement développé, les états voisins sont rangés dans la pile dans ce même ordre.

Le choix qui consiste à ranger les alternatives par ordre croissant de *f* traduit premièrement le souci d'atteindre la solution optimale le plus tôt possible(valeur de *f* et *h* petite).

Soit la fonction d'estimation minorante *h de h**

$$h(u)=\begin{cases}\min\{k*(u,v)/\, v\in S(u), \text{S(u) est l'ensemble des successeurs de u}\} \text{ si S(u)} \neq \varnothing \\ 0 \text{ si u} \in \text{T}\end{cases} \qquad \text{(I.2)}$$

L'arbre de la figure I.4 serait alors parcourue comme suit:

1. Calculer *f*(1xxx), *f*(2xxx), *f*(3xxx), *f*(4xxx). Etant donnée que *f*(1xxx)= *f*(2xxx)= *f*(3xxx)= *f*(4xxx) = 0 nous choisissons donc aléatoirement l'un des 4 successeurs. Soit le sommet (2xxx) le sommet élu, on empile alors les trois autres sommets dans la pile en respectant l'ordre de *f* et de *g*.

   Pile = {(1xxx),(3xxx),(4xxx)}

2. Calculer *f*(21xx)=27, *f*(23xx)=18, *f*(24xx)=15, c'est donc l'un des trois sommets générés dans l'étape(1) qui sera le prochain sommet parcouru. Soit (1xxx) ce sommet. Les trois nouveaux sommets générés sont alors insérer dans la pile.

   Pile = {(3xxx),(4xxx),(23xx),(23xx),(21xx)}

3. Calculer *f*(12xx)=0, *f*(13xx)=2, g(14xx)=6. Les sommets (12xx),(3xxx),(4xxx) ayant la valeur des fonctions *f* et *g* il faudra donc choisir aléatoirement le prochain sommet exploré, soit (12xx) se sommet. Les sommets (13xx) et (14xx) sont empilés.

   Pile = {(3xxx),(4xxx),(13xx),(14xx),(24xx),(23xx),(21xx)}

4. Calculer *f*(1234)= 27, *f*(1243)=25. Le sommet (3xxx) ayant un coût de la fonction *f* minimal il sera le prochain sommet exploré. Les sommets (1234) et (1243) sont empilés.

   Pile = {(4xxx),(13xx),(14xx),(24xx),(23xx),(1243),(21xx),(1234),(21xx)}

Ce processus est réitéré jusqu'à ce que la pile devienne vide. L'algorithme archive le sommet terminal u* de coût minimal. La procédure ci-dessous décrit le processus de recherche A*.

```
Début
    Pile :=S(u0); u:=u0;
        Trier(Pile, f+g);
        Tant que Pile ≠ ∅ faire
        u :=Tête(Pile);
        if u ∈ T et f(u)<f(u*)
                alors u* := u; {u* désigne la meilleure solution trouvée au cours de la recherche}
        Pile := Pile ∪ S(u);
        Trier(Pile, f+g);
    fait
Fin
```

Figure I.5: Algorithme A*

Une telle approche n'est en réalité efficace que si la valeur *f*(u*), correspondant au coût de la solution optimale est connue d'avance. Sinon la complexité de la méthode est de l'ordre de (*n*!) où *n* désigne le nombre de valeurs que peut prendre une variable(dans notre cas "4").

I.8.1.1.2 Recherche en profondeur ordonnée

Dans ce type d'algorithmes les états voisins du dernier état exploré sont triés par ordre croissant de *f* et décroissant de *g*, puis rangés à la tête de la pile. Cette approche se distingue par une complexité au moins égale à celle de A*.

I.8.1.1.3 Recherche en largeur

Elle consiste à ranger les états au fur et à mesure de leur rencontre, en queue de la pile. Ce qui correspond à développer tous les successeurs de u_0 puis tous les successeurs de ses successeurs,… etc.

I.8.1.1.4 Recherche aveugle

La recherche se fait dans ce cas sans utilisation d'heuristiques autrement dit $h(u)=0$, $\forall$ l'état u. Les états représentés dans la pile sont rangés alors par ordre croissant de *g*, ce qui correspond à une exploration systématique de tout état de coût réduit. Dans le cas où les coûts de transition entre états seraient égaux à 1(coût unitaire) l'algorithme équivaudra à une recherche par largeur.

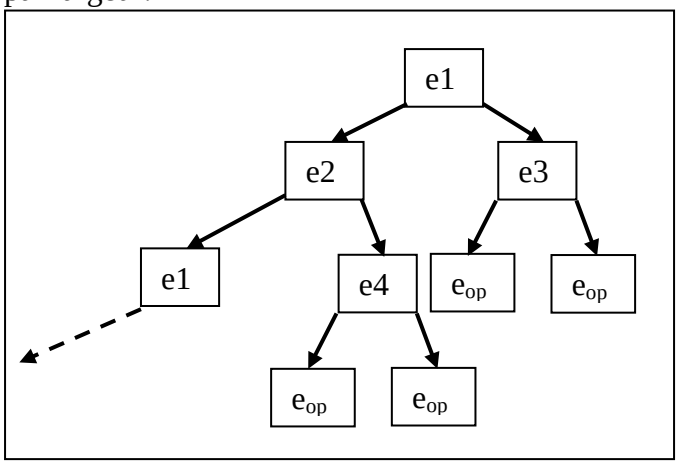


Figure I.6 : Bouclage infini de la recherche

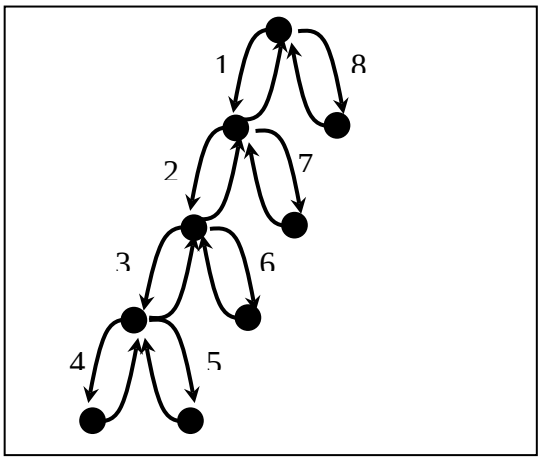


Figure I.7 : Algorithme BSG

I.8.1.1.5 Recherche ordonnée par l'heuristique h

La recherche s'effectue exclusivement en se basant sur l'heuristique *h*. L'information *g* étant éliminée ($g(u)=0$), la recherche ne sera influencée que par l'estimation du coût minimal pour atteindre une solution optimale à partir d'un état donné(recherche par exploitation). L'inconvénient d'une telle approche est le risque de bouclage indéfini de la recherche(voir la figure I.6).

I.8.1.1.6 Recherche avec élagage (Branch & Bound)

Dans de nombreux problèmes, la codification d'un état demande un espace mémoire immense et le nombre d'états engendrés et mémorisés dans la pile est élevé. La recherche par élagage procède alors par écartement d'un certain nombre d'états engendrés en s'appuyant sur une estimation majorante Y du coût $f^*(u_0)$. Ainsi tout état engendré pouvant conduire la recherche à un coût dépassant Y sera écarté. Ceci revient à abandonner l'investigation d'un coin de l'espace chaque fois que l'on estime qu'aucune amélioration ne peut être obtenue. Ceci peut être fait d'une manière très simple. Premièrement, on conserve l'algorithme A* tel qu'il est

actuellement avec la seule retouche suivante: si $f(u) \geq f(u^*)$ alors le sommet u ne sera pas empiler dans la pile puisque de toute manière la branche u ne peut conduire à une solution meilleure que celle déjà trouvée. Le nombre d'alternatives éliminées et donc l'efficacité de la recherche sont étroitement liés à la qualité de l'heuristique *h.* Plus l'heuristique *h* approche mieux la fonction h^* plus le graphe de représentation est parcouru de façon plus optimale. La méthode ainsi décrite s'apparente à la formulation de *Ignall* et *Schrage*(1965) se basant sur la méthode de *Branch & Bound* cité dans [Sim 82] pour la résolution du problème de Flow Shop. La méthode a aussi été appliquée avec succès au problème de Job Shop[Cas 95][Car 89] et d'Open-Shop[Gué 98].

I.8.1.2 Les algorithmes à développement partiel

Dans certains cas, l'algorithme ne peut se permettre d'engendrer la totalité des états accessibles à partir de l'état actuel souvent en raison du nombre important d'alternatives à explorer chaque fois et la difficulté de les construire[Far 85][Sim 82]. En plus du temps important pris pour vérifier si un état ait été déjà visité et pour évaluer l'heuristique. Alors que dans des domaines tel que la robotique ces considérations peuvent être ignorées, dans des domaines tel que l'ordonnancement, elles se présentent comme des facteurs déterminants de l'efficacité de l'algorithme[Far 85].

I.8.1.2.1 L'algorithme Backtrack Search (BSG)

Cette approche se base sur une estimation à priori du coût maximal d'une recherche aboutissante. L'algorithme est similaire alors à une recherche par profondeur à développement partiel (un seul successeur est généré à la fois) excepté le fait qu'une fois le coût d'une recherche dépasse le coût maximal l'algorithme opère un retour arrière afin d'explorer un autre successeur(voir la figure I.7). Ceci revient en quelque sorte à une recherche par élagage dans le cas d'une recherche partielle.

I.8.1.2.2 L'algorithme de recherche redondante

Alors que l'algorithme BSG élimine la contrainte de développement total de l'état actuel. L'algorithme de recherche redondante ne s'occupe plus de savoir si un état a été déjà exploré ou pas. Par conséquent un état peut apparaître plusieurs fois dans la pile et être visité plus d'une fois.

I.8.2 Programmation Linéaire (simplex)

C'est certainement l'approche la plus utilisée dans le domaine d'optimisation linéaire [Sim 82][Le 98][Mun 91]. Le terme linéaire fait référence au fait qu'aussi bien que la fonction objectif que les contraintes sont exprimées sous forme d'expressions linéaires. La formulation standard d'un problème de programmation linéaire s'écrit comme suit:

Maximiser $(Z = C^T x)$, $X_0 = \{x \in R^n : Ax < b$ où $A \in R^m x R^n$ et $b \in R^m\}$ s.c $x \in X$ (I.3)

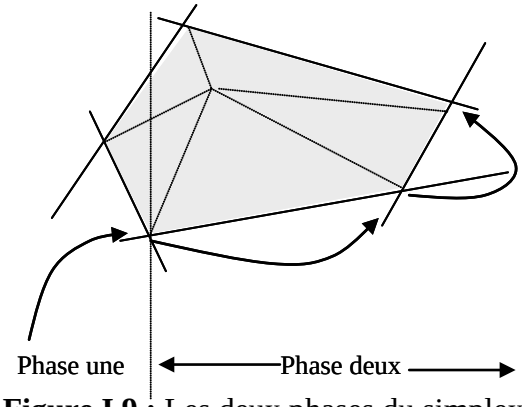


Figure I.9 : Les deux phases du simplex

X représente l'ensemble des solutions admissibles c'est à dire celles qui vérifient l'ensemble des contraintes. Cet ensemble correspond au polyèdre formé par les contraintes à satisfaire. La méthode de simplex procède en deux phases. La première, consiste en la détection d'une solution admissible. Puis, une succession de rotations est réalisées sur l'espace de solutions admissibles, ce qui fait passer l'algorithme d'une extrémité de ce polyèdre à un une autre(figure I.9).

Une part impressionnante dans la littérature est dédiée à la résolution de problèmes d'optimisation par des méthodes s'appuyant sur la programmation linéaire. Des tentatives qui se heurtent souvent à la forme non linéaire des contraintes ou à l'aspect discret de l'espace de recherche, ce qui est souvent le cas pour les problèmes d'ordonnancement. Une révision de la méthode pour tenir compte de ces aspects est donc inéluctable. Un autre inconvénient de la méthode consiste en sa lenteur dès que la taille du problème augmente (nombre de contraintes). Ce qui est dû au parcoure des bords de l'ensemble admissibles sans exploration de l'intérieur de cet ensemble.

Les méthodes proposées travaillent souvent sur des variables binaires, ce qui conduit à une inflation du nombre de variables. L'exemple I.1 peut être modélisé par le système d'équations suivant:

$$\begin{cases} X_{ij} \le 1 \qquad \forall\, i, j \in [1..4] \\ \min \sum_{i=1}^{4} (d_i - l_i) X_{i1} + \sum_{i=1}^{4} \left(\left(\sum_{j=1}^{4} d_j X_{j1} \right) + d_i - l_i \right) X_{i2} + \sum_{i=1}^{4} \left(\left(\sum_{j=1}^{4} \left(\sum_{k=1}^{4} d_k X_{k1} \right) + d_j \right) X_{j2} + d_i - l_i \right) X_{i3} + \\ \sum_{i=1}^{4} \left(\left(\sum_{j=1}^{4} \left(\sum_{k=1}^{4} \left(\sum_{l=1}^{4} d_l X_{l1} \right) + d_k \right) X_{k2} + d_j \right) X_{j3} + d_i - l_i \right) X_{i4} \end{cases} \qquad \text{I.4}$$

Les variables X_{ij} désignent des variables binaires prenant la valeur 1 si le job Ji est assigné à la position j et 0 sinon. Ce qui donne 16 variables binaires X_{ij}.

Pour palier à ces lacunes des variantes proposent l'élargissement de la méthode vers des problèmes non linéaires. D'autres tentent d'exploiter les possibilités de parallèlisation de la méthode [Wie 93].

I.8.3 Programmation dynamique

Initiée par *Bellman* en 1957 la programmation dynamique c'est avérée une méthode très efficace pour la résolution des problèmes d'optimisation combinatoire[Mun 91]. La méthode consiste en la décomposition du problème en une séquence de problèmes concentriques. Chaque problème étant partiellement résolue via la résolution du problème précédent. En revenons à notre exemple I.1, une simple observation concernant la structure de la solution optimale nous fait suggérer que les k premiers jobs (k=1,2,3) doivent constituer des ordonnancements optimaux aux problèmes réduits a ces jobs seulement. En d'autres termes, si l'ordonnancement (J1,J4,J3,J2) est supposé optimal alors les ordonnancements (J1,J4) et (J1,J4,J3) le sont aussi pour le problème réduit aux seuls jobs {J1,J4} et {J1,J3,J4} respectivement. L'algorithme de *Held* et *Karp* [Sim 82] appliqué au problème I.1, consiste alors en les étapes suivantes:

1. Evaluer pour chaque couple de jobs (Ji, Jj) le job à programmer en dernier ceci en estimant le coût de retard subit par chacune des deux possibilités.

2. Evaluer pour chaque triplet de jobs (Ji, Jj, Jk), le job à traiter en dernier. Ceci en estimant le coût de retard observé suite aux trois alternatives possibles et en prenant la meilleur possibilité pour les deux premiers sur la base des résultats de la première étape.

3. Evaluer pour l'ensemble de tous les jobs, le job à traiter en dernier. Ceci toujours en estimant le retard subit suite à chacune des quatre possibilités et prenant la meilleur possibilité pour les trois premiers jobs au regard des résultats de la deuxième étape.

4. Programmer le job trouvé à la troisième étape en dernier. Après quoi, en observe pour les trois autres jobs qui restent, le job à programmer en troisième rang suivant le résultat de l'étape 2. En effectue les même étapes pour les deux jobs non encore planifiés, a savoir choisir parmi les deux jobs celui qu'il faudra planifier en deuxième rang suivant les résultats de la première étape. Le job à programmer en premier sera donc trivialement reconnu.

L'inconvénient d'une telle approche réside dans le grand volume d'espace de stockage qu'il faudra utiliser pour sauvegarder l'ensemble des résultats des étapes antérieures, en plus d'un temps de calcul très élevé. Dans ce cas de l'ordre de 2^{n-1} où n est le nombre de jobs à planifier.

Conclusion

Les méthodes de recherche exacte deviennent rapidement inacceptables dès que la taille du problème augmente. Alors que souvent, l'utilisateur peut largement se satisfaire d'une solution approchée du problème.

I.9 Approche de recherche approximative

L'objectif de telles approches n'est plus de trouver la solution optimale, mais d'approcher celle-ci le plus possible. De telles méthodes ne sont intéressantes qu'en absence d'algorithmes constructives ou de résolution à temps polynomial. La majorité des solutions proposées ci-dessus peuvent être révisés afin de ne garantir que des solutions approchées. Ceci à pour intérêt premier de minimiser considérablement le temps de calcul.

I.9.1 Approches exactes révisées

La méthode de programmation linéaire, peut être appliquée à des problèmes linéaires à variables entières sans aucune transformation du problème. La solution trouvée étant à valeurs réelles, on prend alors la solution à valeur entière la plus proche. Cette méthode porte, pour des raisons très évidentes, le nom très significatif de Programmation linéaire arrondie[Sim 82]. La méthode de Branch and Bound à fait aussi le sujet de réadaptations. Celle dû à *Ashour* nommée Branch and Bound sans retours arrières[Fre 82] consiste en un parcours sans retour arrière du graphe de décomposition. Cette fois-ci, étant à un sommet de l'arbre, on choisit le nœud fils, ayant la plus basse borne(dans notre cas, la valeur de f(u) la moins élevée). L'opération se répète pour le nœud choisi jusqu'à atteinte d'un nœud terminal. La solution ainsi trouvée sera considérée comme la solution finale approchée. Aucun retour arrière n'est opéré afin d'explorer d'autres alternatives. Les résultats expérimentaux obtenus par *Ashour* sont très prometteurs. Sur un ensemble de 100 problèmes la méthode atteint une efficacité moyenne de 95%. Et en aucun cas son efficacité n'est tomber au dessous de 75%.

I.9.2 Technique de recherche Hill-climbing et Greedy

La technique de recherche *Hill-climbing* procède par amélioration successive d'une solution initiale, jusqu'à ce qu'aucune amélioration ne soit alors possible. Le voisinage d'une solution décrit l'ensemble V des solutions directement accessibles depuis la solution actuelle. De ce fait le voisinage d'une solution est complètement définissable d'après le type d'opérateurs de transition choisis. L'application de la méthode à un problème donné exige la réponse à certaines questions au préalable dont :

1. Comment construire la solution initiale?
2. Le type d'opérateurs de transition à utiliser?
3. Comment choisir parmi l'ensemble des solutions avoisinantes la prochaine solution actuelle ? Généralement, on choisit aléatoirement une qui améliore la solution actuelle ou tout simplement, la meilleur.

Pour l'exemple I.1, nous pouvant définir le voisinage d'une solution comme étant l'ensemble des solutions construites par échange de n'importe quel paire de jobs adjacents dans la solution actuelle. Par exemple: $V(1234)=\{2134, 1324, 1243\}$ et comme solution initiale on peut trier les jobs par ordre croissant de leur temps d'achèvement limite. Le principal inconvénient d'une telle approche est le risque de tomber sur un optimum local.

La méthode d'optimisation *Greedy* part de la supposition qu'une solution partielle optimale appartient toujours à une solution globalement optimale. La méthode construit la solution par génération de solutions partielles optimales. Lors de chaque étape, une nouvelle variable intervenant dans la définition du problème est liée à la plus adéquate valeur. Par exemple, dans le problème de voyageur de commerce, nous pouvons choisir de construire la solution en choisissant chaque fois de relier le dernier sommet visité au sommet le plus proche et ainsi de suite.

I.9.3 Les méta-heuristiques

Les méta-heuristiques ont prouvé leur grande habilité à garantir de bonnes solutions pour des problèmes renommés être Np-complet. Les méta-heuristiques représentent une large famille de méthodes de recherche. Les plus connues et utilisées sont les algorithmes évolutionnistes

(AE)[Hoo 96][Ren 94][Pre 95][Gol 89][Gen 96], recherche tabou (RT)[Glo 96][Hoo 97], recuit simulé[Sia 95] et colonies de fourmis[Gra 99]. Ces méthodes se distinguent aussi par leurs applicabilité à une vaste gamme de problèmes d'optimisation combinatoire. Ceci est dû au fait que ces méthodes travaillent sur une représentation interne du problème plutôt que sur le problème lui-même. Ces approches sont munies de mécanismes de diversification leur garantissant plus ou moins d'échapper aux optimums locaux.

I.10 Problèmes de satisfaction de contraintes (PSC)

D'une manière générale, un problème d'optimisation ne se limite pas à trouver une solution qui maximise ou minimise un certain nombre de paramètres exprimés sous forme de fonctions objectifs. La solution doit aussi satisfaire un ensemble de contraintes. Prenons l'exemple du problème de placement de processus sur des processeurs[Mun 91]. Le placement ainsi fait doit prendre en charge les contraintes temporelles imposées par la nature temps réel de l'application ou imposées par la machine(la limitation de la taille mémoire). Il doit aussi minimiser la longueur des chemins de communication, la charge de travail des processeurs et le temps de réponse total de l'application.

De ce point de vu un PSC est un cas particulier de problèmes de recherche. L'ensemble des méthodes développées ci-dessus convient de ce fait à de tels problèmes. L'heuristique *h* peut être vue dans ce cas comme le nombre de contraintes qui restent à vérifier à partir d'un état donné. D'autres techniques[Min 93] plus spécifiques à ce genre de problèmes ont été adoptées.

I.10.1 Recherche avec retours arrières (Backtrack search)

Ces algorithmes procèdent par construction itérative de la solution. Le problème se présente comme suit: un ensemble (v_1, v_2, …, v_n) de variables et un ensemble de contraintes (c_1, c_2, …, c_m) tel que c_i = ((satisfait v_i *val*) ssi condition(v_{i1}, v_{i2},…, v_{ik})). La formulation de la contrainte c_i signifie que l'assignation de la valeur *val* à la variable v_i est permise si et seulement si la condition mettant en jeux les variables v_{i1}, v_{i2},…, v_{ik} est vérifiée.

Au début de l'algorithme, les variables ne sont liées à aucune valeur. L'algorithme opère alors en choisissant à chaque itération de la recherche une variable non encore liée, puis une valeur à lui affecter. Dans le cas où aucune valeur ne serait permise l'algorithme effectue un retour arrière afin de choisir une autre valeur pour la variable précédemment choisie. Selon la stratégie de choix d'une variable plusieurs méthodes sont possibles[Min 93].

- ✓ ***Priorité aux variables les plus contraintes***: Dans ce cas en préfère choisir la variable dont les valeurs possibles sont restreintes. Généralement ce sont les variables dont la contrainte se rapportant met plus de variables en jeux.

- ✓ ***Priorité aux variables les plus contraignantes***: Dans ce cas on privilégie la variable qui apparaît le plus fréquemment comme membre des parties conditions des contraintes.

- ✓ ***Priorité aux variables les moins contraignantes***: Cette approche avantage les variables qui apparaissent le plus rarement dans les parties conditions des contraintes.

- ✓ ***Recherche dictée par les dépendances***: Elle se base sur l'interdépendance entre variables pour éliminer certaines alternatives. Dans [Fri 94] l'interdépendance entre les

variables est utilisée afin de déterminer l'ordre d'instantiation des variables. L'algorithme dû à *Freuder* [Fre 82] déterminer l'ordre d'instantiation garantissant un nombre minimum de retours arrière (Backtracking).

I.10.2 Recherche par propagation

Le principe de base de cette technique est d'arriver à réduire le nombre de valeurs candidates au test une fois la variable est choisie[Cas 95]. Chaque fois qu'une valeur est assignée à une variable un filtre est appliqué à toutes les variables relatées par des contraintes mettant en cause la variable en considération. Les filtres ainsi générés définissent les valeurs possibles des variables non encore liées.

I.10.3 Recherche par retardement d'évaluation

La recherche par retardement d'évaluation[Zwe 89] se base sur le principe d'évaluation à la demande. Ce qui signifie qu'une partie des structures de données n'est générée qu'une fois le besoin s'en fait sentir. De ce fait, le filtre d'une variable n'est calculé qu'une fois que celle-ci aie été choisie.

I.11 Problèmes d'ordonnancement

L'ordonnancement constitue un domaine d'application de la Recherche Opérationnelle. *Esquirol* et *Lopez* le définissent par : "*Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, …) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises. Une tâche est une entité élémentaire de travail localisée dans le temps par une date de début ou de fin, dont la réalisation nécessite une durée et qui consomme des moyens.*"

Selon cette définition la tâche se trouve être l'entité élémentaire à ordonnancer. Un travail(*Job*) désigne une succession de tâches. On parle alors d'ordonnancements des jobs. Lorsque chaque job est composé d'une seule tâche on parle alors de problème d'ordonnancement *mono-tâche*. Les tâches composants le même job peuvent être sujets à des contraintes de précédence définissant la gamme d'usinage du job. Les critères à optimiser sont de l'ordre de deux[T'ki 99]: Les critères relevant du temps et les critères relevant du coût.

1. Critères temporelles : sont des estimations faisants intervenir le temps d'achèvement final de l'ordonnancement, les retards enregistrés dans l'accomplissement des jobs, …

2. Critères de coût : sont des mesures faisants intervenir le coût d'utilisation des ressources, le coût induit par les délais d'attente des jobs avant et après leurs traitements, …

Pour résoudre un problème d'ordonnancement on est souvent amené à résoudre également un problème d'affectation, lorsqu'il s'agit en plus de déterminer la ou les ressources à allouer aux jobs.

I.11.1 Domaines d'application

Les champs d'application des problèmes d'ordonnancement sont très variés[T'ki 99]. On les retrouve aussi bien dans le domaine de la production de biens (problèmes d'atelier) que dans la gestion de projets ou les applications liées aux systèmes informatiques.

I.11.1.1 Problèmes d'atelier

Tout atelier planifie sa production à court ou long terme[Ste 99]. La limite des moyens de production entraîne la nécessité d'ordonnancer ce processus. Dans un atelier où la production est régit par les commandes des clients, la phase de planification consiste à répartir les commandes par périodes de temps. Le processus d'ordonnancement consiste alors à organiser le traitement des commandes par les ressources pour chaque période. Les ressources de production s'apparente généralement dans se cas à des machines.

I.11.1.2 Système informatique

Dans un système multiprocesseurs, l'exécution optimale des applications rend nécessaire le placement ingénieux des divers processus composant les programmes sur les processeurs. Pour cela plusieurs facteurs peuvent prévaloir [Mun 91] tel que la longueur des chemins de communication, l'équilibrage de la charge de travail des différents processeurs et des liens de communication. L'ordonnancement doit aussi satisfaire aux contraintes temporelles imposées par le programme comme dans les applications temps réels, ou dues aux limites matérielles telle que la taille mémoire d'un processeur.

I.11.1.3 Gestion de projet

Il s'agit dans ce cas[Lin 99] de planifier dans le temps la réalisation d'un certain nombre de tâches parallèles requérant un ensemble de ressources de quantités variables. Les problèmes d'emploi du temps dans les établissements éducatifs, la planification de la construction d'ouvrages (maison, barrage,…) font partie de ce type de problèmes.

I.11.2 Classification des problèmes d'ordonnancement

L'étude réalisée par *T'Kindt* et *Billaut* [Bil 99] sur les problèmes d'ordonnancement font surgir la typologie (figure I.10) suivante:

1. Problème à ressource unique : une seule ressource où chaque job est assimilé à une seule tâche.
2. Problème de Flow Shop : les jobs possèdent la même gamme de production sur les ressources.
3. Problème de Job Shop : les jobs possèdent des gammes de production différentes.
4. Problème de Open-Shop : les jobs possèdent des gammes de production différentes pouvant même être incomplètes(un job peut ne pas avoir besoin de l'ensemble de toutes les ressources).

Le problème d'affectation vient se rajouter au problème d'ordonnancement quand plusieurs ressources se trouvent aptes à traiter une certaine tâche. On distingue dans ce cas deux situations possibles.

1. Une ressource appartient à un seul pool. Donc il n'y a pas d'intersection entre les différents pools.

2. Une ressource appartient à plusieurs pools à la fois. Les intersections entre pools n'étant pas vides cela donne lieu à un problème d'affectation généralisé. Un exemple de cette situation est le cas d'un établissement éducatif composé de salles de cours de capacités variables. La classification des salles en pools est réalisée sur la base du type de cours(tâches) qu'elle peut abriter.

   Soit *Pool1* l'ensemble des salles pouvant abriter des cours regroupant un nombre d'étudiants allant jusqu'à 100. *Pool2* désigne l'ensemble des salles pouvant abriter des cours de 50 étudiants…

   Il semble clair que l'ensemble *Pool1* est complètement inclus dans *Pool2(Pool1⊂Pool2*).

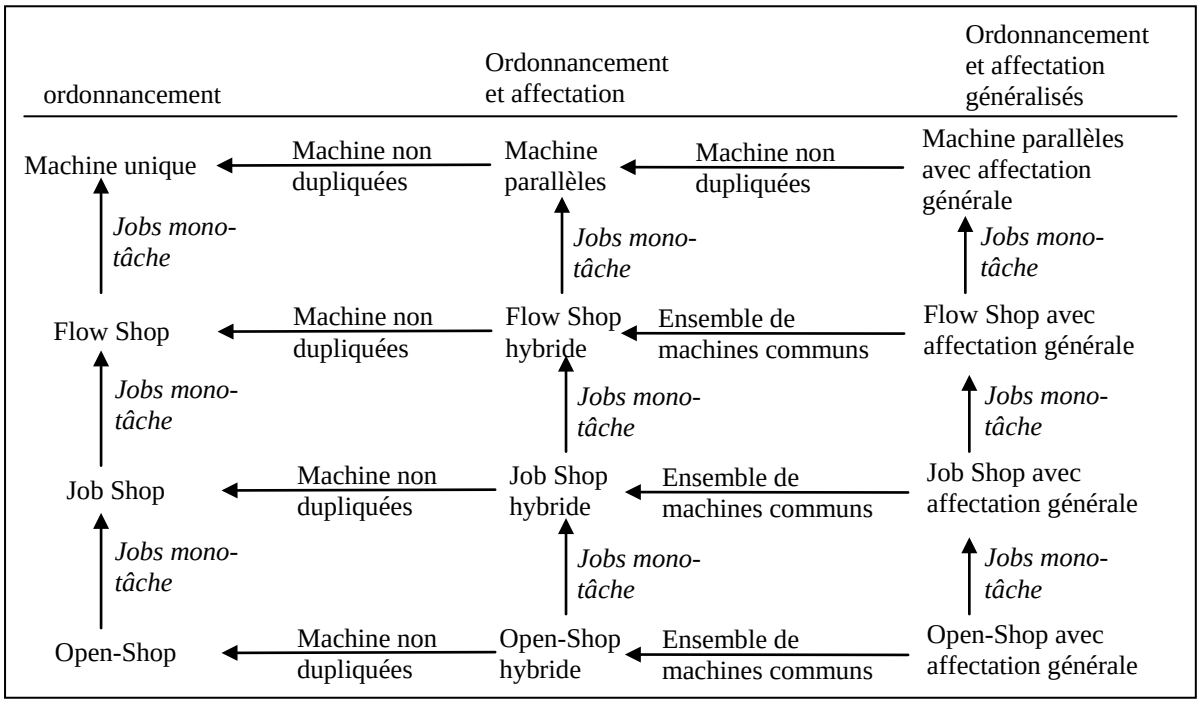


Figure I.10 : Typologie des problèmes d'ordonnancement

La classification ainsi exposée s'appuie sur la nature des ressources et des tâches. Cependant, d'autres critères de classification peuvent être retenus[Bil 99]:

1. L'aspect déterministe(certain) ou stochastique(probabiliste) des données du problème.
2. L'aspect unitaire ou répétitif des tâches qui composent le problème.
3. L'aspect statique ou dynamique des données du problème. Dans certaines situations les tâches continuent d'affluer alors que l'ordonnancement est à pied d'œuvre. Il faut donc être en mesure de remettre en cause en temps réel l'ordonnancement en cour (voir [Ger 66] pour une étude du problème de Job Shop dynamique).

I.11.3 Quelques problèmes d'ordonnancement

Nous proposons maintenant de décrire quelques problèmes d'ordonnancement référencés dans le reste de cette thèse.

I.11.3.1 Problème de Flow Shop

Le problème de Flow Shop[Mar 98] a fait l'objet de beaucoup d'attention depuis 1954, année où *S. M. Johnson* proposait son algorithme de résolution des problèmes à deux machines. Les méthodes adoptées pour sa résolution varient entre méthodes exactes tel que le Branch & Bound [Bra 91], heuristiques de recherche [Raj 91][Bil 99][Gup 69] et méta-heuristiques [Dia 91][Mur 94][Ben 98][Now 96].

Le problème de Flow Shop se présente comme un ensemble de N jobs à ordonner sur M machines. Les machines sont des ressources critiques dans le sens où une machine ne peut être affectée à deux jobs simultanément. Tout job est composé de M tâches consécutives(non parallèles), $J_i=\{t_{i1}, t_{i2}, \ldots, t_{iM}\}$. La tâche t_{ij} représente la $j^{ème}$ tâche du job J_i requérant la machine m_j. Par conséquent, tous les jobs ont la même séquence d'usinage sur les machines. A chaque tâche t_{ij} est associée une durée d'exécution d_{ij}, et chaque job J_i est limité par une date l_i avant laquelle il doit obligatoirement s'achever.

L'ordonnancement des tâches sur les différentes machines doit s'effectuer de façon à optimiser certains critères dit "*réguliers*". Ces critères varient selon la spécificité du problème traité, et consistent généralement à minimiser l'une des fonctions suivantes[Bil 99]:

- C_{max} **:** Temps de terminaison final, correspondant à la date d'achèvement de la dernière tâche planifiée.
- $\overline{C}$ **:** Moyenne des temps de réponses des jobs.
- T_{max} **:** Maximum des retards subits.
- T **:** Somme des retards subits.
- T_{nb} **:** Nombre de jobs ayant subits un retard par rapport à leurs dates d'achèvement au plus tard.
- F_{max} **:** Maximum des Flux des jobs. Le flux désigne la différence entre la date de disponibilité d'un job et sa date de terminaison.
- $\overline{F}$ **:** Moyenne des flux des jobs.

Le terme "*régulier*" désigne le fait qu'un retard dans la planification d'une tâche ne peut qu'induire une dégradation de la solution suivant ce critère. Formellement ceci tend à dire que si C et C' sont deux vecteurs définies comme suit : $C = (C_{11}, C_{12},...,C_{21}, C_{22},..., C_{N1}, C_{N2},... C_{NM})$ et $C'=(C'_{11}, C'_{12},..., C'_{21}, C'_{22},..., C'_{N1},C'_{N2},...C'_{NM})$ où les C_{ij} et C'_{ij} désignent les dates d'accomplissement de la tâche t_{ij} et si F est un critère régulier, alors le fait que $C \leq C'$ implique que $F(C) \leq F(C')$. Ceci nous permettre par la suite de ne considérer que les ordonnancements semi-actifs, c'est à dire, où chaque tâche est programmée le plus tôt possible[Bil 99][Sim 82].

I.11.3.1.1 Problème de Flow Shop à deux machines

L'ordonnancement optimal est dans ce cas généré suivant la régle de *Johnson*[Sim 82][Gen 96]. La règle stipule qu'un Job J_i précédera le Job J_j si $\min\{d_{i1}, d_{j2}\} \leq \min\{d_{i2}, d_{j1}\}$. L'algorithme de *Johnson* est constitué des quatre étapes suivantes:

1. Soit $U = \{ J_i / d_{i1} < d_{i2} \}$ et $V = \{ J_i / d_{i1} \geq d_{i2} \}$
2. Trier les jobs dans U par ordre croissant de $d_{i1.}$
3. Trier les jobs dans V par ordre décroissant de $d_{i2.}$
4. L'ordonnancement optimal est obtenu en plaçant l'ensemble U ordonné en premier suivie de l'ensemble V.

I.11.3.1.2 Heuristique pour le problème de Flow Shop général

Beaucoup d'heuristiques ont été proposées [Gen 96]assurant une bonne est rapide solution. Généralement, le critère à optimiser est C_{max}. Nous citerons brièvement celles qui sont les plus connues.

I.11.3.1.2.1 Heuristiques de Palmer et Gupta

L'idée de *Palmer* est d'associer à chaque job un indice qui va mesurer le taux avec lequel s'accroît les temps de traitement des tâches de machine en machine. Les jobs dont le temps de traitement tend à s'accroître le long des machines reçoivent une plus grande priorité et sont donc planifiés en premiers. L'heuristique de *Gupta* suit le même raisonnement, seule la formule servant à calculer l'indice d'accroissement des jobs les différencie[Gen 96].

I.11.3.1.2.2 Heuristique CDS

Dû à *Campbell*, *Duderk* et *Smith*(CDS)[Gen 96], l'heuristique présente les deux points forts suivants :

- La méthode utilise la règle de *Johnson* de façon heuristique
- Conduit généralement à la construction de plusieurs ordonnancements dont le meilleur sera retenu.

L'algorithme génère M-1 problèmes à deux machines par agrégation systématique des M machines en deux groupes. Puis applique l'algorithme de *Johnson* sur les problèmes ainsi produits. Lors de la première phase, seules les machines m_1 et m_M sont considérées. Lors de la deuxième phase, deux machines artificielles seront considérées. La machine 1 correspond à l'agrégation des machines $\{m_1,m_2\}$et la machine 2 à l'agrégation des machine $\{m_{M-1},m_M\}$. Lors de la $k^{ème}$ phase, deux machines artificielles seront considérées. La machine 1 correspond à l'agrégation des machines $\{m_1,m_{2,\ldots,}m_k\}$et la machine 2 à l'agrégation des machine $\{m_{M-k+1},\ldots,m_M\}$. L'agrégation de plusieurs machines en une seule, correspond à traiter les différentes tâches sur chaque machine, comme une seule tâche de temps de traitement égal à la somme des temps de traitement des tâches se produisant sur la même machine.

$$d'_{i1} = \sum_{j=1}^{k} d_{ij} \quad \text{et} \quad d'_{i2} = \sum_{j=1}^{k} d_{i,m-j+1} \tag{I.5}$$

I.11.3.1.2.3 Heuristique NEH

Cette algorithme constructif dû à *Nawaz, Enscore* et *Ham*(NEH)[Naw 83][Kou 98][Ben 98] s'appuie sur la supposition qu'un job dont le temps total de traitement est le plus important doit recevoir la priorité la plus élevée. La construction de l'ordonnancement final passe par les trois étapes suivantes:

1. Ordonner les *N* jobs par ordre décroissant de leur temps de traitement total sur les machines.
2. Prendre les deux premiers jobs est les ordonnancer de la manière la plus optimale afin de générer un ordonnancement partiel restreint à seulement ces deux jobs.
3. Pour k de 3 jusqu'à *N* faire: insérer le k^{ème} job à la meilleure position (parmi les k possibles) dans l'ordonnancement formé des k-1 premiers jobs planifiés et générer ainsi un nouveau ordonnancement partiel.

I.11.3.1.2.4 Branch and Bound

L'application de la méthode au problème de flow shop hybride a été réalisée par *Brah* et *Hunsucker* [Bra 91]. Lors de chaque itération, l'algorithme choisit une nouvelle tâche planifiable (dont le prédécesseur est déjà planifié) afin de la rajouter à l'ordonnancement partiel *u*. L'arc reliant les deux ordonnancements partiels est de poids égal à $C_{\max}(v)$- $C_{\max}$(u). l'heuristique *h* est calculer suivant la formule suivante:

$$h(u) = \max_{i=[1..N]} \left\{ C(i,u) + \sum_{t_{ij}} d_{ij}/t_{ij} \text{ n'est pas encore planifié dans u} \right\} - C_{\max}(u) \qquad \text{(I.6)}$$

Cette formule correspond à une estimation minorante du *makespan* du meilleur ordonnancement issu de *u*. $C(i, u)$ désigne le temps d'accomplissement du job *i* dans l'ordonnancement partiel *u.*

I.11.3.2 Problème de Job Shop

Ce cas se présente comme une extension du problème de Flow Shop. Les jobs contrairement au problème de Flow Shop, n'ont pas nécessairement la même gamme d'usinage sur les machines. Les travaux réalisés sur le problème de Job Shop abondent dans la littérature[Car 89][Cas 95][Ger 66]. Etant donnée la complexité du problème il paraît donc naturel de s'intéresser aux méthodes approchées produisant des ordonnancements acceptables plutôt qu'aux méthodes exactes tel que le Branch & Bound [Car 89]. *Mitsuo* et *Cheng*[Gen 96] classifie les heuristiques élaborées pour le problème de Job Shop en deux grandes classes.

1. Heuristique à une seule passe: opère par construction complète d'une seule solution en planifiant à chaque fois une tâche(Algorithme Greedy). L'ordre de planification des tâches se base sur une règle de distribution de priorités.
2. Heuristique multi-passe: l'heuristique est lancée de façon répétitive afin d'obtenir des ordonnancements meilleurs.

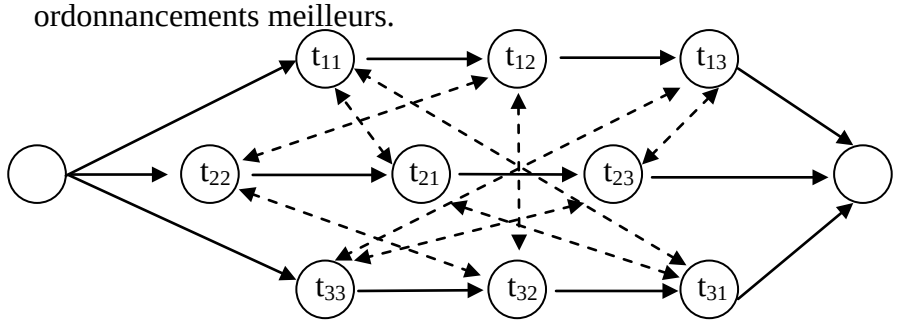


Figure I.11: Graphe de présentation disjonctif d'un problème de Job Shop
⟶ Contrainte de précédence ⟷ Conflit sur machine

I.11.3.2.1 Heuristique de distribution des priorités

C'est le genre d'heuristiques les plus utilisées pour la résolution des problèmes d'ordonnancement du fait de la simplicité de leur implémentation et la rapidité de leur exécution. L'algorithme de *Giffler* et *Thompson* [Sim 82][Gen 96]basé sur la génération d'ordonnancement actif est certainement le principe de base de toute heuristique de distribution de priorités. La procédure de construction de l'ordonnancement travaille sur un ensemble de tâches planifiables à chaque phase. Une tâche planifiable correspond à une tâche non encore ordonnancée et dont l'ensemble des prédécesseurs immédiats le sont(tâche précédente dans le job et tâche précédente sur la machine). Le nombre de phases correspond à *N*M*. A chaque phase une tâche est sélectionnée pour être rajoutée à l'ordonnancement partiel. Soit les notations suivantes:

PS_t = l'ordonnancement partiel composé de t tâches.
S_t = l'ensemble des tâches planifiables à la phase t, en regard à l'ordonnancement PS_t.
σ_i = la date de début au plus tôt de la tâche $i \in S_t$.
ϕ_i = la date de terminaison au plus tard de la tâche $i \in S_t$.

la procédure de génération de l'ordonnancement actif est la suivante:

1. t=0, $PS_t = \varnothing$, S_t est l'ensemble de toutes les tâches n'ayant pas de prédécésseurs.
2. Déterminer $\phi_t^* = \min_{i \in S_t} \{\phi_i\}$ et la machine m* qui traite la tâche ϕ_t^* .
3. Pour toute tâche $i \in S_t$ requérant m* dont $\sigma_i < \phi_t^*$, *calculer l'indice de priorité en se basant sur une règle de priorité spécifique*, choisir la tâche avec l'indice de priorité le plus élevé et l'ajouter à l'ordonnancement PS_t ce qui génére un nouveau ordonnancement partiel PS_t.
4. Retirer la tâche i de l'ensemble S_t puis rajouter le successeur directe de i à S_t pour former S_{t+1} et finalement incrémenter t et passer à l'étape 2.

I.11.3.2.2 Heuristique de distribution aléatoire

L'idée principale est d'user de plusieurs règles de distribution de priorités(voir [Gen 96] pour avoir une liste détaillée de ces règles). L'algorithme de *Giffler* et *Thompson* est lancé plusieurs fois(multi-passes). La règle utilisée lors de la troisième étape est choisit de façon aléatoire, se qui donne lieu à des solutions différentes, la meilleure étant choisie.

I.11.3.2.3 Heuristique basée sur les goulots

C'est actuellement l'une des heuristiques les plus efficaces[Gen 96], basée sur l'ordonnancement des machines une à une. Le problème se ramène alors à des problèmes de Job Shop à une machine avec dates de disponibilité des tâches. Chaque fois, une machine est ordonnancée et toutes les séquences antérieurement établies sont localement réoptimisées. Le choix de la prochaine machine à ordonnancer se base sur la mesure du degrés d'étouffement de la machine(la difficulté qu'incombe d'ordonnancer la machine tout en satisfaisant l'ensemble des contraintes). L'identification de la machine qui étouffe ainsi que la procédure de réoptimisation locale sont toutes les deux basées sur une relaxation répétitive du problème original en problème mono-machine.

I.11.3.3 Problème d'emploi du temps

Le problème d'emploi du temps[Mab 97] se présente comme un ensemble de cours à planifier sur un créneau horaire définissant les périodes ouvrables. Le problème revient à l'assignation des périodes aux cours de façon à prévenir les conflits sur les enseignants, sur les étudiants et sur les salles. Le problème regain de complexité quand d'autres contraintes viennent se rajouter telles que :

1. Indisponibilité ou préférence des enseignants pour certaines périodes d'enseignement.
2. Répartition des cours d'un étudiant ayant trait au même module sur les jours.
3. La non uniformité des durées de cours (nombre de périodes consécutives exigées par chaque cours).

Les objectifs à optimiser varient aussi selon le type et la spécificité de l'établissement.

- Minimiser les déplacements des étudiants dans l'école.
- Minimiser les temps creux.
- Répartir les cours équitablement sur les jours ou au contraire les regrouper afin de procurer des journées libres aux étudiants.

La taille souvent immense de tels problèmes, le nombre et la nature des contraintes qu'ils englobent rend le problème d'emploi du temps d'une complexité draconienne. Ce fait rend l'utilisation de méthodes exactes peu intéressant pour résoudre le problème de façon pratique.

I.11.3.3.1 Approches de résolution basée sur la théorie des graphes

Ces approches partagent en commun le fait de s'appuyer sur les notions de la théorie des graphes telles que : le nombre chromatique, l'indice chromatique ou le nombre de stabilité. Il s'agit alors de modéliser le problème réel en un problème de coloration ou de recherche de sous graphes stables[Mab 97].

I.11.3.3.1.1 Modélisation par coloration de graphe

On suppose donnée un ensemble d'enseignements E définit par des triplets (e, p, l) désignant pour chaque enseignement e le local l où il se déroule et l'enseignant p qui l'assure. Un graphe G non orienté est construit où les sommets désignent les enseignements de l'ensemble E. Les arêtes représentent les contraintes de non simultanéité(les enseignements ne pouvant pas se dérouler en même temps). Autrement dit, deux enseignements (e, l, p) et (e', l', p') sont reliés par une arête si $l = l'$ ou $p = p'$ ou que e et e' concerne un même étudiant.

La technique de résolution consiste à colorier les sommets de G par un nombre minimum de couleurs, de sorte que deux sommets adjacents ne soient pas de même couleurs. Le nombre chromatique (nombre de couleurs) ainsi obtenu, correspond au minimum de périodes nécessaires pour programmer tous les enseignements. Chaque couleur correspondra alors à une période donnée(par exemple 8H à 9H).

I.11.3.3.1.2 Modélisation par ensembles de stables

Cette approche part de la supposition que les locaux sont de même type. Un graphe G est alors construit comme précédemment où les sommets représentent les enseignements et les arêtes les contraintes de non simultanéité. Il s'agit dans ce cas, de dégager un partitionnement des sommets du graphe en de sous graphes stables de cardinalités inférieures à L. L désigne le nombre de locaux disponibles. Chaque sous graphe stable correspond à un sous ensemble d'enseignements qui doivent être planifiés à la même période.

Les approches de résolution se basant sur la théorie de graphe soufrent cependant de plusieurs lacunes, la plus importante réside dans l'impossibilité de modéliser l'ensemble de toutes les contraintes.

I.11.3.3.2 Approche de programmation linéaire

Le modèle mathématique pour le problème d'emploi du temps peut être formulé par le système suivant [Mab 97] :

$$Max\ Z = \sum_{e\in E}\sum_{p\in P}\sum_{l\in L}\sum_{t\in T} C(e,p,l,t)\, f(e,p,l,t)$$

sachant que :

$$\begin{aligned}
&(1).\sum_{t\in T}\sum_{e\in E} f(e,p,l,t) \le 1 \quad \forall l \in L, \forall p \in P\\
&(2).\sum_{l\in L}\sum_{e\in E} f(e,p,l,t) \le 1 \quad \forall t \in T, \forall p \in P\\
&(3).\sum_{l\in L}\sum_{p\in P}\sum_{t\in T} f(e,p,l,t) = 1 \quad \forall e \in E\\
&(4).\sum_{e\in S_c}\sum_{l\in L} f(e,p,l,t) \le 1 \quad \forall p \in P, \forall c\\
&(5).\sum_{e\in E_k}\sum_{l\in L}\sum_{t\in T} f(e,p,l,t) \le N_k(p) \quad \forall p \in P, \forall k\\
&(6).\sum_{e\in S_c}\sum_{l\in L}\sum_{p\in H}\sum_{t\in T} f(e,p,l,t) \le N(c) \quad \forall c, \forall H\\
&(7).\ f(e,p,l,t) - f(e',p',l',t') \le 0\\
&(8).\ f(e,p,l,t) + f(e',p',l',t') \le 1\\
&(9).\ f(e,p,l,t) = 0 \text{ ou } 1 \quad \forall e \in E, \forall p \in P, \forall l \in L
\end{aligned} \qquad \text{(I.7)}$$

f est une fonction de $E\times P\times L\times T$ dans {0,1}, telle que $f(e, p, l, t)$ = "1" si l'unité d'enseignement e est programmé pendant la période p dans le local l et elle est assurée par l'enseignant t et "0" sinon. $N_k(p)$ est le nombre de locaux de type k disponibles à la période p, H désigne les périodes appartenantes à la même journée, $N(c)$ désigne la charge journalière maximale d'un étudiant c, S_c désigne l'ensemble des enseignements concernant un même étudiant(ou une classe) c.

Ces contraintes signifient que toutes les unités d'enseignements doivent être enseignées dans la semaine(3), de telle façon qu'un étudiant(4), un local (1) ou un enseignant (2), ne soit affecté pendant une même période qu'à une seule unité d'enseignement au plus. La contrainte (5) se réfère à la disponibilité des locaux. La contrainte (6) est une contrainte de non surcharge des journées d'un étudiant c. La contrainte (7) peut soit modéliser une contrainte de succession si $p' = p + 1$ ou une contrainte de simultanéité si $p = p'$. La contrainte (8) sert à modéliser des contraintes d'exclusion, par exemple empêcher la succession ou la simultanéité de deux enseignements donnés.

I.11.3.3.3 Approche heuristique

Chergui[Che 91] propose une approche par décomposition pour résoudre le problème d'emploi du temps. Dans un premier temps, les classes(groupes d'étudiants) sont triées par ordre décroissant du nombre d'enseignements auxquels elles participent. L'algorithme effectue

alors la planification des enseignements à commencer par les cours associés aux classes surchargées. Cette programmation est réalisée toute en évitant les conflits. Après cela l'algorithme entame la phase d'affectation des locaux aux enseignements.

I.11.3.4 Problème de routage de Véhicules

Le problème de routage de véhicules avec fenêtres horaires[Lou 99] consiste en un ensemble de véhicules dont la capacité et le temps de service sont limités. Ces véhicules sont sensés desservir un ensemble de clients répartis géographiquement. Chaque client C_i est caractérisé par une quantité q_i de biens commandés à livrer obligatoirement pendant un intervalle de temps $[A_i, B_i]$. Il s'agit donc d'associer à chaque véhicule un trajet desservant un sous-ensemble de clients de façon à assurer la satisfaction de l'ensemble de tous les clients dans les délais(conformément aux fenêtres horaires de livraison). Généralement, chaque véhicule est caractérisé par une capacité maximale Q_i et une distance maximale du trajet T_i.

Les objectifs à optimiser dépendent des besoins du décideur.

- Minimiser la somme des trajets parcourus par les véhicules.
- Minimiser le nombre de véhicules utilisés afin d'en garder le maximum au dépôt pour le dépannage.
- Répartir la charge de travail équitablement sur l'ensemble des véhicules.

L'intérêt que porte le problème de point de vue académique (puisque il se présente comme une extension du problème de voyageur de commerce) et pratique à fait émerger une grande classe de méthodes de résolution.

I.11.3.4.1 Programmation linéaire

Kulkarni et *Bhave* [Kul 85] proposèrent une modélisation du problème sous forme linéaire à variable binaire X_{ij}.

$$X_{ij} = \begin{cases} 1 & \text{si un véhicule relie les clients i et j directement} \\ 0 & \text{sinon} \end{cases}$$

Sujet aux contraintes suivantes :

$$\begin{cases} \sum_{j=1}^{N} X_{ij} = 1 \text{ pour i} = 1..\text{N} - 1 \\ \sum_{i=1}^{N} X_{ij} = 1 \text{ pour j} = 1..N - 1 \\ \sum_{i=1}^{N} X_{iN} \le V \\ \sum_{i=1}^{N} X_{Ni} \le V \\ \sum_{i \in L_m \cup \{N\}} \sum_{j \in L_m \cup \{N\}} d_{ij} X_{ij} \le \text{T}_\text{m} \quad \forall \text{ v}_\text{m} \in \text{V}, \text{L}_\text{m} \text{ ensemble des clients visités par v}_\text{m} \\ \sum_{i \in L_{mi} \cup \{N\}} \sum_{j \in L_m \cup \{N\}} q_i X_{ij} \le Q_m \quad \forall \text{ v}_\text{m} \in \text{V}, \end{cases} \tag{I.8}$$

d_{ij} désigne la distance séparant les clients *i* et *j*.

I.11.3.4.2 Approches par modélisation en problème de voyageur de commerce

Le PRV est, de façon quasiment évidente, une extension du problème de voyageur de commerce(PVC) classique. La similitude qui lie ces deux problèmes à pour conséquence de justifier l'utilisation des méthodes développées pour le PVC pour la résolution du PRV [Chr 76]. En effet, le dépôt x_0 est remplacé par *V* dépôts artificiels $x_0^1, x_0^2, ..., x_0^V$, tous à la même localisation(distance(x_0^i, x_0^j) = 0, $\forall$ *i*, *j*$\in$[1..*V*]). Les *V* trajets réalisés par les différents véhicules démarrant de x_0 peuvent être vus comme un seul tour débutant depuis x_0^1 visitant quelques clients et retournant à x_0^2 puis visitant d'autres clients et revenant à x_0^3, et ainsi de suite jusqu'à ce que le véhicule retourne finalement à x_0^1. Cette approche ne peut cependant pas être étendue pour le traitement des problème de routage avec fenêtres d'asservissement.

I.11.3.4.3 Modélisation par un problème de partition d'ensembles

Soit la matrice *G* où les lignes correspondent aux clients et les colonnes aux différents trajets possibles. *G*[*i*, *j*] est égal à "1" si le client x_i est servi par le trajet S_j et "0" sinon. Le problème de routage de véhicules revient alors à déterminer le sous ensemble de colonnes(trajets) qui minimise la distance totale des trajets et où chaque ligne a une entrée égale à "1" sous exactement une seule colonne du sous ensemble choisi. Ce problème connu sous l'appellation de partition d'ensembles peut être résolu de façon optimale pour des nombres de colonnes raisonnables.

Dans[Chr 76] ont trouve la description d'autres méthodes de résolution telles que le Branch & Bound, Génération de colonnes, …

I.12 Conclusion

Les problèmes d'optimisation combinatoires et plus spécialement ceux d'ordonnancement se présentent comme étant intraitables par l'algorithmique classique. La raison est que la topologie de tel problèmes est imprévisible. Souvent le paysage est d'un degré de complexité élevé, présentant plusieurs optimums locaux et se distingue par un espace de recherche discret et non euclidien. L'autre source de difficulté découle du volume immense d'informations qui composent le problème (données et contraintes). Ceci fait que les algorithmes constructifs ne peuvent être appliqués que pour des cas précis du problème traité.

Les méthodes proposées, procèdent dans la majorité par construction ou amélioration itérative d'une ou plusieurs solutions. Ces méthodes varient entre heuristiques de recherche en intelligence artificielle, Branch and Bound, programmation dynamique, programmation linéaire ou algorithmes de recherche descendante. Ces approches sont cependant très générales et manquent de mécanisme d'apprentissage les accompagnant. L'aspect général de ces techniques les rend d'une lenteur inadmissible pour des problèmes d'optimisation de grandes tailles.

Les approches utilisées peuvent être classifiées principalement en deux grandes classes: méthodes de recherche exacte et approchée. Les méthodes de recherche exacte tentent

d'atteindre la solution optimale du problème coûte que coûte, ce qui induit des temps de recherche importants, alors que souvent les décideurs semblent largement se satisfaire d'une bonne solution approximative. Les techniques de recherche approchée tentent donc d'approcher la solution optimale autant que possible ce qui entraîne des temps de recherche plus réduits.

Les méta-heuristiques sont une famille d'approches de recherche approchée dotées de mécanismes généraux leurs permettant une bonne investigation de l'espace de recherche. Cette famille renferme des méthodes telles que la Recherche Tabou, les Algorithmes Génétiques, le Recuit Simulé et les Colonies de Fourmis.

Le chapitre suivant sera dédié à l'étude des méta-heuristiques, les fondements de bases les régissant et leurs applicabilité aux problèmes d'ordonnancement.

Partie II: Méta-heuristiques et problèmes d'optimisation combinatoire

II.1 Introduction

En l'absence d'approches exactes garantissant des temps d'exécution raisonnables et capables de résoudre certains problèmes d'optimisation combinatoire. Les techniques de recherche et particulièrement les méta-heuristiques se sont illustrées comme une alternative prometteuse. La pratique nous fait observer que les décideurs semblent se satisfaire largement et généralement d'une bonne approximation de la solution optimale. Les méta-heuristiques se présentent dans ce cas comme une bonne façon de garantir des bonnes solutions approchées tout en respectant des délais de réponse raisonnables.

Les méta-heuristiques[Col 97] forment une famille de techniques incluant le Recuit Simulé (RS)[Sia 95], la Recherche Tabou(RT)[Glo 96], les Algorithmes Evolutionnistes (AE)[Gol 89][Gen 96], les Colonies de Fourmis (CF)[Gra 99]... Chaque technique est décrite par un ensemble de mécanismes et de principes plus ou moins généraux plutôt qu'une méthode précise et rigide. Elles sont de ce fait, applicables à une large gamme de problèmes d'optimisation. Et il incombe donc au concepteur d'adapter ces mécanismes et principes afin de palier aux exigences du problème traité.

L'application des méta-heuristiques aux problèmes d'optimisation et d'ordonnancement en particulier à jouit d'une grande attention de la part des chercheurs. Ce qui a fait immerger trois tendances générales d'approches. Premièrement, les méthodes se basant exclusivement sur une heuristique unique telle la recherche tabou [Her 95], les algorithmes évolutionnistes[Cau 95] ou le recuit simulé[Fle 95][Abr 91] ou encore les colonies de fourmis [Son 99]. Deuxièmement, celles qui procèdent par hybridation de plusieurs méthodes afin de tirer profit de leurs puissances respectives, citons entre autres : la Recherche Tabou et l'Algorithme Génétique [Gre 98], la Recherche Tabou et Branch and Bound [Woo 99], L'Algorithme Génétique et recherche descendante[Mur 94], Algorithme Génétique et Recuit Simulé[Ozc 96]. La troisième approche, exploite la possibilité d'effectuer la recherche sur les machines parallèles. Ceci étant conditionné par la disposition de la méta-heuristique a être parallélisée. Comme exemple de telles tentatives citons: la Recherche Tabou parallèle [Tal 98a][Tal 98b], Les Algorithmes Génétiques parallèles[Tal 91][Tal 95][Abr 92] et le recuit simulé parallèle [Abr 91][Ram 96].

Ce chapitre se consacre à l'étude de ces méta-heuristiques, leurs mécanismes de base et les principes qui les régissent. Nous essayerons d'élaborer cette étude sur la base de critères de performance qui nous serviront de paramètre de comparaison entre les différentes heuristiques. Les possibilités de parallélisation et d'hybridation seront aussi décrites et leur contribution dans l'amélioration de la recherche détaillée.

II.2 Principes communs des méta-heuristiques

De façon générale les méta-heuristiques[Tal 95] consistent en une boucle itérative caractérisée par les quatre points suivants:

1. Un état courant qui peut être soit une solution unique ou une collection de solutions appartenants à l'espace de recherche. Dans le cas des Colonies de Fourmis, l'état courant est une solution incomplète au sens où seule une partie des variables composants le problème sont liées à des valeurs.

2. Un ensemble d'opérateurs qui définissent la manière de construire le voisinage de l'état courant. Le voisinage d'une solution représente l'ensemble des solutions qui sont potentiellement candidates à être ou à faire partie de l'état suivant. Les opérateurs définissent quant à eux les techniques de transition d'un état à un autre et peuvent même se présenter comme des heuristiques à part entière[Mur 94].

3. Une règle de transition qui détermine le prochain état courant à choisir parmi l'ensemble des voisins.

4. Un critère d'arrêt qui spécifie la condition d'arrêt de l'algorithme, qui représente généralement le nombre limite de solutions à visiter ou tout simplement le nombre d'itérations au bout du quel si aucune amélioration n'est observée, l'algorithme est arrêté.

La construction de l'état initial est réalisée soit d'une manière aléatoire ou en se basant sur une heuristique quelconque [Var 98].

L'efficacité d'une méta-heuristique est jugée sur la base de critères tels que la gestion du compromis intensification et diversification, l'utilisation de mémoires et l'habilité à échapper aux optimums locaux. Et c'est suivant le mérite de ces trois critères que se définira la qualité de la solution engendrée et le taux de succès de la méthode.

II.2.1 Maintien de la balance intensification/diversification

L'intensification décrit le processus par lequel l'approche encourage la recombinaison des bonnes propriétés contenues dans des solutions trouvées antérieurement. Delà, la stratégie d'intensification vise à mieux explorer les régions attractives de l'espace de recherche ayant une forte concentration de bonnes solutions. La diversification quant à elle vise à introduire de nouvelles propriétés inexistantes au niveau des solutions déjà visitées. Ce qui à pour effet d'orienter la recherche vers de nouvelles régions de l'espace de recherche non ou peu explorées.

Dans la recherche tabou [Glo 96], la notion de voisinage est vue comme un moyen d'intensification de la recherche. Dans les Algorithmes génétiques[Ren 94] les opérateurs de croisement et de mutation sont considérés comme des outils d'intensification et de diversification. Dans le recuit simulé[Sia 95] on admet la détérioration de la solution courante avec une certaine probabilité qui décroît avec le temps ce qui à pour effet d'accentuer la diversification dans les phases primaires de la recherche, puis céder le pas à l'intensification lors des phases finales.

II.2.2 Utilisation de mémoires

Les méta-heuristiques se distinguent aussi par l'utilisation ou non de mémoire de stockage. Le recuit simulé n'use d'aucune stratégie de mémorisation, le choix de la prochaine solution courante ne dépendant que de la solution actuelle. Les Algorithmes Evolutionnistes travaillent sur une population de solutions. Cette population peut être considérée, en quelque sorte, comme mémoire à court terme. La recherche tabou quant à elle excelle dans l'utilisation de mémoire à court et long terme. De ce fait le choix de la prochaine solution courante dépend aussi des phases antérieures de la recherche.

II.2.3 Risque de tomber sur un optimum local

La stratégie de diversification dans la Recherche Tabou rend la méthode capable d'échapper aux optimums locaux en introduisant de nouvelles propriétés inexistantes aux niveaux des solutions déjà ou récemment visitées. De nouveaux mécanismes ont été introduits dans les algorithmes génétiques pour empêcher la convergence prématurée de la population vers un optimum local. Dans le recuit simulé la possibilité de dégrader la solution courante permet dans les phases primaires de la recherche d'échapper aux optimums locaux.

II.3 Les Algorithmes évolutionnistes

Les Algorithmes évolutionnistes[Ren 94][Gen 96] sont apparus aux années soixante à trois endroits différents et pour des raisons distinctes.

- En Allemagne, *Schwefel* et *Rechenberg* proposèrent les stratégies d'évolution pour l'optimisation de fonctions à valeurs réelles.

- A Université de Californie, *Fogel*, *Owens* et *Walsh*[Fog 66] introduisirent La programmation évolutive au domaine de l'apprentissage automatique.

- A l'Université de Michigan, *Holland*[Hol 75] développait les Algorithmes Génétiques pour modéliser et étudier l'adaptation artificielle et naturelle des espèces vivantes. Ces algorithmes représentent un cas particulier des Algorithmes Evolutionnistes, sauf que les AGs adoptent une représentation binaire des points de l'espace de recherche.

Les Algorithmes Génétiques eurent alors une popularité grandissante, due à leur succès dans le domaine de l'optimisation de fonctions multi-modales[Hoo 96] et suite aux travaux de *Goldberg*[Gol 89]. Leur principe de base découle du phénomène d'évolution des espèces vivantes. Cette évolution se fait par deux mécanismes: la sélection naturelle et la reproduction. La sélection fait que seuls les individus les mieux adaptés survivent et se reproduisent. Quant à la reproduction, elle assure le brassage des gènes parentaux ce qui aboutit en général à des descendants dotés de nouvelles potentialités. Ce phénomène est réitéré sur une population nombreuse, qui après plusieurs générations, produit des individus mieux adaptés à l'environnement.

II.3.1 Les fondements des AGs

Les Algorithmes Génétiques (AGs) sont une transposition directe du processus naturel d'évolution au monde de l'informatique. Des chaînes représentant les individus de la

population et pouvant coder de grandes variétés de structures, sont ainsi amenées à évoluer comme les espèces vivantes.

La programmation génétique est fondée sur cinq points:

1. Une représentation chromosomique des solutions du problème.

2. Une méthode pour construire la population initiale de solutions.

3. La fonction objectif qui mesure l'aptitude d'une solution donnée et joue en quelque sorte le rôle de l'environnement.

4. Les opérateurs génétiques qui définissent la manière dont les caractéristiques des parents sont transmises aux descendants (croisement et mutation)

5. Les valeurs des paramètres utilisés par l'algorithme génétique (la taille de la population, les probabilités liées à l'application de chaque opérateur génétique et le critère d'arrêt de l'algorithme, …).

II.3.2 Algorithmes Génétiques en action

L'évolution de la population d'individus se fait via deux opérateurs : le croisement et la mutation. Le croisement opère sur un ensemble d'individus, généralement au nombre de deux, appelés parents en mélangeant certaines de leurs caractéristiques afin de produire un ou plusieurs individus enfants.

Le croisement classique consiste généralement en un échange de valeurs de gènes (voire la figure II.1). Cependant, un tel procédé s'avère inapproprié pour la majorité des problèmes d'optimisation combinatoire spécialement ceux de l'ordonnancement[Mab 97][Abr 92][Cau 95]. La mutation quant à elle agit sur les propriétés d'un individu en les modifiant, ce qui génère de nouvelles propriétés non existantes chez les parents.

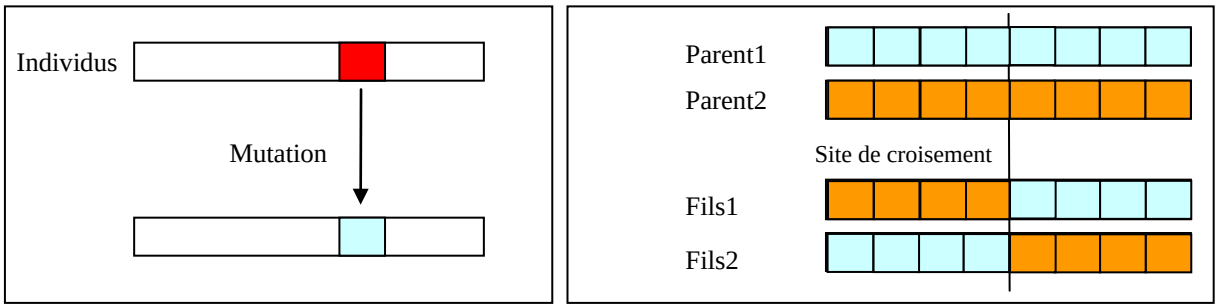


Figure II.1 : Opérateurs de mutation et de croisement classiques

A chaque itération, appelée génération, une nouvelle population est construite. La génération de la nouvelle population à partir de l'ancienne ce fait en trois étapes:

II.3.2.1 L'évaluation

L'AG évalue la fonction objectif $f(x)$ de chaque individu x de l'ancienne population.

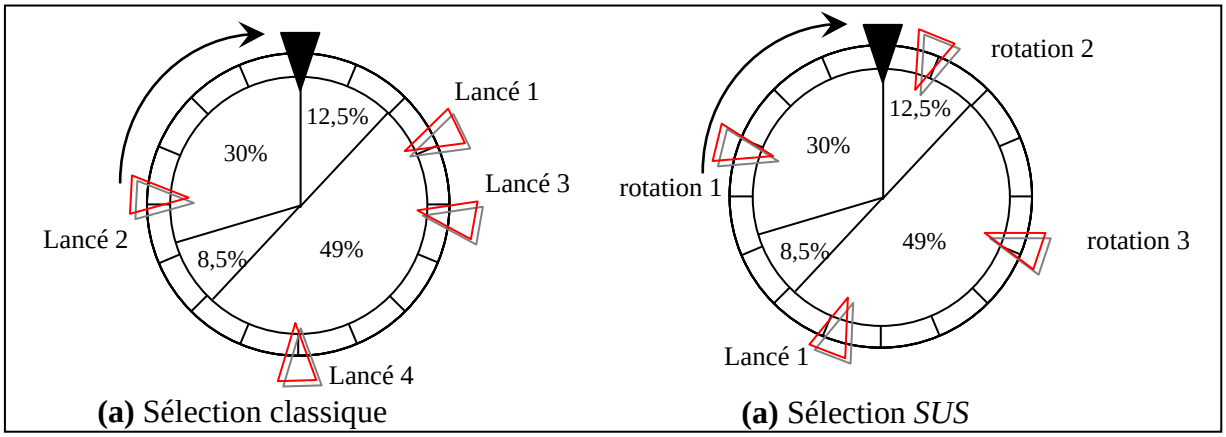
II.3.2.2 La sélection

L'AG sélectionne les individus sur la base de leurs valeurs de fonction objectif. Plus précisément, l'opérateur de sélection choisit chaque individu *x* avec une probabilité $\frac{f(x)}{\sum_{y \in Population} f(y)}$. Comme l'opérateur de sélection est appliqué *taille_pop* fois, l'espérance mathématique du nombre d'enfants de l'individu *x* sera de $\frac{n.f(x)}{\sum_{y \in Population} f(y)}$. Les individus sélectionnés constituent une population intermédiaire P`. Ce procédé est connu sous la nomination de sélection par roue biaisée. Cependant, et afin d'éviter le risque de convergence prématurée, ce qui est souvent le cas quand certains individus de la population sont beaucoup plus adaptés que d'autres, des techniques de sélection plus élaborées ont été proposées visant à minimiser le taux de participation des individus dominants.

- *Changement d'échelle*: ce procédé consiste à modifier la fonction d'évaluation *f* de façon statique ou dynamique[Gen 96] selon que la modification est opérée avant ou pendant la recherche génétique.

- *Ranking*: *Baker*[Bak 85] proposa un opérateur de sélection où les individus sont sélectionnés selon leurs rangs(*ranking*) [Ren 94][Mab 97]. Plus précisément, les individus de la population sont ordonnés selon leur adéquation de façon à ce que l'individu le moins adapté ait le rang 1. Alors chaque individu *x* a une probabilité égale à $\frac{r_x}{\sum_{y \in Population} r_y}$ d'être sélectionné où r_x représente le rang de l'individu *x*.

- *Stochastic Universal Sampling(SUS)*: *Baker*[Bak 87] proposa une autre technique visant quant à elle à éliminer les dérives du choix probabiliste. En effet, le nombre de descendants effectifs d'un individu n'est généralement pas fidèle à celui prévu théoriquement. La sélection *SUS* procède en ne choisissant que le premier individu avec la probabilité $P = \frac{r_x}{\sum_{y \in Population} r_y}$. Les autres individus sont sélectionnés en faisant tourner la roue avec un angle fixe égal à 1/*taille_pop* chaque fois (voir la figure II.2).



(a) Sélection classique **(a)** Sélection *SUS*

Figure II.2: La sélection se comporte comme une roue de loterie biaisée. Chaque individu dans la population possède son propre secteur dont les dimensions sont proportionnelles au degrés de son adéquation.

- *Sélection locale*: Dans ce type de sélection chaque individu appartient à une région de l'espace appelée voisinage. Les individus n'interagissent(reproduction) qu'avec des individus appartenant à la même région. Dans cette méthode, la moitié de la population intermédiaire P` est sélectionnée moyennant l'un des procédés précédemment cités. Le voisinage de chaque individu sélectionné est calculé, dans lequel un partenaire (pour le croisement : voire ci-dessous) est sélectionné.

- *Sélection par tournoi*[Oei 91]: Un nombre *Tour* d'individus est aléatoirement choisit de la population et le meilleur est sélectionné comme parent. Ce procédé est réitéré jusqu'à ce que la population P` soit construite. Le paramètre *Tour* appelé taille du tournoi est compris entre 2 et *Taille-pop*(nombre d'individus dans la population). Il définit aussi la pression de sélection.

II.3.2.3 La reproduction avec mutation et croisement

L'AG combine les individus sélectionnés au moyen d'opérateurs génétiques. La mutation agit en modifiant aléatoirement un ou plusieurs gènes d'un chromosome, tandis que le croisement échange certains gènes d'un parent avec ceux d'un autre. Le croisement peut s'étendre à un échange de groupe de gènes nommés bloc de construction. A chaque paire d'individus de la population P` on associe une probabilité P_c d'effectuer le croisement. Les individus générés constituent la population P``. Après quoi tout gène des individus de P`` à une probabilité P_m d'être modifié. Les individus P`` représente alors la nouvelle population et le cycle continue *Evaluation, Sélection, Reproduction.*

```
Générer la population initiale P(0), t=0
Tant que La condition d'arrêt n'est pas atteinte.
        Evaluer P(t)
        Sélectionner taille_pop individus de P(t) et construire P`
        Recombiner les individus de P` et produire P(t+1)
        T=t+1
```

Néanmoins, ce procédé n'est pas unanime. D'autres approches éliminent totalement le besoin aux populations intermédiaires, les individus générés par mutation ou croisement sont directement insérés dans la population originale.

- *Opérateur de remplacement*: dans [Mab 97] nous avons proposé une approche qui consiste à élire dans la population les individus qui seront remplacés suite à une mutation ou à un croisement. Un individu *i* de la population a une probabilité égale à (*taille_pop*-r_i)/ $\sum r_j$ d'être remplacé. L'individu le mieux adapté ayant une probabilité 0 d'être choisi, garanti de garder trace dans la population du meilleur individu rencontré pendant la recherche.

- *Remplacement Elitiste:* Dans ce cas la méthode veille à recopier lors de chaque génération les meilleurs individus se trouvant dans la population courante et de les compléter par des individus fils générés de façon traditionnelle (par application des opérateurs de croisement et de mutation).

II.3.3 Analyse des points forts des AGs

A. Combinaison de blocs de construction: les AGs se distinguent par leur faculté d'accumuler de l'information concernant un espace de recherche initialement inconnu et exploiter cette information afin de guider la recherche future dans des régions prometteuses. Cette faculté

découle du principe de combinaison de *blocs de construction* implanter par le biais de l'opérateur de croisement. Cette notion est à mettre en rapport avec les interdépendances non-linéaires entre gènes (phénomène appelé *épistasies* en biologie) qui font qu'une recherche exclusivement basée sur la mutation ne serait pas efficace. Autrement dit, les parties des chromosomes qui entraînent l'adéquation élevée des individus qui les contiennent sont combinées afin de produire des individus mieux adaptés.

B. *Le Parallélisme*: les AGs se présentent comme l'une des rares méthodes qui travaillent sur une population de solutions plutôt que sur une seule[Gol 89]. Ce qui est apparemment essentiel pour la recombinaison. En effet, le maniement d'une population d'individus n'aurait pas raison d'être si on travaillait sur une solution unique. N'oublions pas non plus qu'en un certain sens, la population constitue une *base de données* qui résume l'information acquise jusqu'au stade actuel de la recherche. Le travail sur une population d'individus est à inscrire parmi les raisons qui ont rendu les AGs une méthode hautement parallèle[Tal 95].

C. *Manipulation de données symboliques*: les données manipulées par les AGs ne sont pas forcement numériques[Gol 89] et n'offre de ce fait aucune restriction sur le type de l'espace de recherche considéré (continuité, ordre, etc.)

D. *Utilisation minimale d'information a priori*: les AGs n'exigent de l'environnement qu'une manière d'estimation d'une solution donnée, exprimée par la fonction objectif. Plus encore, certains versions des AGs ne requiert de l'environnement qu'une habilité de classement des individus entre eux (relation d'ordre totale).

E. *Balance exploration/exploitation*: dans les AGs les opérateurs de croisement et de mutation sont considérés comme des moyens d'exploitation et d'exploration. L'introduction du hasard lors des phases de sélection et de reproduction constitue un autre atout pour l'exploration. En effet, plutôt que de trancher de façon déterministe en faveur des individus les mieux adaptés, les AGs ne font que les favoriser au moyen de règles probabilistes. Ce qui laisse une petite marge de chance aux individus les moins bons afin de survivre et se reproduire. Cependant, l'effet du hasard peut être néfaste lors des étapes de génération de la population initiale, de la sélection et de reproduction. En effet, les AGs restent sensibles à la distribution de la population initiale et aux effets des *mutations* et *croisements catastrophes* [Glo 96]. La technique de sélection *SUS* (paragraphe II.3.2.2) tente de remédier à ce défaut lors de la phase de sélection.

II.3.4 Elitisme

De Jong suggérait en 1975 la nécessité d'inclure systématiquement le meilleur individu de la population courante dans la génération suivante, afin de prévenir la perte d'une telle solution en raison d'un mauvais échantillonnage lors de la sélection ou d'une destruction dû au opérateurs de reproduction. Cette approche peut être étendue à la copie des *b* meilleurs individus chaque fois dans la prochaine génération. Les tests effectués par *De Jong* montrèrent que l'introduction de l'élitisme conduit généralement à l'amélioration des performances des AGs dans le cadre de l'optimisation uni-modale. Dans le cas multi-modales il peut entraîner la convergence prématurée.

II.3.5 Maintien de la diversité

En raison des règles probabilistes utilisées dans les AGs, la population d'individus ne présente pas un échantillon représentatif de la totalité de l'espace de recherche. Ce qui accentue les risques de convergence prématurée connue sous le nom de *dérive génétique* et le blocage dans les optimums locaux. Afin de prévenir ce type de situations le maintien de la diversité au sein de la population semble inéluctable. Les objectifs de telles techniques sont de l'ordre de deux[Har 94]. Premièrement, préserver et/ou injecter la diversité dans la population afin de ralentir la convergence de l'AG vers un optimum local[Oei 91]. Deuxièmement, la localisation et la préservation de plusieurs solutions(problème multi-modales)[Har 94][Bea 93].

Pour le faire les techniques proposées opèrent de deux façons. La première approche consiste à modeler l'environnement de sélection dans certaines régions de l'espace de recherche. Ce qui a pour effet de prévenir la convergence prématurée. Ce type d'approche englobe le *Sharing* et le *Crowding*. La seconde approche opère par imposition de certaines contraintes sur les individus de la population afin d'empêcher la prolifération d'individus de très bonnes adéquations. La méthode de *maintien de distance* est un exemple de telles approches.

II.3.5.1 Maintien de distance

Mauldin [Mau 84] propose l'utilisation d'une valeur *k* qui définit la distance minimale autorisée entre deux individus de la population. A la génération d'un individu *i* sa distance par rapport au reste de la population est calculée. Après quoi, si l'individu *i* est jugé identique à un autre individu *j* (*distance*(*i, j*) < *k)* alors l'individu *j* subit une série de mutations jusqu'à ce que la distance *k* soit respectée.

II.3.5.2 Présélection de Cavicchio

Cavicchio(1970) part de l'observation que les individus nés d'un croisement ont de grandes chances de ressembler à leurs parents. Dans cette technique, un individu fils plus adéquat que l'un de ses parents est inséré dans la population en remplacement de celui-ci.

II.3.5.3 Niches écologiques (Sharing)

Goldberg[Gol 89] souligne que "*Intuitivement, nous pouvons considérer les niches comme une tâche ou plutôt une règle d'organisation d'un environnement, et nous pouvons voir les différentes espèces comme une classe d'organismes ayant des caractéristiques communes. Cette séparation de l'environnement … en plusieurs sous-ensembles est tellement présente dans la nature…*".

Les techniques à base de *niches écologiques* [Har 94][Tal 99][Gol 89] s'appuient toutes sur des mécanismes de formation de sous populations d'individus stables. Le *Niching Séquentiel* de *Beasley, Bull et Martin*(1993)[Bea 93] procède par lancement successif de la recherche génétique. Lors de chaque exécution de l'AG, la meilleure solution trouvée est sauvegardée. Pour prévenir la convergence vers la même région lors des AGs ultérieures, l'adéquation des points appartenant au périmètre des solutions déjà trouvées est dégradée.

La technique de *Sharing* due à *Goldberg* et *Richardson*(1987) consiste en la dégradation de l'adéquation des individus appartenant à des espaces de recherche de forte concentration de solutions. Le phénomène du *Sharing* est observable dans la nature, quand

une espèce animale est contrainte de contrôler la taille de sa population en fonction de la quantité des ressources qu'elle se partage. Ce procédé a pour effet de:

- Changer le paysage du problème.
- Favoriser la dispersion des solutions de la population dans l'espace de recherche.
- La formation de sous populations d'individus semblables.

La dégradation de l'adéquation d'un individu est réalisée grâce à une fonction appelée fonction de partage (*sh*). La fonction de coût révisée d'un individu S_i notée $f'(S_i)$ est égale à la fonction de coût originale *f* multipliée par le compteur de niche de l'individu $m(S_i)$.

$$f'(S_i) = f(S_i) * m(S_i) \tag{II.1}$$

Le compteur de niche calcule le degré de similarité qu'a un individu avec le reste de la population. La fonction de partage *sh* est définie comme suit :

$$m(x) = \sum_{y \in pop} sh(dist(x, y)) \tag{II.2}$$

$$sh(dist(x, y)) = \begin{cases} 1 - \left(\frac{dist(x, y)}{\gamma} \right)^{\alpha} & Si \quad dist(x, y) < \gamma \\ 0 \text{ sinon} \end{cases} \tag{II.3}$$

Dans cette formule la constante γ désigne le seuil de dissimilarité (taille des niches). C'est à dire la distance au bout de laquelle deux individus S_i et S_j ne sont plus considérés comme appartenant à la même niche. La constante α par contre permet de contrôler et réguler la forme de la fonction *sh*. Selon que la distance entre les individus est calculée dans l'espace de décision (la représentation chromosomique de l'individu) ou dans l'espace objectif (adéquation de l'individu) ou les deux, trois possibilités peuvent se présenter. Dans [Mah 95] on peut trouver une étude comparative des deux méthodes de diversification génotypique et phénotypique.

II.3.5.4 Crowding

Une quatrième technique de maintien de la diversité est connue sous l'appellation de *crowding* [Tal 99][Har 94]. *De Jong*(1975)[Tal 99] a été le premier à suggérer l'utilisation de l'opérateur de *crowding* dans la phase de remplacement des AGs (voir le paragraphe II.3.2). Après la génération d'un nouvel individu, il ne s'agit plus de l'insérer dans la population en remplacement de ces parents comme dans l'approche de *Cavicchio* ou de choisir une autre victime à faire sortir comme dans [Mab 97][Rah 99]. L'individu généré remplace la solution qui lui est la plus semblable parmi *CF* autres individus. *CF* désigne le facteur de *crowding*. Les tests réalisés par *De Jong* sur des problèmes multi-modales montraient une plus grande rapidité de convergence vers l'optimum global[Har 94].

II.3.5.5 Restriction de voisinage

D'autres travaux[Tal 99] proposés pour le maintien de la diversité sont basés sur la *restriction de* voisinage. Le principe est de permettre la reproduction entre deux individus que s'ils

s'avèrent être similaires. D'autres travaux vont dans le sens contraire en empêchant plutôt la reproduction entre individus similaires pour éviter l'inceste.

II.3.6 Algorithmes Génétiques Parallèles

Jamais une méta-heuristique n'a été autant sujet à des tentatives de parallélisation que les AGs[Tal 95][Mab 97][Tal 91]. Ce qui est en grande partie dû à la disposition des AGs à l'être et le coût d'exécution important des AGs séquentiels. *Talbi*[Tal 95] propose une classification des approches de parallélisation selon la répartition de la population d'individus sur les différents processeurs qui composent l'architecture cible. Les architectures parallèles peuvent être soit à mémoire commune (architecture centralisée) ou à mémoire distribuée et dans ce cas les processeurs communiquent entre eux par envoi de messages.

Selon cette distinction trois modèles parallèles d'AG *ont* été identifiés:

1. *Le modèle centralisé*: utilise une population unique d'individus sur la quelle travaillent plusieurs processus.
2. *Le modèle distribué:* utilisant plusieurs sous populations à raison d'une pour chaque processus.
3. *Le modèle fortement ou totalement distribué:* dans ce cas chaque individu de la population correspond à un processeur.

II.3.6.1 Modèle centralisé

Le modèle centralisé consiste à utiliser l'AG standard en effectuant les étapes d'évaluation, de sélection et de reproduction de façon simultanée. L'aspect du parallélisme apparaît alors dans le recouvrement entre étapes ce qui conduira nul doute à des temps d'exécutions meilleurs.

Une mise en œuvre de l'AG sur une ferme de processus à été réalisée dans [Tal 95](voir la figure II.3). Un processus maître effectue la sélection des individus et les diffuse au processus esclaves. Une fois qu'un processus esclave reçoit une paire d'individus, il effectue sa reproduction, évalue les individus produits et les renvois au processus maître. Celui-ci effectue le remplacement sur la population commune et relance le cycle de nouveau sur la nouvelle population.

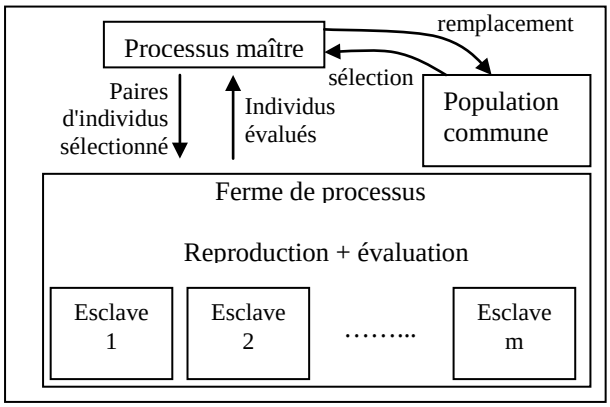


Figure II.3: Modèle centralisé de *Talbi*

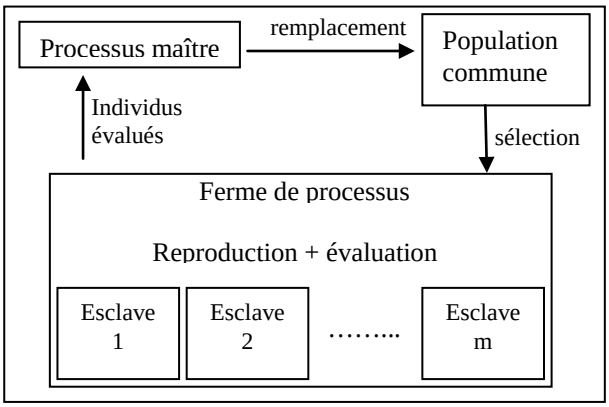


Figure II.4: Modèle centralisé d'*Abramson*

Abramson propose une autre variante de ce procédé [Abr 92](voir la figure II.4) dans laquelle le processus maître ne s'occupe plus de la sélection des paires d'individus, il se limite

à la seule tâche de remplacement. La phase de sélection est réalisée au niveau de chaque processus esclave.

Ce qui fait l'intérêt du modèle d'*Abramson* est que l'étape de sélection est réalisée en parallèle, chaque processus esclave s'occupe de la sélection de ces propres individus. De ce fait la synchronisation entre processus n'est opérée qu'une seule fois lors de l'étape de remplacement contrairement au premier modèle où la synchronisation est obligatoire même pendant la sélection.

Les inconvénients de cette approche, résident premièrement dans la perte de temps due aux délais d'attente du processus maître des résultats provenant des processus esclaves et vis versa. Deuxièmement, l'état du processus maître conditionne le comportement de tout l'algorithme et une panne de celui-ci s'avère fatale pour la recherche. Troisièmement, le coût de communication croît exponentiellement en fonction de la taille de la population[Tal 91].

II.3.6.2 Modèle distribué (modèle à décomposition)

Dans ce modèle plusieurs sous populations sont générées, chaque processus exécute l'AG standard sur une sous population qui lui est associé. A des périodes de temps déterminées les processus procèdent à un échange d'individus. Ce phénomène est connu sous l'appellation de *migration*(figure II.5). Le voisinage d'un individu, la fréquence d'échange, la stratégie du choix de l'individu migrateur et de l'individu à remplacer lors de la réception d'un nouvel individu sont autant de paramètres de l'algorithme.

Ce modèle est intéressant en vue de son indépendance vis à vis de l'architecture de la machine (nombre de processeurs). Elle donne de bonnes performances pour des nombres de processus restreints. L'inconvénient principal reste le fait que la disposition naturelle des AGs à être parallélisés n'est pas complètement exploitée. L'application de l'AG sur chaque sous population étant réalisée de manière séquentielle[Tal 95].

Le taux de diversification et d'intensification est largement affecté par le choix des paramètres de l'algorithme. Quand le taux de connexion entre processus est élevé, ceci tend à favoriser l'implantation des meilleurs individus dans toutes les sous population, ce qui du coup accentue l'intensification. Une fréquence d'échange élevée intensifie la coopération entre processus fortement connectés, ce qui débouche sur des sous population qui tendent rapidement à être similaires. L'utilisation de plusieurs sous populations relativement indépendantes a pour effet d'engendrer une dispersion de la recherche dans l'espace dû à une pression de sélection moins importante. Ce qui est d'un grand apport pour la diversification et d'une grande utilité en cas d'optimisation multi-modales.

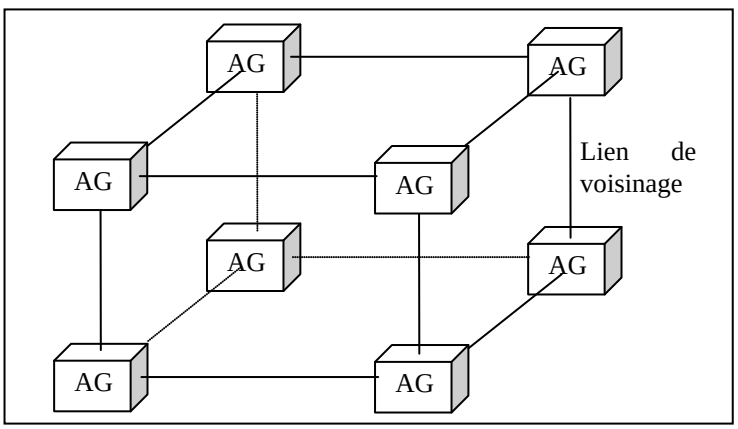


Figure II.5: Modèle distribué

II.3.6.3 Modèle Massivement parallèle

Dans [Tal 95]on trouve une description d'AG massivement parallèle s'exécutant sur une machine à architecture cible à base de *transputers T800*. Sur chaque processeur est placé un individu de la population. A chaque itération, un individu reçoit les individus voisins et se reproduit avec eux. Les individus générés localement sont alors évalués et la solution locale est mise à jour. Un grain de parallélisme plus grand est engendré, avec une plus faible pression de sélection ce qui offre un acquis pour la diversification. L'individu local est mis à jour à partir des solutions générées en choisissant la meilleure trouvée.

II.3.7 Algorithmes Génétiques et problèmes d'ordonnancement

Les AGs ont souvent été appliqués aux problèmes d'ordonnancement. Nous allons nous étaler dans ce qui suit sur la façon dont ils ont été utilisés pour chacun des problèmes : d'emploi du temps, problèmes d'ateliers (*Job Shop*, *Flow Shop*) et routage de véhicules. Ces problèmes étant souvent doublés d'un problème de satisfaction de contraintes, nous expliquerons tout dabord comment les AGs prennent en compte les contraintes liées au problème.

II.3.7.1 AG et satisfaction de contraintes

Dans les AGs, le problème de satisfaction de contraintes est traité de l'une des trois façons suivantes:

- *Pénaliser la fonction objectif* : ce qui est utilisé quand les opérateurs génétiques sont passibles de générer des individus fils non admissibles, dont le sens qu'ils ne respectent pas l'ensemble des contraintes. La fonction objectif révisée f' prend la forme :

$$f'(i) = f(i) + \sum_{c \in Contraintes} P_c \times V(i,c) \qquad \text{(II.4)}$$

$V(i, c)$ = {1 si la contrainte c est violée dans l'individu i et 0 sinon}

P_c représente la pénalité à infliger à une solution suite à la violation de la contrainte c. Elle détermine aussi l'importance de la contrainte associée.

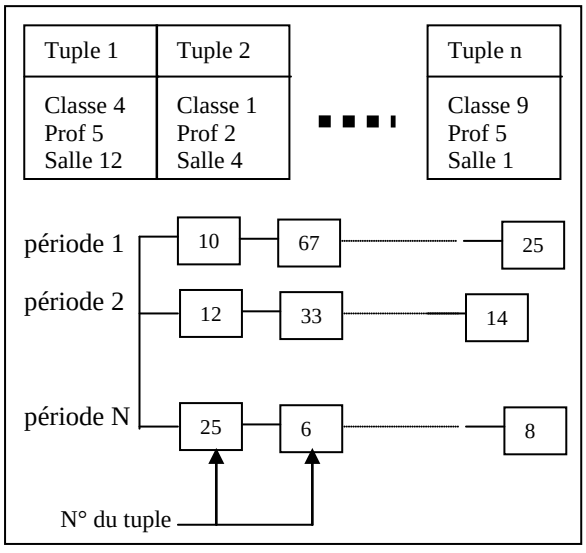


Figure II.6: représentation d'un emploi du temps d'Abramson

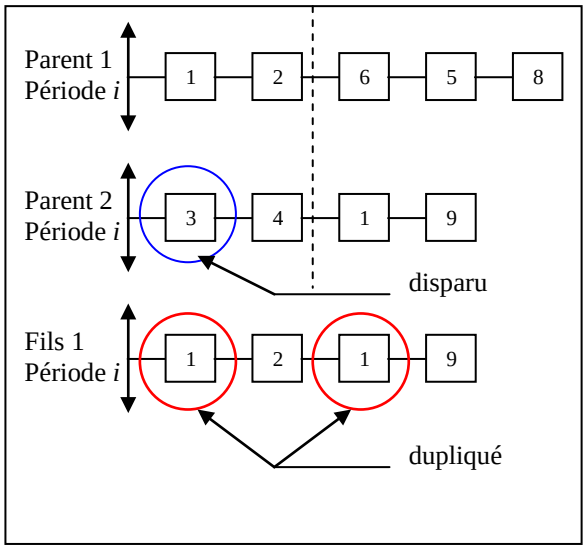


Figure II.7: Opérateur de croisement d'Abramson

- *Réparation des solutions* : Cette technique consiste à permettre la génération d'individus fils non admissibles par les opérateurs génétiques. Après quoi, ces individus seront corrigés dès leur naissance en lançant un algorithme de réparation. Dans [Abr 92] par exemple(figure II.6), le croisement de deux emplois du temps peut générer des individus où certains cours se voient dupliquer dans le chromosome et d'autre disparaissent carrément (figure II.7). L'algorithme réparateur est alors lancé, afin d'éliminer les individus dupliqués et insérer les individus disparus.

- *Opérateurs génétiques adaptés au problème* : Cette troisième alternative, consiste à concevoir des opérateurs génétiques garantissants la génération d'individus admissibles en leur introduisant les connaissances que l'on a du problème et de la structure du chromosome.

II.3.7.2 AG et Problème d'emploi du temps

L'application des AGs aux problèmes d'emploi du temps abonde dans la littérature[Abr 92][Ros 95][Wez 95][Pae 95].

II.3.7.2.1 Approche d'Abramson & Abela

Abramson et *Abela*[Abr 92] se base sur une représentation directe de l'emploi du temps(voir la figure II.6), où les cours ayant lieu à la même période sont regroupés dans la même liste. Le but de la résolution se limite à éliminer les cas de conflits. La fonction objectif correspond à la somme des violations des contraintes. L'opérateur de croisement schématisé dans la figure II.7, consiste à définir pour chaque période un site de croisement. Alors la liste des cours, associés à la même période dans l'emploi du temps fils, sera composée des premiers éléments du parents1 suivie des derniers éléments du parent2.

II.3.7.2.2 Approche de Ross, Corne et Fang

Ross, Corne, Fang [Ros 95] introduisirent deux nouveaux mécanismes servant à guider et à optimiser la recherche génétique.

- *Mutation dirigée par la violation*: *Ross, Corne, Fang* proposèrent une variante intéressante de l'opérateur de mutation capable de détecter les sources de conflits dans une solution. Lors de l'étape d'évaluation d'un individu un score de violation est gardé pour chaque gène du chromosome. Ce score correspond au nombre de contraintes violées aux quelles contribue le gène. Par exemple, quand une contrainte d'exclusion mutuelle entre deux cours t_i, t_j est violée les gènes correspondants aux périodes de déroulement de ces deux cours voient leurs scores de violation augmentés de un. Ainsi les gènes ayants des scores de violation élevés seront ceux qui sont les plus problématiques. Cette information est utilisée par la suite pour favoriser l'altération des gènes à scores de violation élevés.

- *La délta-évaluation*: Les auteurs de l'article proposent aussi une amélioration de l'étape d'évaluation visant à réduire son temps de calcul. La *délta-évaluation* consiste à utiliser l'évaluation des parents afin de calculer la valeur d'adéquation des enfants. La technique part de la remarque suivante: les individus enfants présentent des similarités avec leurs parents, delà seul les modifications sur la fonction objectif dues aux parties du chromosome altérées serons examinées dans l'évaluation de l'individu enfant.

II.3.7.2.3 Notre Approche

Dans [Mab 97][Rah 99], nous proposons la résolution du problème d'emploi du temps par les algorithmes génétiques. L'opérateur de croisement consiste à choisir aléatoirement une période *p*. Deux emplois du temps fils sont alors générés. Le premier est identique au *Parent1* dans la partie antérieure à *p* et identique au *Parent2* dans la partie ultérieure à *p* et inversement pour le deuxième enfant. Dans cette approche nous proposons aussi l'utilisation de plusieurs opérateurs de mutation spécialisée. Un opérateur de mutation est associé à chaque type de contrainte.

Onze types de contraintes sont considérés: non conflit sur les étudiants, non conflits sur les enseignants, non conflit sur les locaux, éviter qu'un enseignant ou qu'un étudiant ait des temps creux, ne pas programmer deux cours du même module à la même journée, respecter les capacités des locaux, éviter les cours uniques d'une journée, respect des préférences des cours pour une plage de périodes ou de jours. Chaque opérateur de mutation spécialisée, garde des scores de violation qui concernent seulement les contraintes du type pour le quel il s'est spécialisé. La mutation d'un gène est effectuée selon le principe de la mutation dirigée par la violation.

Le but de chaque opérateur de mutation spécialisée est d'éliminer les violations enregistrées concernant le type de contraintes associé. La fréquence d'appel d'un opérateur de mutation spécialisée est en fonction de l'importance et du degré de violation de la contrainte associée.

II.3.7.3 AG et Problèmes Flow Shop

Plusieurs applications des l'AG ont été dévouées aux problèmes de Flow Shop. L'objectif à atteindre consiste généralement à minimiser le temps d'accomplissement total(*makespan)* ou celui du retard.

Le premier souci des chercheurs a été l'élaboration de techniques de codage d'un ordonnancement, trois tendances sont apparues :

- *Représentation directe d'un ordonnancement*: qui consiste à décomposer le chromosome en une suite de sous-chromosomes, un par ressource (machine). Chaque sous-chromosome correspond à l'ordre des tâches sur la ressource. Un gène désigne alors la date de début de la tâche (figure II.8). La difficulté dans ce type de représentation réside dans le choix des opérateurs génétiques qui doit se faire de manière à maintenir les contraintes d'*exclusion* et de *précédence* vérifiées.

- *Représentation indirecte d'un ordonnancement*: dans ce cas le chromosome ne représente plus un ordonnancement mais la séquence de passage des tâches sur les différentes ressources. L'ordonnancement est déduit plus tard par un algorithme spécifique. Dans [Sim 82], l'ordonnancement considéré est celui correspondant au calendrier au plus tôt des différentes tâches. L'intérêt de ce type de représentation est qu'il permet un meilleur contrôle de l'ensemble des contraintes.

- *Représentation adaptée au problème*: dans ce cas la représentation prend en compte les connaissances liées au problème. De ce faite des structures en arbre, matrice ou graphe, peuvent s'avérer mieux adaptées au problème traité.

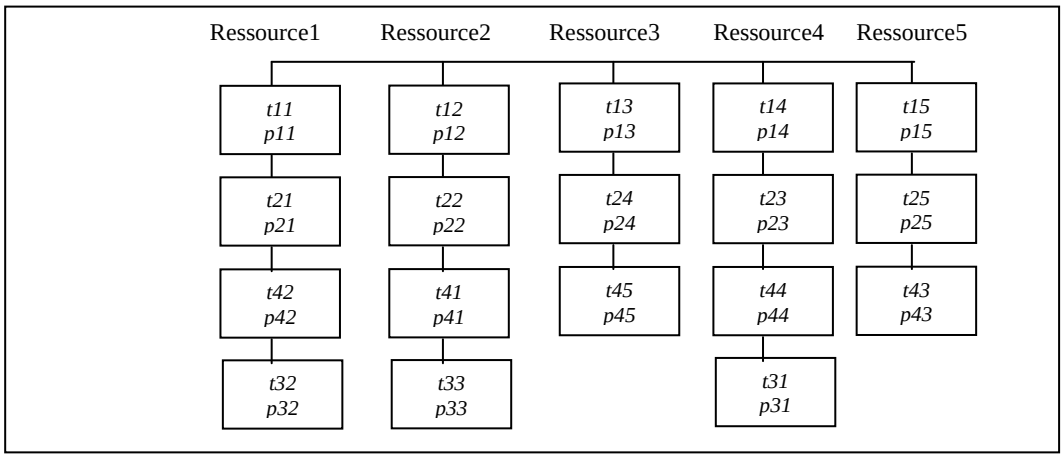


Figure II.8: représentation directe d'un ordonnancement, p_{ij} représente la dates de début de t_{ij}

II.3.7.3.1 Approche de Murata, Ishibuchi

Murata et *Ishibuchi* [Mur 94] proposent des opérateurs génétiques adaptés au problème du *Flow Shop*. Les individus(ordonnancements) sont codés par des permutations déterminant l'ordre de passage des tâches sur les différentes machines(*représentation indirecte*).

Les opérateurs de croisement et de mutation utilisés sont décrits dans la figure II.9. L'opérateur de croisement travaille avec deux points de coupure. L'individu fils généré correspond au mélange des gènes se trouvant aux extrémités du premier parent et des gènes représentant les tâches restantes, dans leur ordre d'apparition, dans le deuxième parent.

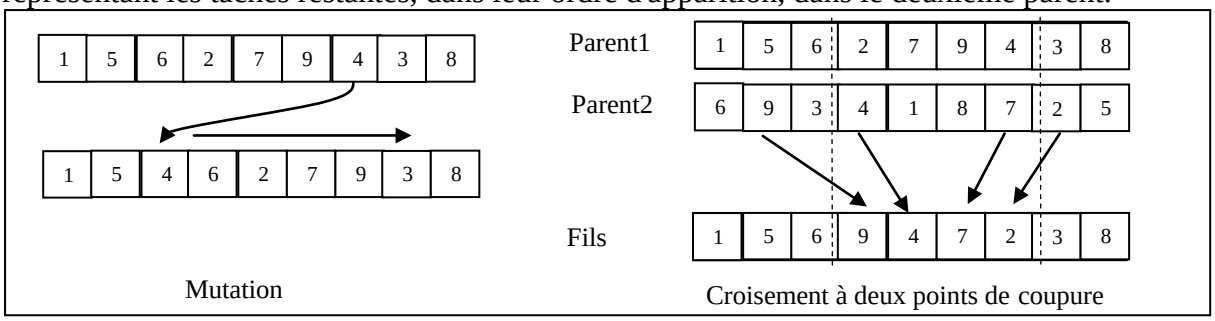


Figure II.9 : Opérateur de mutation et de croisement pour le problème du *Flow Shop*

L'approche prônée par *Murata* et *Ishibuchi* use de la recherche locale comme supplément de la recherche génétique. A chaque génération, la recherche locale est lancée sur chaque individu de la population. Pour réduire le temps d'exécution induit par ce procédé, les auteurs limitent le nombre de voisins générés pour chaque individu.

II.3.7.3.2 Approche de Gen, Tsujimura et Kubota

Comme pour *Murata* et *Ishibuchi* [Gen 96], un ordonnancement est codé par une permutation. Les auteurs adoptent le croisement de *Goldberg* appelé *PMX*. Nous donnons dans la figure II.10 un exemple de l'application de cet opérateur. La permutation Fils générée et construite de la façon suivante.

```
Fils = {0,0,0,…,0}
Pour i := 1 à N
Faire
    P = random(2)
    Si P = 0 et Parent1[i]∉{Fils[1],Fils[2],…,Fils[i-1]} alors Fils[i] = Parent1[i]
    Sinon Si P = 0 et Parent1[i]∈{Fils[1],Fils[2],…,Fils[i-1]} alors Fils[i] = J∈{1,2,…N} -{Fils[1],Fils[2],…,Fils[i-1]}
```

Sinon Si P = 1 et Parent2[i]∉{Fils[1],Fils[2],…,Fils[i-1]} **alors** Fils[i] = Parent2[i]
Sinon Si P = 0 et Parent2[i]∈{Fils[1],Fils[2],…,Fils[i-1]} alors Fils[i] = J∈{1,2,…N} - {Fils[1],Fils[2],…,Fils[i-1]}

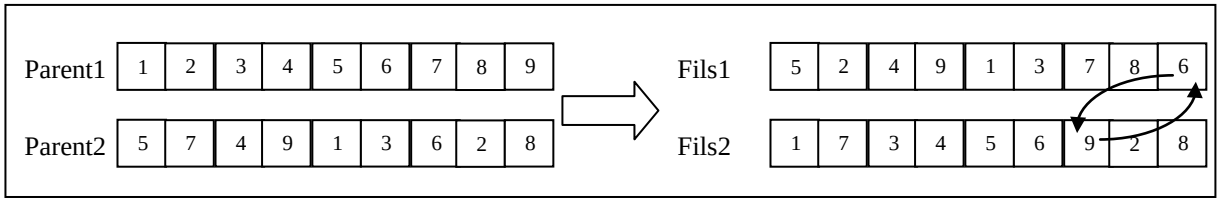


Figure II.10: Opérateur de croisement de *Goldberg*.

L'opérateur de mutation quant à lui revient à la permutation aléatoire des positions de deux jobs. Les tests effectués par *Gen, Tsujimura* et *Kubota* démontrent que l'utilisation des AGs comme méthode de recherche est plus efficace que l'ensemble de toutes les heuristiques citées dans le paragraphe I.11.3.1.2.

II.3.7.3.3 Approche de Reeves

La particularité de l'approche de *Reeves*[Ree 95] est l'utilisation de l'heuristique *NEH*(paragraphe I.11.3.1.2.3) lors de la génération de la population initiale. Un individu de la population est construit en utilisant l'heuristique *NEH*, le reste des individus est généré de façon aléatoire. *Reeves* fait remarquer que ce procédé entraîne une plus grande rapidité dans l'atteinte de la solution finale sans dégradation de la qualité de la solution.

L'opérateur de croisement adopté par *Reeves* ressemble à celui de *Murata* et *Ischibuchi* sauf qu'il est à un seul point de coupure. L'individu fils est constitué de la séquence du *parent 1* précédant le point de coupure suivit des jobs restants dans le même ordre que celui rencontré dans le *parent 2*(figure II.11).

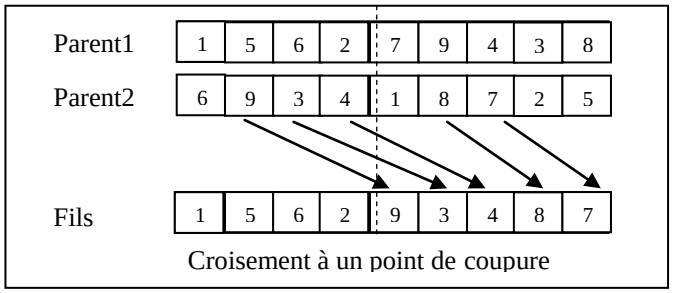


Figure II.11: Opérateur de croisement de *Reeves*

II.3.7.4 AG et Problèmes Job Shop

Le problème du Job Shop étant une extension du problème de Flow Shop, sa résolution s'avère plus compliquée et nécessitant des mécanismes supplémentaires. Ces améliorations peuvent être classées en deux grandes classes :

- L'adaptation des opérateurs génétiques: cette approche consiste généralement à la conception d'opérateurs de croisement inspirés de l'algorithme de *Giffler* et *Thompson*(paragraphe I.11.3.2.1) ou à l'utilisation d'opérateurs de mutation basés sur la recherche locale.

- User d'autres heuristiques comme supplément à la recherche génétique: cette approche englobe l'utilisation d'heuristiques pour générer une bonne population initiale.

La conception des opérateurs de croisement et particulièrement de mutation dépend largement du type de codification choisi. *Gen* et *Cheng* [Gen 96] ont recensé huit types de codifications utilisées dans la littérature.

II.3.7.4.1 Croisement basé sur l'algorithme de Giffler et Thompson

L'algorithme suivant décrit l'opérateur de croisement basé sur l'algorithme de *Giffler* et *Thompson*[Sim 82][Gen 96].

S = l'ensemble des tâches actuellement planifiables.
ϕ_{ij} = la date de terminaison au plus tard de la tâche t_{ij}.

1. Soit S l'ensemble des tâches n'ayant pas de prédécesseurs
2. Déterminer $\phi^* = \min\{ \phi_{ji} \mid o_{ji} \in S \}$ et la machine r^* sur laquelle ϕ^* est réalisé
3. Soit G_r^* l'ensemble des tâches $t_{ji} \in S$ qui requièrent la machine r^*
4. Choisir une tâche depuis G_r^* comme suit:
5. Générer un nombre aléatoire $\varepsilon \in [0,1[$ et le comparer au taux de mutation P_m.
6. Si ($\varepsilon < P_m$) alors choisir une tâche aléatoirement de G_r^*
7. Sinon sélectionner l'un des parents P_s avec une probabilité égale à 0,5. Chercher la tâche t_{ji}^* ordonnancée la première dans P_s parmis toute les tâches de G_r^* . Ordonnancer t_{ji}^* dans le fils suivant ϕ_{ji}
8. Mettre à jour *S en supprimant* t_{ji}^* et en la remplaçant par son successeur direct
9. Retour à l'étape 2 jusqu'à ce que l'ordonnancement fils soit complété

II.3.7.4.2 Mutation basée sur la Recherche Locale

Il s'agit de voir dans l'opérateur de mutation un moyen d'intensification[Mur 94] de la recherche afin d'améliorer les solutions déjà existantes. L'opérateur de mutation consiste à générer l'ensemble des voisins de l'individu à muter et à choisir le meilleur voisin améliorant celui-ci. Le procédé s'arrête quand aucune amélioration n'est possible.

II.3.7.4.3 Approche de Gen, Tsujimura et Kubato

Cette approche[Gen 96] s'appuie sur une représentation à base de tâches. Un ordonnancement est codé par une permutation à valeurs redondantes. Un numéro dans la permutation désigne le numéro du job. La première apparition d'un job correspond à la première tâche de ce job et ainsi de suite(figure II.12).

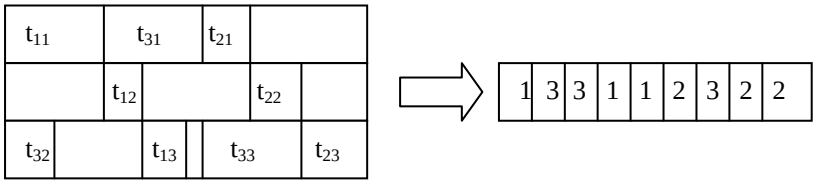


Figure II.12: représentation basée sur les tâches

Les auteurs proposent un opérateur de croisement par échange d'ordonnancements partiels. Ayant une paire d'individus P_1 et P_2 comme dans la figure II.13, on choisit une position dans le parent P_1 de façon aléatoire, soit par exemple la sixième position correspondante au job J_4. On localise alors la position de la prochaine tâche du même job J_4 dans P_1. Nous obtenons donc l'ordonnancement partiel (4 1 2 4). Comme les tâches prises dans P_1 correspondent à la première et la deuxième tâche du job J_4, nous prenons de la même façon l'ordonnancement partiel (4 1 3 1 1 3 4) de P_2 compris entre la première et la deuxième tâche du job J_4. L'échange des deux ordonnancements partiels est représenté dans la figure II.14. L'échange ainsi réalisé génère généralement des individus fils non admissibles vu les pertes et les

redondances qui peuvent apparaître. La réparation de l'individu fils passe premièrement par le recensement des gènes perdus et ceux redondants. Les gènes redondants seront supprimés de façon à conserver l'ordre des tâches dans l'ordonnancement partiel et à conserver intacte les ordonnancements partiels échangés. Par exemple dans F_1(figure II.14) nous avons un excès de deux tâches du job J_1. Puisque dans P_2, d'où l'ordonnancement partiel (4 1 3 1 1 3 4) est tiré, les trois tâches du job J_1 correspondent au 1^{er}, $2^{ème}$ et $3^{ème}$ tâche du job, nous éliminons donc les tâches de la $3^{ème}$ et $14^{ème}$ position de F_1 (figure II.15). Les gènes perdus sont insérés de façon aléatoire n'importe où dans l'ordonnancement sauf à l'intérieure de l'ordonnancement partiel.

P_1=[3 2 1 2 3 **4 1 2 4** 4 1 3 4 1 2 3]

P_2=[**4 1 3 1 1 3 4** 1 2 3 4 2 2 2 3 4]

Figure II.13: Sélection des ordonnancements partiels

F_1=[3 2 1 2 3 **4 1 3 1 1 3 4** 4 1 3 4 1 2 3]

F_2=[**4 1 2 4** 1 2 3 4 2 2 2 3 4]

Figure II.14: Echange des ordonnancements partiels

F_1=[**3** 2 **1** 2 **3** 4 1 3 1 1 3 4 4 **1** 3 4 1 2 3]

F_1=[2 2 4 1 3 1 1 3 4 4 3 4 1 2 3]

Figure II.15: Elimination des gènes redondants

F_1=[2 2 4 1 3 1 1 3 4 2 4 3 4 1 2 3]

Figure II.16: Insertion des gènes perdus

II.*3.7.4.4 Approche de Cheng, Gen et Tsujimura*

Cette approche[Gen 96] se présente comme une amélioration de l'approche précédente. Ces améliorations concernent trois points clés:

1. L'utilisation d'une procédure de décodage produisant des ordonnancements actifs. Un ordonnancement actif est un ordonnancement où chaque tâche est lancée dès que les contraintes d'exclusion(disponibilité de la machine) et de précédence sont vérifiées.

2. Un opérateur de croisement plus simplifié est proposé. Dans ce cas deux ordonnancements partiels ayant le même nombre de gènes sont choisis de façon aléatoire depuis les deux parents. On recense alors les gènes perdus et redondants après l'échange des deux ordonnancements partiels. Seulement cette fois-ci les insertions et éliminations des gènes sont opérées de façon aléatoire.

3. Les auteurs proposent l'utilisation d'un opérateur de mutation à base de recherche locale.

II.3.7.4.5 Approche de Falkenauer et Bouffouix

Cette approche se base sur une représentation par liste de préférence. Un ordonnancement dans ce cas est codé par une liste de *M* sous listes. Chacune est composée de *N* éléments représentant les tâches sensées être traitées par la machine en question. L'ordre des tâches dans une sous liste ne décrit pas l'ordre de traitement des tâches sur la machine, mais simplement la tâche à planifier en premier en cas d'indécision lors du décodage(voir [Gen 96]).

Les auteurs proposent une variante de l'opérateur de croisement *OX* (opérateur souvent appliqué au problème du voyageur de commerce) noté *LOX*. L'opérateur *LOX* est réalisé sur chaque paire de sous listes séparément(figure II.17) de la façon suivante:

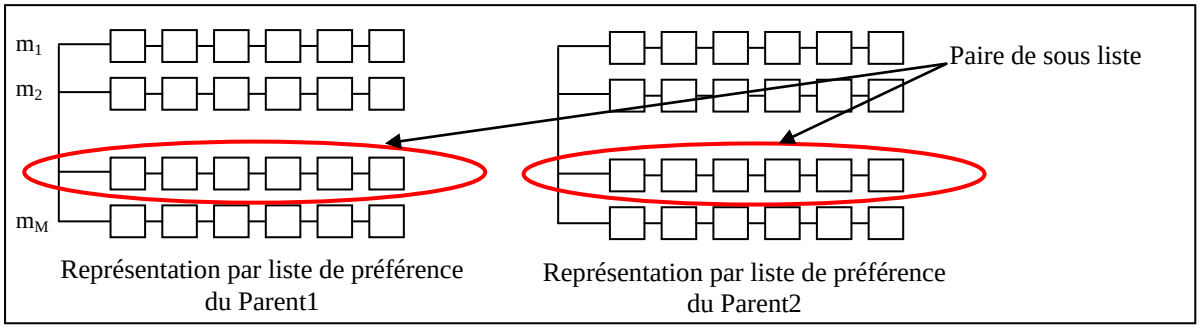


Figure II.17: le croisement opère sur chaque paire de sous listes.

1. Sélectionner une partie de chaque sous liste de même taille comme dans la figure II.18. Supprimer la partie$_2$ de P$_1$, ce qui laissera des trous (notés par des t).

2. Centrer alors les trous démarrant des extrémités et arrivant au centre comme c'est montré dans la figure II.19. De façon similaire on supprime la partie$_1$ de P$_2$ et on centre les trous de la même façon.

3. Insérer la partie$_2$ dans P$_1$ et partie$_1$ dans P$_2$ en remplacement des trous.

P$_1$=[1	2	3	4	5	6	7	8	9]	⇒	P$_1$=[1	t	t	t	5	6	7	8	9]
P$_2$=[7	6	9	4	3	2	1	5	8]		P$_2$=[7	t	9	t	3	2	1	t	8]

Figure II.18: Suppression des parties partie$_1$ et partie$_2$

P$_1$=[1	5	6	t	t	t	7	8	9]		P$_2$=[7	9	3	t	t	t	2	1	8]

Figure II.19: Dégager les trous vers le centre du chromosome.

F$_1$=[1	5	6	4	3	2	7	8	9]		F$_2$=[7	9	3	4	5	6	2	1	8]

Figure II.20: Insertion des parties du chromosome pour générer les individus fils

II.3.7.4.6 Approche de Varela et al.

Cette approche a été proposée pour la résolution du problème de satisfaction des contraintes de Job Shop[Var 98]. Dans ce cas, chaque job est caractérisé par une date de disponibilité et

une date échéante(date limite). Il s'agit donc d'arriver à une solution faisable. La particularité de l'approche proposée par les auteurs est l'utilisation d'une heuristique probabiliste pour générer la population initiale. L'heuristique ordonnée par variable et valeur consiste à calculer pour chaque tâche la probabilité de survie de sa réservation de la machine pour une période donnée. Cette information servira à définir l'ordre de planification des tâches sur les machines.

II.3.7.5 AG et Problème de Routage de véhicules

Le problème de routage de véhicules[Chr 76] se présente comme une extension du problème de voyageur de commerce.

II.3.7.5.1 Approche de Louis, Yen et Yuan

Louis, Yin et Yuan [Lou 99]proposent une approche se basant sur une relation d'ordre globale entre les clients. Cet ordre est réalisé sur la base des fenêtres d'asservissement de chaque client.

Un ordonnancement possible est représenté par un vecteur dont les éléments correspondent aux clients(figure II.21). La procédure de décodage opère par répartition aléatoire des *k*(nombre de véhicules) premiers clients du chromosome sur les véhicules(les véhicules sont identiques). Les *N* - *k* clients restants sont alors distribués sur les véhicules de façon déterministe et répétitive. Suivant l'ordre des clients dans le chromosome, un autre client, non encore planifié, est pris en considération. Les coûts des sous tours, engendrés par l'affectation du client à chacun des véhicules sont évalués. Le meilleur sous tour est retenu et le client est affecté au véhicule correspondant.

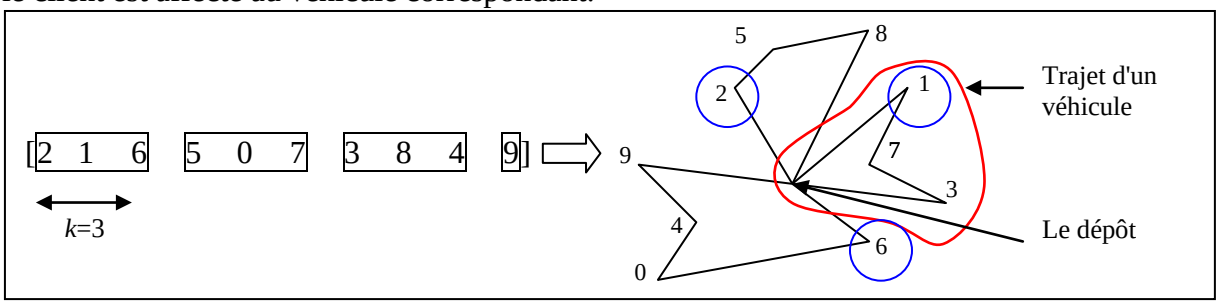


Figure II.21: Représentation en vecteur d'un ordonnancement

L'opérateur de croisement fait intervenir le vecteur de précédence global. Ayant deux chromosomes A et B comme par exemple dans la figure II.22, un seul individu fils AB sera généré. En premier lieu, les deux clients planifiés 2 et 4 seront sélectionnés des deux parents. Puisque 2 précède 4 dans le vecteur de précédence global, alors c'est 2 qui sera planifié en première position dans AB. Une permutation est alors opérée entre 2 et 4 dans B. Ce procédé est réitéré pour les autres paires de clients dans l'ordre de leur apparition dans les deux parents.

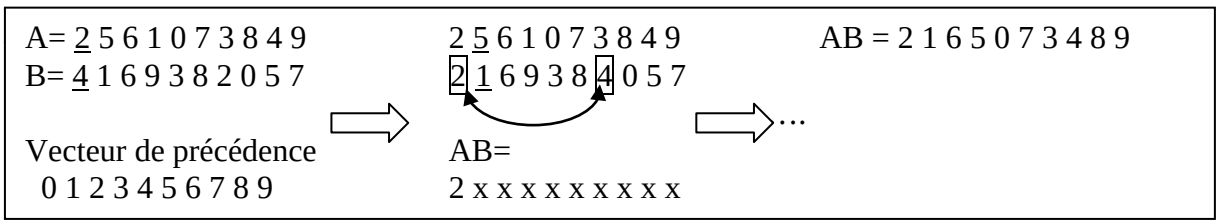


Figure II.22: Opérateur de croisement basé sur le vecteur de précédence

II.3.7.5.2 Notre approche

Dans cette approche [Ben 00] un ordonnancement est représenté par un vecteur de listes associant une entrée pour chaque véhicule. Une liste associée à un véhicule décrit l'ordre d'asservissement des clients par le véhicule en question.

L'opérateur de croisement consiste en l'échange de la distribution des clients sur les véhicules. Le figure II.23 décrit le processus de croisement de deux ordonnancements.

1. Un site de croisement *s* est choisit aléatoirement.

2. Deux ordonnancements fils sont alors générés. Le premier est le produit des *s* listes supérieures du premier parent avec les *k-s* listes inférieures du deuxième parent et inversement pour le deuxième fils.

3. Les deux ordonnancements fils générés sont alors réparés afin d'éliminer la redondance des clients ou insérer ceux disparus.

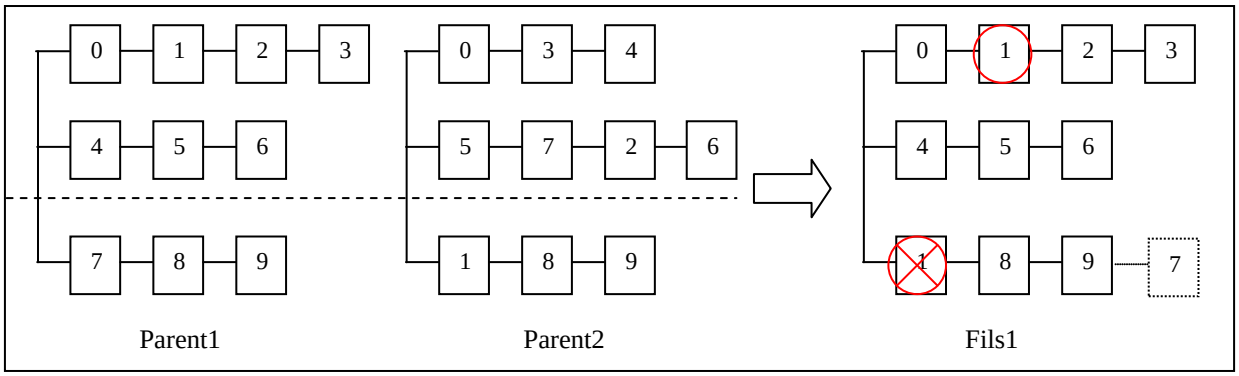


Figure II.23: Notre opérateur de croisement.

II.4 La Recherche Tabou

La Recherche Tabou(RT)[Glo 96] est une méta-heuristique de recherche réputée pour son habilité à échapper aux optimums locaux. Les dix dernières années, ont vu multiplier les travaux sur la RT dû au succès qu'a rencontré la méthode face aux problèmes d'optimisation.

La méthode a été utilisée avec succès en coopération avec d'autres heuristiques et méthodes pour la résolution des problèmes d'ordonnancement[Her 95], d'allocation de ressources[Hou 97] et de télécommunication[Glo 96].

II.4.1 Fondements de la Recherche Tabou

Ayant une fonction *f* à optimiser, la RT procédé par transition itérative d'une solution à une autre jusqu'à ce qu'un critère d'arrêt soit vérifié. A chaque solution S_i est associée un voisinage $V(S_i)$ représentant l'ensemble des solutions directement accessibles depuis S_i par une opération notée *mouvement*. La RT adopte une stratégie de modification du voisinage $V(S_i)$ au fur et à mesure que la recherche progresse, ce qui donne lieu à un nouveau ensemble $V^*(S_i)$. Pour opérer cette modification, la RT s'appuie sur des structures spéciales de mémoire.

La RT est fondée sur les cinq principes suivants:

1. Définir un moyen de construction du voisinage, ce qui revient à déterminer le type de mouvements à adopter.

2. Une règle d'interdiction de certains mouvements ou solutions (considérés *tabou*). Généralement, afin de prévenir les cycles dans la recherche.

3. Une règle de choix d'une solution parmi l'ensemble $V^*(S_i)$.

4. Un critère d'aspiration permettant d'autoriser un mouvement considéré *tabou*.

5. Un mécanisme de diversification servant à opérer de grands sauts dans l'espace de recherche.

II.4.2 La Recherche Tabou en action

La RT procède en démarrant d'une solution initiale, qui est soit générée de façon aléatoire ou en utilisant une heuristique quelconque. Itération après itération, la RT fait passer la recherche d'une solution S_i à une solution S_j via de petites modifications opérées sur les attributs de la solution actuelle*(mouvement)*. Le passage d'une solution à une autre ce fait suivant les étapes suivantes:

II.4.2.1 La génération de voisinage

Ayant la solution S_i, la RT commence par calculer l'ensemble des voisins $V(S_i)$. Certaines des solutions de $V(S_i)$ sont alors éliminées donnant lieu à l'ensemble $V^*(S_i)$. Les solutions admises dans $V^*(S_i)$ sont déterminées de deux manières: soit utilisant la liste tabou, ou en utilisant la mémoire explicite.

II.4.2.1.1 Mémoire attributive (liste tabou)

Lors du passage d'une solution à une autre, les valeurs des attributs qui ont subits des modifications seront rajoutées à la *liste tabou*. Par exemple, pour un problème de Flow Shop, les attributs peuvent correspondre aux positions des jobs modifiés dans la permutation(figure II.24). Ces attributs seront alors considérés comme *tabou* pour une certaine période de temps appelée *tenure tabou.* Lors de la génération de l'ensemble $V(S_i)$, toute solution présentant un attribut compris dans la liste tabou, sera ignorée ou seulement pénalisée.

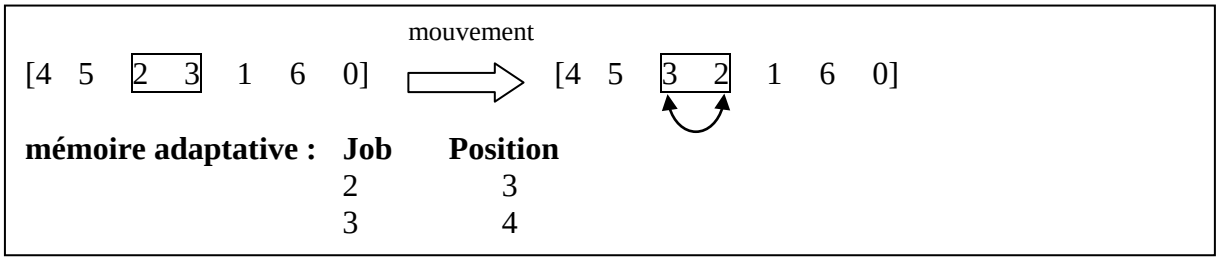


Figure II.24: Liste tabou ou mémoire adaptative

II.4.2.1.2 Mémoire Explicite

Cette mémoire quant à elle contient les meilleures solutions visitées pendant la recherche ou celles dont le voisinage n'est pas exploré. Ces solutions seront insérées à des périodes de temps stratégiques pour élargir l'ensemble $V^*(S_i)$.

Vu que la taille de l'ensemble $V^*(S_i)$ peut être assez grande, des stratégies visant à réduire l'ensemble des mouvements considérés ont été proposées[Glo 95]. Ces stratégies sont connues sous la nomination de stratégies de liste candidate.

II.4.2.2 Critère d'aspiration

Le statut *tabou* d'un attribut n'est pas rigide et peut vêtir plusieurs niveaux de restriction. Il est ainsi possible de surmonter le caractère *tabou* d'une solution quand certaines conditions dites critères d'aspiration sont réunies. Par exemple une solution qui est meilleure que toute autre solution déjà visitée, sera considérée admissible même si elle présente des attributs *tabous*. Une telle solution peut cependant être sujette à une pénalisation en fonction du degré *tabou* quelle présente. Les niveaux *tabous* peuvent être mesurer en fonction de:

1. La période pendant laquelle l'attribut est resté *tabou*.
2. Le nombre d'attributs *tabous* que présente la solution.
3. La nature des critères d'aspiration quelle comporte.

II.4.2.3 Sélection de la solution suivante

Quand l'ensemble des voisins est construit, la RT choisit la meilleure solution présente dans cet ensemble. Que fait donc la RT quand l'ensemble $V^*(S_i)$ est vide? C'est à dire quand l'ensemble des voisins de la solution courante S_i sont tous *tabous* et aucun ne présente un critère d'aspiration. Dans ce cas la RT reconsidère l'ensemble $V(S_i)$ en appliquant de fortes pénalités selon le statut *tabou* des attributs.

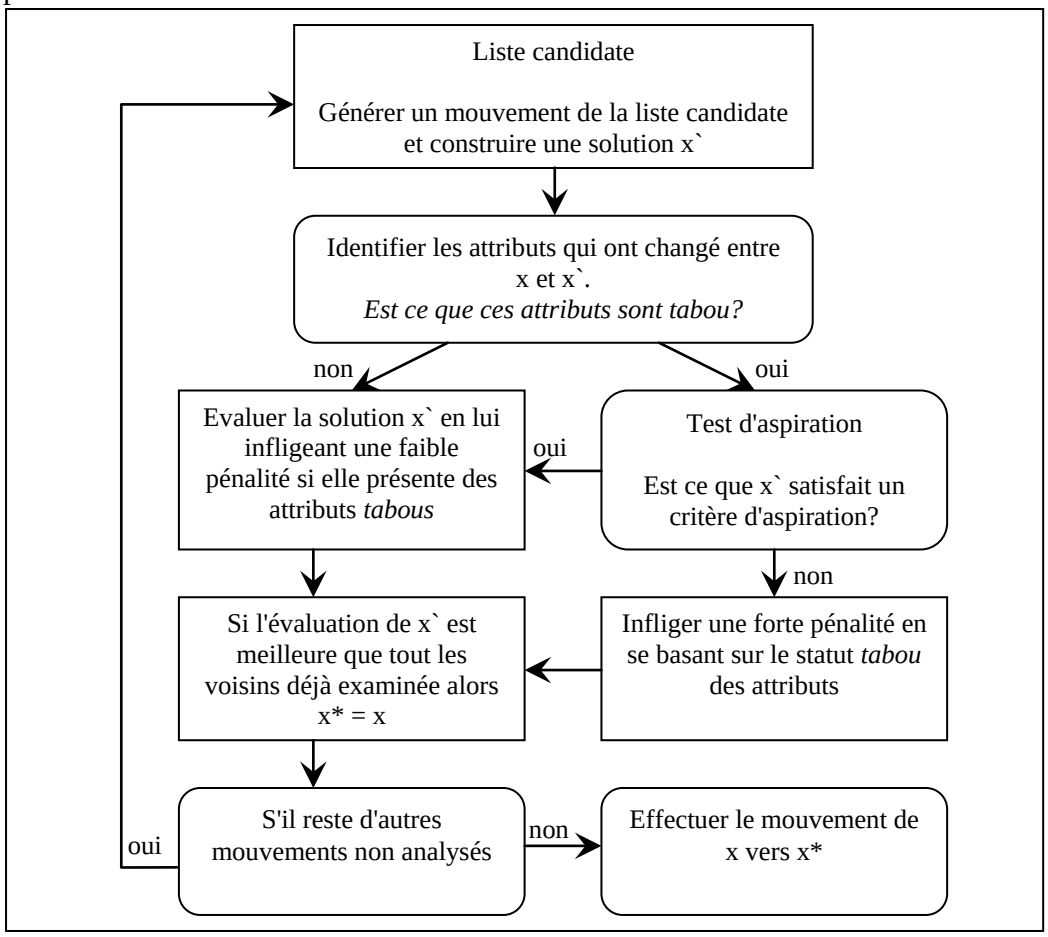


Figure II.25: Fonctionnement d'une itération de la Recherche Tabou

Une fois la nouvelle solution S_j est déterminée, la RT identifie l'ensemble des attributs altérés en passant de S_i à S_j. Ces attributs seront considérés *tabou* (ajouter à la liste *tabou*) pendant un certain nombre d'itérations.

II.4.2.4 Condition d'arrêt

La recherche s'arrête généralement quand un nombre d'itérations limite est atteint. Comme on peut choisir de fixer un nombre d'itérations au bout du quel, si aucune amélioration de la meilleure solution n'est obtenue, on arrête l'investigation. Il est aussi possible de déterminer une borne inférieure f^* de f. On décidera d'arrêter la recherche dès qu'une solution de valeur $f(S_i)$ proche de f^* est trouvée. Le diagramme de la figure II.25 schématise le fonctionnement de la RT pendant une itération.

II.4.3 Analyse des points forts de la Recherche Tabou

A. *Utilisation de mémoire à court et à long terme:* la liste tabou peut être considérée comme une mémoire à court terme. Comme supplément de cette mémoire, la RT emploie des mémoires à long terme pour renforcer son efficacité. Les mémoires fréquentatives représentent bien ce type de structures. Elles procèdent par pénalisation et encouragement de certains attributs. Ceci en maintenant des informations telles que la durée de temps pendant la quelle les attributs apparaissent dans les solutions visitées, ainsi que la façon avec la quelle les attributs changent. *Glover* [Glo 96] soutient que les avantages tirés de l'utilisation de la mémoire à long terme n'apparaît pas nécessairement qu'après une longue période de temps.

B. *Manipulation de données symboliques:* Comme pour l'AG la RT n'est pas contrainte par le caractère entier ou continu des variables manipulées.

C. *Utilisation minimale d'information a priori:* La RT ne requière qu'un moyen d'évaluer la qualité d'une solution.

D. *Recherche intensif:* La stratégie d'intensification est basée sur la modification des règles de sélection afin d'encourager les attributs de solutions historiquement prouvés bons. Ceci peut être réalisé lors de l'étape de génération du voisinage. On pourra alors choisir de n'admettre dans $V^*(S_i)$ que les solutions présentant des attributs spécifiques. L'intensification peut aussi s'opérer par initiation d'un retour aux régions attractives de l'espace de recherche afin de mieux les explorées. Une implémentation de ce principe consiste en la stratégie de sélection élite. Ce procédé se base sur une liste de solutions élites contenant les meilleures solutions trouvées pendant la recherche. *Voss*(1993) introduit une mesure de diversité qui assure que les solutions maintenues dans la liste élite soient assez différentes les unes des autres. *Voss* propose l'effacement de la mémoire à court terme (liste *tabou*) après chaque procédé d'intensification qui consiste à prendre la meilleure solution présente dans la liste élite comme point de recherche ultérieure. L'approche d'intensification proposée par *Niwicki* et *Smutnicki* (1993) utilise une liste séquentielle à taille limitée. Une solution est ajoutée à la fin de la liste seulement si elle est meilleure que toutes celles déjà visitées. Lors d'une intensification, la solution présente à la fin de la liste sera prise pour relancer la recherche. La liste *tabou* dans ce cas est maintenue.

Glover, *Chiu* et *Xu* (1995), maintiennent une liste ordonnée des *k* meilleures solutions. Après un nombre d'itérations déterminé, on parcoure la liste à commencer par la solution la moins bonne. La solution actuellement sélectionnée relancera une nouvelle recherche pendant un certain nombre d'itérations après quoi on passe à une autre solution dans la liste. Pendant ce temps la liste continue à être mise à jour.

Glover [Glo 95]classe ces approches sous l'appellation de *mouvement restructuré*. D'autres approches garde trace des meilleurs voisins non explorés (ceux examinés dans la liste candidate) ou des voisins des optimums locaux.

Un autre type d'approche consiste en *l'intensification par décomposition*, où des restrictions sont imposées sur une partie du problème ou sur la structure de la solution. Un exemple classique est de considérer les attributs communs entre les solutions élites. Ces attributs seront alors fixés dans les solutions ultérieurement.

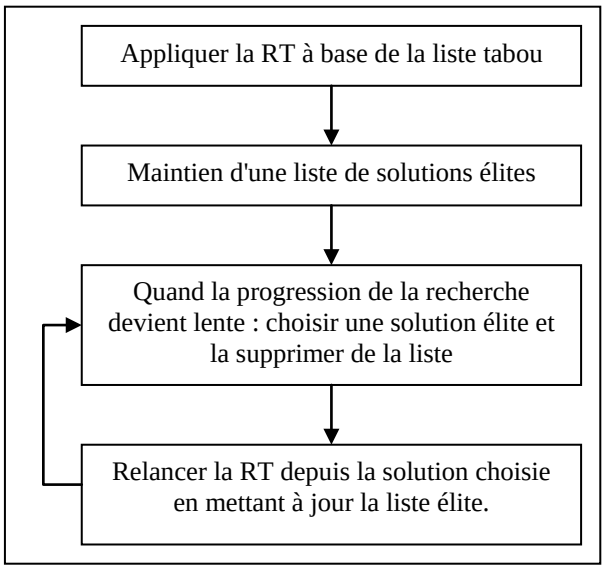


Figure II.26: Stratégie d'intensification

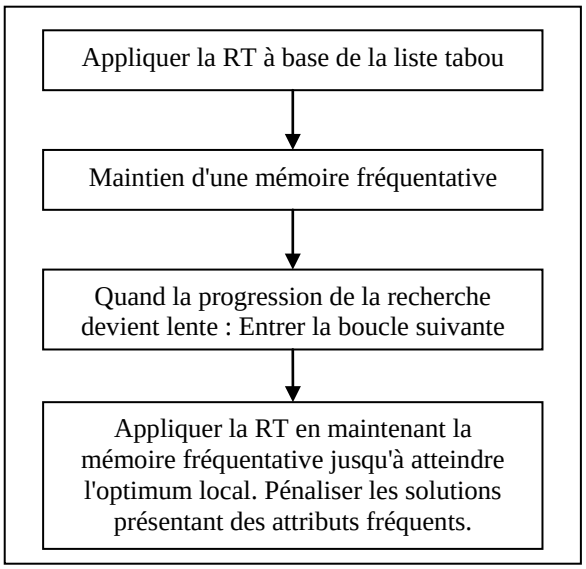


Figure II.27: Stratégie de diversification

E. Introduction de mécanismes de diversification: Werra et *Hertz*[Wer 89] définissent le compromis à tenir entre intensification et diversification par "*Une recherche intelligente, ne doit pas seulement explorer entièrement les régions de bonnes solutions, mais elle doit aussi avoir une vue générale de l'espace de recherche et de s'assurer qu'aucune région n'a été complètement ignorée*". La diversification désigne donc l'ensemble des moyens servant à explorer de nouvelles régions. Souvent ces mécanismes se basent sur une modification des règles de sélection afin d'introduire des attributs rarement utilisés. Ces attributs peuvent être introduit de manière partielle ou totale ou en pénalisant les solutions contenant des attributs fréquemment utilisés. Le moment d'introduire la diversification est souvent cruciale, par exemple lors de l'atteinte d'un optimum local ou après un nombre déterminé d'itérations.

La RT peut aussi se servir de la *tenure* des attributs *tabou* comme moyen de diversification. Quand la progression de la recherche devient lente, la *tenure* des attributs *tabou* est augmentée.

La RT peut aussi être lancée plusieurs fois. Pour empêcher que la recherche aboutisse toujours à la même solution finale, la fonction *f* à optimiser est révisée afin de favoriser les régions non explorées. On peut aussi s'assurer de démarrer avec des solutions initiales équitablement réparties dans l'espace de recherche.

II.4.4 Recherche Tabou Parallèle

La classification des approches de parallélisation de la RT a été opérée de différentes façons selon les critères pris en compte: Le nombre de solutions initiales, Le caractère identique ou variant des paramètres de recherche(taille de la liste tabou, *tenure* des attributs *tabou*, …), stratégies de contrôle et de communication. *Talbi*, *Hafidi* et *Geib* [Tal 98a] classifient ces approches en deux grandes classes: Décomposition de domaine et RT à multiples tâches (figure II.28).

II.4.4.1 Décomposition de domaine

Deux cas peuvent se présenter: soit une décomposition de l'espace de recherche ou de l'ensemble de voisinage.

II.4.4.1.1 Décomposition de l'espace de recherche

Le problème principal est décomposé en plusieurs sous problèmes de plus petites tailles. Chaque sous problème sera traité par un algorithme de RT différent.

II.4.4.1.2 Décomposition du voisinage

La recherche du meilleur voisin est effectuée de façon parallèle, chaque sous ensemble de voisins est évalué par un processus distinct. Une telle approche exige un grand degré de synchronisation entre les différents processus.

II.4.4.2 Recherche Tabou à multiples tâches

Cette approche consiste à lancer en parallèle plusieurs RT. Les différents processus de RT travaillent soit avec les mêmes paramètres ou avec des paramètres différents. Les différents processus peuvent communiquer entre eux, on parle alors de processus coopérants. Comme ils peuvent être indépendants donc non communicants.

Dans les algorithmes de RT coopérative, un processus diffuse la meilleure solution qu'il a trouvée après un nombre d'itérations déterminé. La meilleure solution devient alors la solution initiale pour la prochaine phase.

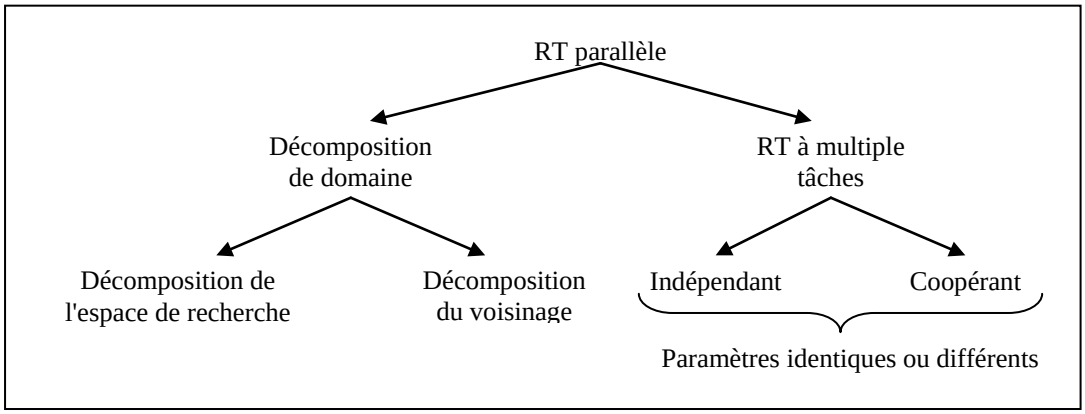


Figure II.28: Classification des algorithmes de RT parallèle

Les approches à base de décomposition de l'espace de recherche ou du voisinage sont des problèmes fortement dépendants. La deuxième classe de parallélisation est moins restrictive et plus générale. La combinaison des deux principes de parallélisation a été proposée par *Badeau* et *al*[Tal 98a].

II.4.4.3 Non adaptative

Cette famille se distingue par la fixation du nombre de processus et leur localisation dès l'étape de compilation(ordonnancement statique). La distribution des processus et des données sur les processeurs reste inchangée pendant toute la durée de l'exécution. La plupart des approches de parallélisation obéissent à ce principe.

Les inconvénients d'une telle approche sont clairs. Premièrement, la rapidité de la recherche est conditionnée par le temps d'exécution final sur toutes les machines(les machines à degré de chargement élevé et celles de faibles puissances). Deuxièmement, l'augmentation du nombre de processus implique souvent des temps d'attente(synchronisation) considérables.

II.4.4.4 Semi-adaptative

Dans ce cas le nombre de processus composant la recherche est fixé dès le départ. Cependant, la répartition des processus sur les processeurs est équilibrée de façon dynamique(lors de l'exécution) en tenant compte des événements qui surviennent sur l'application(fin d'un processus, augmentation de la charge d'un processeur par des processus extérieure à l'application, …). Dans la décomposition du voisinage, l'équilibrage de charge revient à une migration de données entre les processus.

Dans certain cas le degré de parallélisation exigé dépend de l'état de la recherche. Un nombre de processus élevé, fait que plusieurs processus effectuent la même tâche. Par contre, un nombre insuffisant fait perdre des opportunités de parallélisation.

II.4.4.5 Adaptative

Dans un algorithme adaptatif, les processus sont créés et détruits dynamiquement selon le degré de parallélisme requis. La répartition des processus sur les processeurs est révisée pendant l'exécution de l'application pour tenir compte des inégalités de charge entre machines.

II.4.5 Recherche Tabou et Problèmes d'ordonnancement

Les problèmes d'ordonnancement sont avérés être un domaine où la RT excelle [Glo 95][Her 95]. Dans ce qui suit nous allons passer en revue des exemples de l'application de la méthode de RT au problème de satisfaction de contraintes, de flow shop, de job shop et de routage de véhicules.

II.4.5.1 Recherche Tabou est problème de satisfaction de contraintes

Hao et *Pannier*[Hao 98] proposèrent un algorithme de RT pour la résolution du PSC. Cet algorithme se base sur les trois points suivants: évaluation d'une configuration, la liste tabou et le critère d'aspiration.

1. *Evaluation d'une configuration*: Les auteurs utilisent une matrice δ de dimension |V|*|D|. Les lignes de la matrice désignent les variables et les colonnes les valeurs possibles de celles-ci. La valeur δ[*v,d*] est calculée lors de chaque itération une seule fois et correspond au coût induit par l'assignation de la valeur *d* à la variable *v* (c'est à dire la somme pondérée des violations et des satisfactions de contraintes engendrées par ce changement dans la solution actuelle *S*). L'évaluation des voisins de *S* est alors réalisée par une simple opération d'addition de *f*(*S*) et des éléments correspondants dans δ. Après le mouvement(choix du meilleur voisin), la matrice est mise à jour.

2. *Liste Tabou*: Etant donné qu'un mouvement est identifié par un couple (V_i, v_i), V_i et v_i font référence à la variable altérée et la valeur qui lui est assignée. Quand un mouvement fait passer la valeur d'une variable V_i de v_i à v'_i, le couple (V_i, v_i) est rajouter dans la liste tabou afin d'empêcher qu'un autre mouvement ne restaure cette valeur de nouveau pour les *k* itérations futures. *Hao* et *Pannier* implémentent la liste tabou comme une matrice *T* de dimension |V|*|D|. L'élément *T*[*i, j*] correspond à un mouvement possible (V_i, v_j). Quand le mouvement (V_i, v_j) est entrepris, *T*[*i, j*] est mis à jour pour indiquer la *tenure tabou* de l'attribut.

3. *Critère d'aspiration:* Le statut *tabou* d'un attribut est surmonter si le mouvement engendre une solution qui est meilleur que toute celles déjà visitées. Dans ce cas l'élément *T*[*i, j*] est remis à zéro.

II.4.5.2 Recherche Tabou est problème de Flow Shop

La RT a souvent été appliquée au problème de Flow Shop. Nous citerons dans ce qui suit les approches dues à *Widmer* et *Hertz*(1989), *Diaz*(1992), *Ben-Daya* et *Al-Fawzan* (1998), *Niwicki*(1999).

II.4.5.2.1 Approche de Widmer (SPIRIT)

Widmer et *Hertz*[Her 95] définissent le voisinage d'une solution *S, V*(*S*) comme étant l'ensemble des permutations obtenues à partir de *S* en échangeant les positions de deux jobs non nécessairement adjacents.

1. *Solution Initiale: Widmer* et *Hertz* adoptent l'algorithme d'insertion pour générer la solution initiale d'où débute la recherche. Cet algorithme s'inspire des techniques appliquées au problème de voyageur de commerce. Une distance entre les jobs est définie de la façon suivante:

$$dist(J_i, J_j) = d_{i1} + \sum_{k=2}^{m} (m-k) \times \left| d_{ik} - d_{j,k-1} \right| + d_{jm} \qquad \text{(II.5)}$$

   L'algorithme consiste à déterminer les deux jobs *k* et *l* les plus proches l'un de l'autre. Ces deux jobs constitueront le chemin $J_k \rightarrow J_l$. La suite de l'algorithme consiste à choisir chaque fois un job aléatoirement parmi les jobs non encore planifiés. Ce job sera inséré dans le chemin de façon à engendrer un chemin de longueur minimale.

2. *Evaluation*: La fonction objectif correspond au *makespan,* le meilleur voisin est donc celui présentant le meilleur temps d'accomplissement total.

3. *Liste Tabou:* Lors du passage d'une solution *S* à une solution voisine *S'*, deux positions *i* et *j* sont choisies aléatoirement et les jobs *x* et *y* se trouvants à ces positions sont permutés. Les deux couples (*x*, *i*) et (*y*, *j*) sont rajoutés à la liste tabou, de sorte que tout mouvement pouvant remettre *x* à la position *i* ou *y* à la position *j* est interdit pendant un certain nombre d'itérations.

4. *Choix des paramètres*: la *tenure tabou* des attributs a été fixée à 7. Le nombre d'itérations limite sans amélioration a été fixé à *N+M*.

II.4.5.2.2 Approche de Diaz

Diaz [Dia 92] propose un algorithme à base de Recherche Tabou pour la résolution du problème de Flow Shop où le but est de minimiser la somme pondérée des retards. L'intérêt de l'approche est l'utilisation d'un mécanisme de *restriction de voisinage* visant à réduire le temps d'une itération. Comme pour *Widmer* et *Hertz*, le voisinage d'une solution correspond à l'ensemble des permutations nées de l'échange de deux jobs. Selon ce procédé, lors de chaque itération un nombre $N(N-1)/2$ de voisins est examiné.

Diaz définit le *rang de transposition* d'un mouvement comme étant la valeur tr(s→s')=|*i* - *j*|, *i* et *j* correspondent aux positions des jobs échangés. L'étude réalisée par *Diaz* démontre que lors des étapes primaires de la recherche la distance $|i-j|$ est élevée. Cette distance commence à se détériorer dès que la recherche avance jusqu'à ce que seule les jobs adjacents soient échangés. *Diaz* propose alors de restreindre les mouvements examinés lors d'une itération *it* à seulement ceux dont le *rang de transposition* est inférieur à la valeur de la fonction *ra*(*it*)[Dia 92].

$$ra(it) = [N\, e^{(it-ret)/k'}]_4^{N-1} \qquad \text{(II.6)}$$

ret désigne le nombre d'itérations initiales pendant lesquelles le voisinage entier est examiné.

$k' = estab/\ln(4/N)$, où *estab* désigne le nombre d'itérations après lequel nous supposons que nous n'aurons pas besoin de *rang de transposition* supérieur à 4.

II.4.5.2.3 Approche de Ben-Daya et Al-Fawzan

Les apports de la méthode[Ben 98] se résument aux points suivants

1. *Solution initiale*: *L*es deux heuristiques: *NEH* (*Paragraphe* I.11.3.1.2) et insertion(Approche de *Widmer* et *Hertz*) ont été implémentées. Cependant, les tests comparatifs montrèrent que l'utilisation de l'heuristique *NEH* est plus efficace.

2. *Génération de voisinage*: Trois types de mouvement sont implémentés. Lors de chaque itération l'un des trois mécanismes de génération de voisinage est sélectionné. Le premier agit en permutant deux jobs quelconques. Le deuxième consiste à choisir deux jobs de position *i* et *j*. Le job de la position *j* est alors inséré à la position *i*. La troisième méthode dite d'insertion de bloc opère par choix aléatoire de deux jobs de position *i* et *j* et d'une taille *k* du bloc. Le bloc des *k* jobs débutant à la position *j* est inséré à la position *i*. Pour ne pas générer l'ensemble de tous les voisins ont s'arrête dès qu'on trouve une solution voisine qui améliore la solution courante.

3. *Critère d'aspiration*: Un mouvement est accepté malgré son statut *tabou* s'il est susceptible d'engendrer une solution meilleure que toutes celles déjà vues.

4. *Intensification et diversification*: Les auteurs proposent l'utilisation d'une mémoire fréquentative comme mécanisme de renforcement de l'intensification et de la diversification. La mémoire fréquentative est une matrice *F(N*N)*. Les lignes correspondent aux différents jobs et les colonnes aux positions possibles. *F*[*i, j*] désigne le nombre de fois que le job J_i a été planifié à la position *j*. Le procédé d'intensification et de diversification consiste à relancer la recherche à partir d'une nouvelle solution initiale construite en se basant sur la matrice *F* de la manière suivante:

1. *k*=1.
2. Déterminer l'entrée *F*[*i, j*] ayant la valeur maximale.
3. Assigner le job J_i à la position *j.*
4. Remettre à zéro la ligne *i* et la colonne *j*.
5. *k*=*k*+1.
6. Si *k*>*N* alors la solution est complètement construite sinon revenir à l'étape (b).

5. *Critère d'arrêt:* l'algorithme est arrêté quand la solution n'est pas améliorée entre deux appels de la procédure d'intensification/diversification.

II.4.5.3 Recherche Tabou et problème de Job Shop

II.4.5.3.1 Approche de Colorni, et al

La méthode [Col 97] se base sur une représentation à base de machines des ordonnancements possibles.

1. *Génération du voisinage:* un mouvement consiste à changer la position *i* d'une tâche dans la machine vers une autre position. Le mouvement est décomposé(figure II.28) en une séquence d'inversements d'ordre(*x, i*).

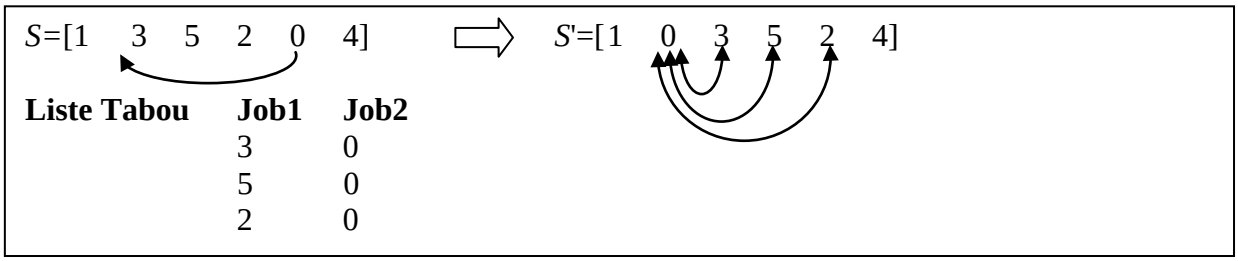


Figure II.29: Décomposition d'un mouvement en différents inversements d'ordre

2. *Liste tabou:* A chaque machine est associée une matrice *T* de dimension *M*M*. L'élément *T*[*i, j*] détermine si un mouvement susceptible de faire précéder la tâche *i* par la tâche *j* est interdit ou pas.
3. *Solution initiale:* Les auteurs applique une variante aléatoire de l'algorithme de recherche locale(algorithme glouton).

II.4.5.3.2 Approche de Pezzella, Merelli

Pazzela et *Merelli*[Paz 00] font observer que la qualité de la solution initiale est un facteur primordial dans l'efficacité de la recherche. L'heuristique basée sur *l'étouffement* est utilisée afin de garantir une bonne solution de départ.

1. *Génération de voisinage:* les voisins d'une solution sont construits par inversement d'un arc dans le chemin critique du graphe disjonctif représentant la solution(figure II.30 et 31).

2. *Liste Tabou:* Lorsque la recherche passe d'une solution à une autre, l'arc représenté par le couple des tâches permutées est inséré dans la liste tabou. La taille de la liste tabou varie dynamiquement en fonction de l'état de la recherche(état de progression ou de stagnation) et du nombre d'itérations actuel.

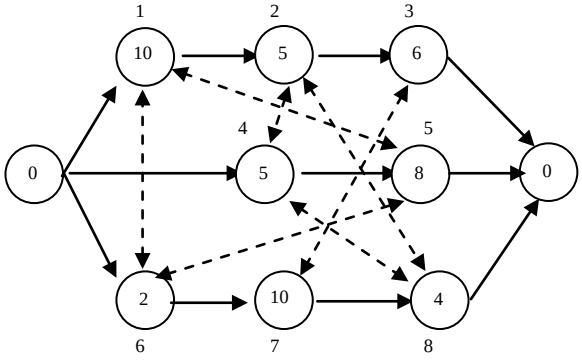


Figure II.30: graphe disjonctif

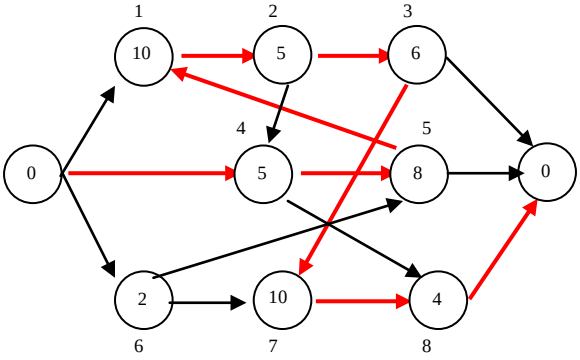


Figure II.31: Chemin critique dans une solution faisable

3. *Procédure de réoptimisation:* Quand la solution actuelle est meilleure que toute autre solution déjà visitée, la recherche procède à un ré-arangement local de cette solution. La procédure de réoptimisation consiste en un ré-ordonnancement des tâches appartenant au chemin critique.

II.4.5.4 Recherche Tabou et problème d'emploi du temps

II.4.5.4.1 Approche de Hertz

Hertz[Her 91] se pencha sur l'élaboration d'une méthode de résolution générale du problème d'emploi du temps. Le problème est décomposé en deux sous problèmes. Le premier, consiste à planifier les cours de façon à prévenir les conflits sur les salles et les enseignants. Le deuxième, revient à regrouper les étudiants au sein des sections de cours. L'approche tente de prendre en compte l'ensemble de toutes les contraintes possibles(cours successifs, cours distancés, …).

Dans cette approche les deux sous problèmes sont considérés en tant que problèmes d'assignement. Pour le problème de planification des cours il s'agit d'assigner à chaque cours une date de début. Dans le problème de regroupement des étudiants, il s'agit d'assigner à chaque étudiant une section de cours convenable.

1. *Recherche Tabou pour le problème d'emploi du temps:* Une semaine est répartie en un ensemble de périodes ouvrables et nous devons obligatoirement assigner à chaque cours une période valable. Nous devons tenir compte de la disponibilité des enseignants, des salles et les contraintes de préassignement. Delà, chaque cours x possède un ensemble $a(x)$ de périodes de début possibles. Un ordonnancement faisable est une solution ne présentant aucun conflit de nature différente de:

- Conflits sur les enseignants.
- Conflits sur les étudiants.
- Cours non assez distancés (donner du temps aux étudiants et enseignants pour se déplacer entre les différentes salles).

Pour chaque cours x nous pouvons maintenant définir le sous ensemble $p(x) \subseteq a(x)$ contenant toutes les périodes t de début pouvant être assignées à x (laissant les autres assignements inchangés) en maintenant l'ordonnancement faisable. Les sous ensembles $p(x)$ sont mis à jour lors de chaque itération de la RT. Le voisinage $V(S)$ d'une solution S englobe toutes les solutions pouvant être générées par le changement de l'assignement fait à un cours x vers une nouvelle période $t' \in p(x)$.

Nous définissant maintenant les deux fonctions $d(x, y)$ et $e(x, y)$ comme étant le nombre d'enseignants (respectivement des étudiants) partagés par les deux cours x et y. Nous considérons aussi la fonction $o(x, y)$ définie comme suit:

$$o(x,y) = \begin{cases} p_1 & \text{si n} = 2 \\ p_2 & \text{si n} = 1 \\ p_3 & \text{si n} = 3 \end{cases} \qquad \text{(II.7)}$$

n désigne le nombre de cours optionnelles dans x et y. Les paramètres p_1 , p_2 et p_3 sont à spécifier par l'utilisateur afin de déterminer les poids relatifs des conflits. La valeur $o(x, y) \times e(x, y)$ représente le coût d'un conflit possible sur les étudiants entre x et y. La valeur $p_4 \times d(x, y)$ désigne le coût d'un conflit sur les enseignants, où p_4 représente le poids d'un tel conflit.

Soit $b(x, y)$ une fonction binaire prenant la valeur 1 si les deux cours x et y se déroulent dans deux salles éloignées et 0 sinon. Le coût induit par le laps de temps à laisser aux étudiants et enseignants pour se déplacer entre salles est égal à :

$$b(x,y)(p_5 d(x,y) + p_6 e(x,y)) \qquad \text{(II.8)}$$

Soit $l(x)$ la fonction qui représente l'étendu du cours x(le nombre de périodes consécutives) et U_t l'ensemble des cours assignés à la période t. Assigné la période t au cours x signifie que x appartiendra aux ensembles U_t, $U_{t+1} \ldots U_{t+l(x)-1}$.

Soit I l'ensemble des périodes t dont le laps de temps entre t et t+1 n'est pas suffisant pour se déplacer entre les salles distantes.

La fonction objectif f doit inclure tous les conflits possibles dépendant des assignements des cours aux périodes.

$$f(S) = \sum_{t=1}^{nb\_pér} \sum_{\substack{x \neq y \\ x,y \in U_t}} \left(p_4 d(x,y) + o(x,y) e(x,y)\right) + \sum_{t \in I} \sum_{\substack{x \in U_t \\ y \in U_{t+1}}} b(x,y)\left(p_5 d(x,y) + p_6 e(x,y)\right) \qquad \text{(II.9)}$$

2. *Recherche tabou pour le problème de regroupement des étudiants:* Dans le problème de groupement il n'est pas permis de changer l'assignement des cours aux périodes. Seule le regroupement des étudiants au sein des cours peut être révisé. La fonction objectif pour ce

sous problème est identique au premier. Mais, alors que dans le sous problème d'emploi du temps les sous ensembles U_t étaient variables, et les valeurs des fonctions *d* et *e* étaient fixées. Ici seule les valeurs de *e* sont variables. Les conflits dus aux enseignants ne peuvent être résolus pendant cette phase. Généralement le nombre d'étudiants dans chaque cours est préfixé. Seule les solutions conformes à ces règles sont considérées faisables.

Le voisinage d'une solution englobe tous les regroupements générées par échange de deux assignements (z_1, x, t_1) et (z_2, x',t_2) où $t_1 \neq t_2$ et x et x' appartiennent à la même unité d'enseignement. Le triplet (z, x, t) indique que l'étudiant z a été assigné au cours x à la période t. Ce qui revient à permuter deux étudiants.

II.4.5.4.2 Approche de Costa

Dans l'approche de *Costa*[Cos 94] les regroupements des étudiants dans les classes est préfixé. Le problème consiste alors à assigner une période de début à chaque cours.

1. *Génération de la solution initiale:* Pour générer un emploi du temps faisable les cours sont triés selon leur recevabilité. La recevabilité d'un cours correspond au nombre de périodes en lesquelles il peut être programmé en considérant les contraintes de préassignement, de conflit sur les enseignants et sur les salles. Les cours sont alors programmer l'un après l'autre à commencer par le cours le moins recevable. Le cours est affecté à la période générant le moins de conflits.

2. *Génération du voisinage:* Le voisinage d'un emploi du temps *T* englobe tous les emplois du temps faisables qui peuvent être engendrés en changeant la période de début d'un seul cours.

3. *Liste tabou:* Pour prévenir les cycles lors de la recherche, le couple (x, t_1) est introduit dans la liste tabou lorsque la RT fait passer la recherche de l'emploi du temps T_1 à T_2 en modifiant la période du cours x de t_1 vers t_2.

II.4.5.5 Recherche Tabou et problème de routage de véhicules

Plusieurs travaux ont porté sur l'application de la RT pour le problème de routage. Nous décrivons ici l'approche adoptée par *Backer* et *Furnon* [Bac 98]. Une solution du problème est représentée par un graphe orienté décrivant les différents tours effectués par les véhicules.

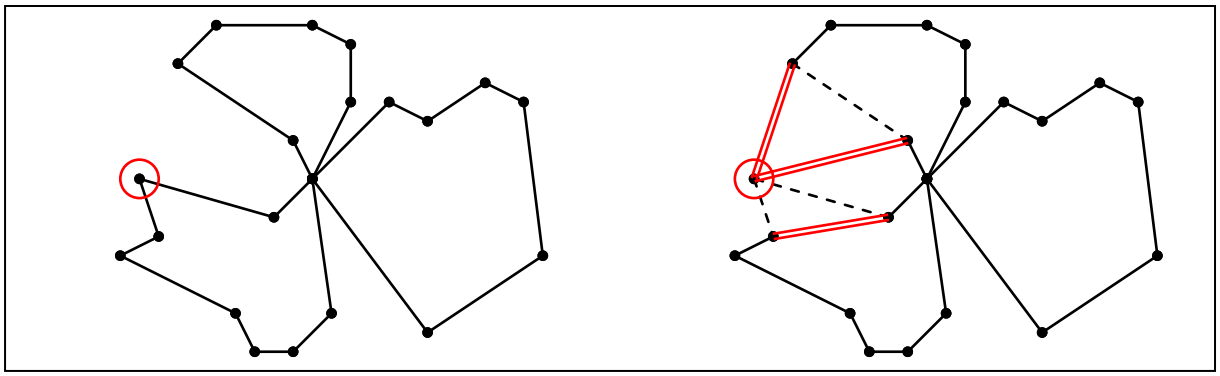


Figure II.32: mouvement de réallocation: arc supprimé arc ajouté

1. *Génération du voisinage:* un mouvement caractérisé par une réallocation d'un client à un autre véhicule. Le mouvement de réallocation est constitué d'une séquence d'ajout d'arcs *AA* et de suppression d'arcs *SA*. Plus précisément il s'agit d'ajouter trois arcs et de supprimer trois autres(figure II.32).

2. *Liste tabou:* Deux listes tabous sont maintenues, l'une stocke les arcs supprimés *LSA* et l'autre les arcs ajoutés *LAA*. Un mouvement de réallocation est considéré tabou s'il vérifie la condition suivante : $card(AA \cap LAA) + card(SA + LSA) > seuil$. Le paramètre *seuil* sert à réguler le nombre de mouvements à considérer tabou. En effet, il sera exagéré de considérer tabou toute réallocation ayant un seul arc tabou.

3. *Stratégie de diversification:* Un entier est associé à chaque arc identifiant le nombre de fois que l'arc a été utilisé dans la recherche. Les solutions présentant un tel arc sont alors pénalisées.

II.5 Recuit Simulé

La méthode du Recuit Simulé (RS)[Sia 95][Ans 97] a été proposée en 1982 par des spécialistes de physique statistique, qui étudiaient les configurations d'objets magnétiques désordonnés de basse énergie, regroupés sous terme de "verres de spin". L'identification numérique de telles configurations soulevait un véritable problème d'optimisation. Cette difficulté est due à l'aspect multi-modales du paysage d'énergie d'un verre de spin. S. *Kirkpatrick* et *al* s'inspirèrent de la technique du recuit utilisé, par les métallurgistes, pour obtenir un état solide bien ordonné, d'énergie minimale. Le procédé consiste à porté à l'état liquide le matériel, puis à le refroidir lentement.

La méthode du recuit simulé a prouvé son efficacité dans des domaines aussi divers que la conception de circuit électronique, le traitement d'images, comme elle s'est montrée trop gourmande ou inadaptée pour d'autres problèmes[Sia 95].

II.5.1 Fondements du recuit simulé

Comme pour la Recherche Locale, le RS opère par transition itérative d'une solution à une autre. Une solution ou configuration selon la terminologie du recuit correspond à une collection de variables(atomes). Dans la physique du recuit, il s'agit de trouver le meilleur positionnement des atomes afin de minimiser l'énergie du système. Par analogie dans un problème d'optimisation il s'agit de trouver la meilleure combinaison des valeurs des variables afin d'optimiser la valeur de la fonction objectif *f*. La figure II.33 montre l'analogie entre problème d'optimisation et certains phénomènes physiques.

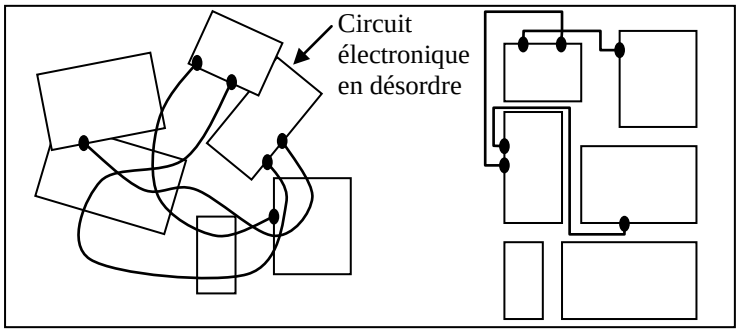


Figure II.33: analogie entre problème d'optimisation et réorganisation des atomes dans les matériaux

L'application du RS pour la résolution d'un problème nécessite

1. Représentation Informatique d'une configuration donnée incluant l'ensemble des variables rentrant dans la codification d'une solution du problème

2. La fonction objectif à optimiser représentant l'énergie du système.

3. Définition du moyen de transition d'une configuration à une autre, l'équivalent en quelque sorte de la notion de mouvement dans la RT.

4. Détermination de la valeur des paramètres de la recherche tel que la température initiale, le pas de refroidissement, …

II.5.2 Recuit simulé en action

L'algorithme de recherche du RS, principalement dû à *Metropolis*[Sia 95][Ans 97], est un processus itératif de transition d'une solution à une autre partant d'une configuration initiale. La configuration initiale peut, soit être généré de façon aléatoire ou en utilisant une heuristique pour garantir une bonne qualité de solution de départ. La température ambiante est fixée à une valeur T_0. Chaque itération consiste en deux étapes principales:

1. *La modification de configuration courante:* on fait subir à la configuration actuelle une petite perturbation, puis en évalue l'énergie de la nouvelle configuration.

2. *Acceptation de la nouvelle configuration:* Deux situations peuvent se présenter. Soit que la perturbation génère une solution d'énergie plus basse($\Delta E \leq 0$)(une meilleure solution au sens de la fonction objectif) et dans ce cas la nouvelle configuration est acceptée et devient de ce fait la nouvelle configuration actuelle. Quand par contre la nouvelle solution est moins adaptée que l'actuelle elle sera acceptée suivant une probabilité de $e^{(-\Delta E/T)}$. Le rôle de la température est compréhensible dans la formule. Une haute température fait tendre la probabilité d'acceptation de la dégradation de la solution à 1(événement sûr). Une température basse par contre fait tendre la probabilité à 0 ce qui fait qu'on tolère de moins en moins la dégradation de la solution.

   Le RS se comporte comme un algorithme de descente classique qui autorise de temps en temps la dégradation de la solution ce qui permet à la recherche de s'extraire des optimums locaux. Après chaque *long* itérations, la température ambiante est abaissée légèrement avec un certain taux défini comme paramètre de la recherche.

L'algorithme du recuit simulé est décrit dans la procédure suivante:

1. Générer la configuration initiale S_0
2. $T=T_0$
3. Tant que le critère d'arrêt n'est pas atteint
   Faire pour i=1 jusqu'à *long*
   Faire
   a. Modifier la configuration courante et générer une nouvelle configuration S'.
   b. Si (f(S')- f(S) ≤ 0) alors accepter S' comme nouvelle solution courante
   c. Sinon - Choisir une nombre p aléatoire compris dans l'intervalle [0, 1]
   - Si p< $e^{(-\Delta E/T)}$ alors accepter S'
   - Sinon maintenir S comme solution actuelle.
   Fait
   Abaisser la température ambiante

Fait.

II.5.3 Analyse des points forts du Recuit Simulé

L'algorithme du RS a suscité de nombreux travaux théoriques pour les deux raisons suivantes[Sia 95]: Premièrement, il s'agissait d'un nouvel algorithme, dont il était nécessaire d'établir les conditions de convergence. Deuxièmement l'algorithme comportait de nombreux paramètres dont il fallait comprendre le mécanisme pour obtenir une efficacité maximale.

A. *Convergence du Recuit simulé:* De nombreuses études faites par des mathématiciens ont tenté de poser les traits principaux de l'analyse de la convergence du RS. *Siarry* souligne que le principal résultat de ces travaux est : "*sous certaines conditions, le recuit simulé converge en probabilité vers un optimum global, en ce sens qu'il permet d'obtenir une solution arbitrairement proche de cet optimum, avec une probabilité proche de l'unité. ce résultat est, en soi, important, car il différencie le RS d'autres heuristique concurrentes, comme la RT, dont la convergence n'est pas garantie"*.

B. *Espace des configurations:* La topologie de l'espace de recherche résulte de la notion de proximité entre configurations, la quelle est mesurée en considération du nombre de transformations élémentaire pour passer d'une solution à une autre. Plusieurs chercheurs se sont penchés sur l'étude de la relation entre la convergence du recuit simulé et les propriétés de l'espace de recherche. Les travaux dus à *Sorkin* montrèrent que certaines propriétés fractales du paysage de recherche, induisent une convergence polynomiale du recuit simulé.

C. *Règle d'acceptation:* Le RS se base sur une règle d'acceptation qui fait qu'occasionnellement des solutions moins adaptées sont admises. Ce qui permet à la recherche d'échapper aux optimums locaux. Il intéressant de signaler que la règle de *Metropolis* n'est pas la seule variante de ce procédé et que d'autres techniques peuvent se montrer plus efficaces.

   D'autres améliorations sur le critère d'acceptation tente de remédier au faite qu'à basse température le taux d'acceptation devient faible au point de rendre l'algorithme moins efficace.

II.5.4 Recuit simulé parallèle

La lenteur d'exécution et l'un des inconvénients reprochés à la méthode du RS. La parallélisation de l'algorithme s'inscrit dans la voie de réduire ce temps de calcul. Malgré la structure séquentielle de la méthode, plusieurs approches de parallélisation ont été envisagées[Sia 95].

II.5.4.1 Approche par subdivision des paliers de température

En considérant K comme étant le nombre de paliers de température, ce procédé consiste à subdiviser chaque palier de température en K étapes. Chaque étape est de $long/K$ itérations. Le premier processus entame la première étape du premier palier. Une fois terminée, il lance le calcul des paramètres d'entrée du deuxième processus. Le deuxième processus exécute l'algorithme du RS pour la première étape du deuxième palier. Pendant ce temps le premier processus continue de traiter les autres étapes du premier palier(figure II.34).

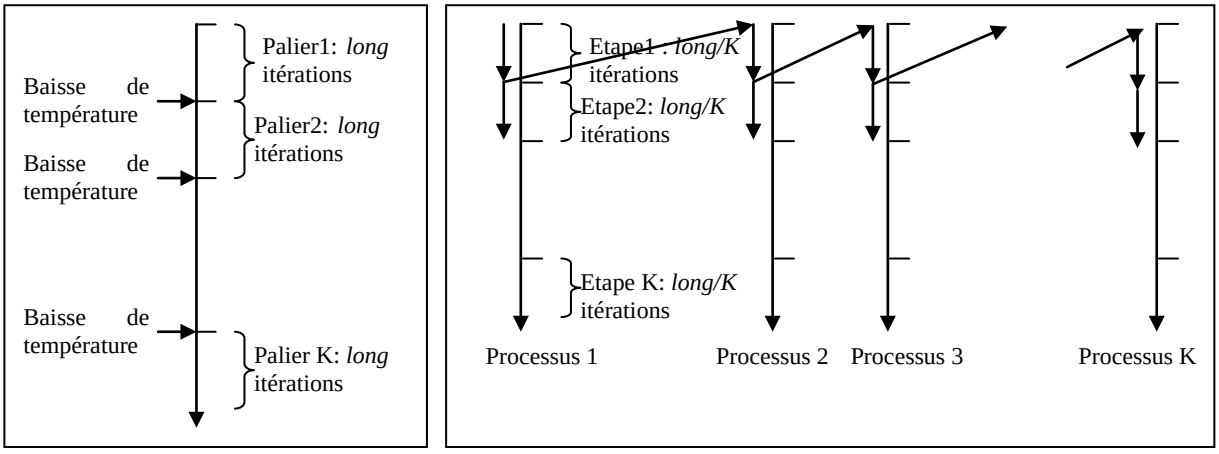


Figure II.34: Approche de parallélisation par subdivision des paliers de température.

II.5.4.2 Approche par production de mouvements en parallèle

Ce procédé[Sia 95] se base sur le faite qu'une fois la température ambiante est basse, la réfutation de mouvements devient plus fréquente. Il est donc possible de considéré ces différentes modifications comme des événements indépendants produits par des processus parallèles. L'algorithme est vu comme un ensemble de processus élémentaires chargés pour chacun d'eu de mesurer la variation d'énergie d'une ou plusieurs modifications. Deux cas peuvent alors se présenter:

- Laisser l'ensemble des processus tester différents mouvements. Un mouvement est choisit de façon aléatoire parmi ceux qui sont acceptés. L'intérêt de ce mode de fonctionnement apparaît lorsque le nombre de modifications acceptables est grand(haute température).

- Mettre à jour la solution courante dès que l'un des processus aboutit à une solution acceptable. Ce procédé est utilisé quand la température ambiante est faible entraînant la rareté des mouvements acceptables.

II.5.4.3 Approche multiple recuit simulé avec échange de solutions.

L'approche proposée par *Ram, Sreenivas* et *Subramaniam*[Ram 96] se base sur la génération de différentes solutions initiales. Un algorithme de RS est exécuté sur chaque solution. Après un nombre d'itérations déterminé, un échange des résultats partiels obtenus par chaque processus est réalisé afin de dégager la meilleure d'entre elles. La totalité des processus reprennent leur recherche, démarrant de la meilleure solution partielle. De nouveau, après un nombre d'itérations déterminé l'algorithme procède à un autre échange, ainsi de suite.

II.5.5 Méthode du Recuit Simulé et problèmes d'ordonnancement

L'application des méthodes inspirées du recuit simulé s'est révélée assez simple[Fle 95] et extensible à une large gamme de problèmes d'ordonnancement. Nous présentons ici un exemple d'application de la méthode au problème de satisfaction de contraintes.

II.5.5.1 Recuit Simulé et problème de satisfaction de contraintes

L'application du RS pour le problème de satisfaction de contraintes se retrouve dans [Ric 96][Hao 98].Une configuration est codée par un ensemble de couples définissant la valeur que prend chaque variable. Une modification consiste à changer la valeur d'une variable donnée. La fonction objectif est désignée par la somme pondérée des violations des contraintes, et elle est mathématiquement formulée par:

$f(S)=\sum_{i=1}^{|C|} p_i \times SAT(C_i)$, $C_{\text{i}} \in C$ ensemble des contraintes, p_{i} le poids de C_{i}.

$$SAT(C_i)=\begin{cases}1 \text{ si } C_i \text{ est violée}\\ 0 \text{ sinon}\end{cases} \qquad \text{(II.10)}$$

Après chaque modification d'une solution la différence *f*(*S'*) - *f*(*S*) est calculée en n'examinant que les contraintes mettant en jeu la variable touchée par la modification. Ce qui induit une complexité de calcul de l'ordre de (|V|×|D|) dans le pire des cas. *V* désigne l'ensemble des variables et *D* l'ensemble des valeurs possibles.

Hoa et *Pannier* [Hoa 98] proposent de réduire la température ambiante T_i suivant la formule suivante : $T_{i+1}=T_i \times \left(1-\frac{A}{i}\right)$. "*A*" désigne un paramètre de contrôle et *i* le numéro de l'itération en cours.

Richards, *Li*, *Jiang* et *Azarmi*[Ric 96] proposèrent la subdivision de l'espace de recherche en plusieurs sous espaces. L'algorithme sauvegarde l'optimum local trouvé dans chacun des sous espaces. La température ambiante et le critère d'acceptation varient d'une région à une autre selon la valeur de l'optimum local. Ceci à pour effet de réguler la vitesse de recherche selon le relief du sous espace.

II.5.5.2 Recuit simulé et emploi du temps

Abrameson [Abr 91] proposait un algorithme à base de RS pour la résolution du problème d'emploi du temps. Un emploi du temps est représenté par un vecteur de listes. Chaque liste désigne l'ensemble des cours affecté à une même période (voir figure II.6). En supposant que les locaux sont de même type, un emploi du temps *T* est évalué suivant la formule suivante:

$$f(T)=\sum_{p\in \text{périodes}}\left(\begin{array}{l}\max\left(0,\ \sum_{t\in professeurs}\left(\left|\text{cours assurés par } t \text{ programmés à } p\right|-1\right)\right)+\\ \max\left(0,\ \sum_{c\in classes}\left(\left|\text{cours de } c \text{ programmés à } p\right|-1\right)\right)+\\ \max\left(0,\left|\text{cours programmés à } p\right|-L\right)\end{array}\right) \qquad \text{(II.11)}$$

le symbole $\left|\ \right|$ désigne le cardinal, *L* désigne le nombre de locaux

Cette formule exprime le fait qu'un enseignant ou une classe ne doit pas apparaître plus d'une fois en la même période et que le nombre de cours planifiés à la même période ne doit pas excéder le nombre de locaux disponibles. L'évolution de la recherche se fait en changeant

chaque fois de période à un cours quelconque. Le critère d'arrêt est atteint quand l'emploi du temps idéal est trouvé($f(T)=0$) ou que le nombre d'itérations limite est atteint.

II.6 Hybridation des Méta-heuristiques

On assiste pendant ces dernières années à un intérêt grandissant vers l'hybridation des méta-heuristiques. Un intérêt justifié vu l'efficacité des méthodes hybrides comparées aux méthodes classiques. Dans cette section nous présentons un résumé des travaux opéré par *Talbi*[Tal 99b]. Cette étude classifie les méthodes d'hybridation de façon hiérarchique(figure II.35).

II.6.1 Classification Hiérarchique

Le premier niveau de classification fait intervenir le degré d'hybridation des différentes méthodes. Deux grandes classes sont identifiées: Hybridation haut niveau et hybridation bas niveau.

II.6.1.1 Hybridation bas niveau ou haut niveau

Les approches d'hybridation bas niveau touchent à la composition fonctionnelle de la méta-heuristique. Une fonction réalisée par la méta-heuristique est attribuée à une autre méthode. Dans les approches d'hybridation haut niveau, les traits de chaque méta-heuristique restent intacts et aucune interaction interne entre les méthodes n'est opérée.

II.6.1.2 Hybridation par relais ou par co-évolution

Dans l'hybridation par relais les différentes méta-heuristiques sont lancées l'une à la suite de l'autre. Chaque méta-heuristique à pour entrée la sortie de la méta-heuristique précédente. L'hybridation par co-évolution consiste en un modèle coopératif dont lequel les différentes méthodes sont lancées en parallèle.

Selon ces deux critères de classification, quatre classes peuvent être dégagées:

- *Hybridation bas niveau par relais (HBR):* Les différentes méta-heuristiques sont combinées en une même méta-heuristique travaillant sur une seule solution. *Martin* et *Otto* [Tal 99b]proposèrent l'hybridation des méthodes de recuit simulé et de recherche locale selon ce schéma. Lors de chaque itération, une forte perturbation est appliquée à la configuration courante (la solution début dans la figure II.36) générant une configuration intermédiaire. La recherche locale est lancée sur la configuration intermédiaire débouchant sur un optimum local. L'optimum local obtenu est alors accepté ou rejeté selon le procédé du recuit simulé (la différence d'énergie est calculée par rapport à la solution de début).

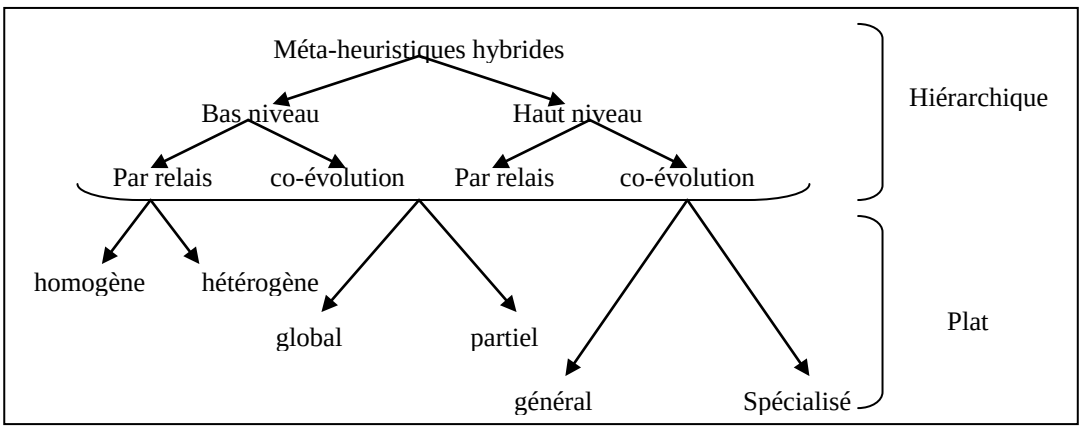


Figure II.35: Classification des méta-heuristiques hybrides.

- *Hybridation bas niveau par co-évolution (HBC):* Généralement l'approche consiste en une recombinaison d'une méta-heuristique à base de population (algorithmes génétiques, colonies de fourmis, …) avec une méthode de recherche locale tel que le recuit simulé, recherche tabou ou la recherche descendante(figure II.37).

  L'algorithme de recherche locale tente d'optimiser localement une seule solution tandis que l'algorithme à base de population tente d'améliorer les solutions globalement. La plupart des méthodes appartenant à cette classe d'hybridation se basent sur l'AG comme procédure d'optimisation globale. Dans ce cas, les opérateurs génétiques sont réadaptés afin d'incorporer une procédure de recherche locale utilisant l'individu comme solution initiale. Le résultat finale remplace l'individu original dans la population. Ce principe s'inspire de la nature, les individus suivent un cycle d'apprentissage avant de pouvoir se reproduire à nouveau.

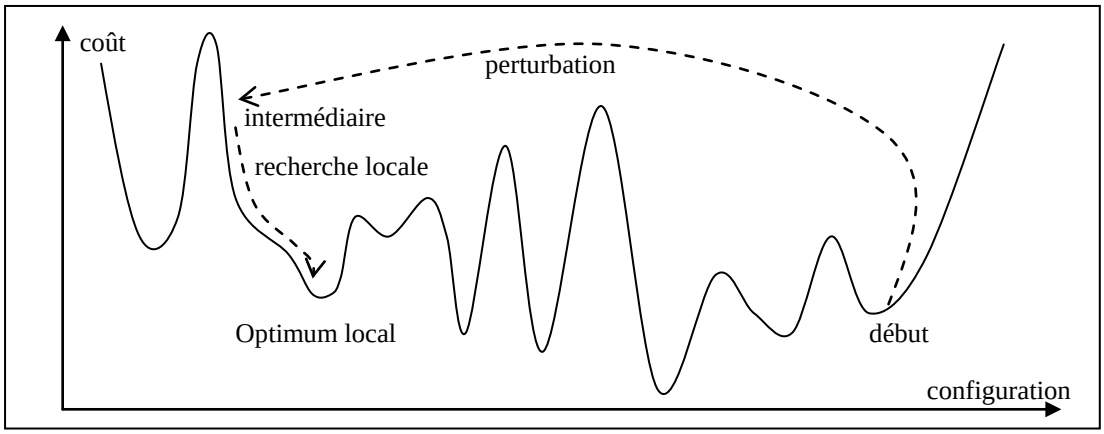


Figure II.36: Hybridation *HBR* du recuit simulé et recherche locale

- *Hybridation haut niveau par relais (HHR):* Les différentes méta-heuristiques sont lancées de façon séquentielle. l'objectif de telles approches est de bénéficier de la puissance de chacune des méta-heuristiques. Par exemple, les AGs sont réputés pour leur habilité à recenser les régions attractives de l'espace de recherche. Cependant, ils trouvent du mal à atteindre les optimums locaux dans ces régions. Il sera donc intéressant d'user de méthodes de recherche locale afin d'accélérer la convergence vers ces optimums(figure II.38).

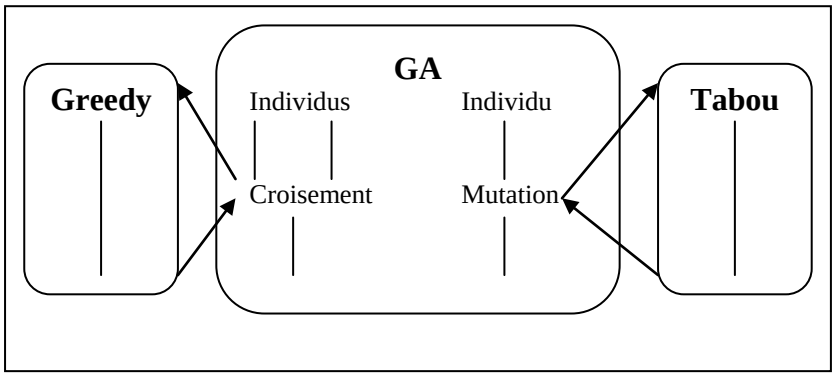


Figure II.37: Hybridation HBC

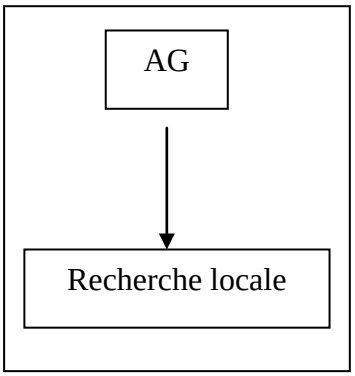


Figure II.38: Hybridation HHR

- *Hybridation haut niveau par co-évolution (HHC):* des méta-heuristiques distinctes effectuent la recherche en parallèle et coopèrent afin de trouver la solution optimale. La coopération revêt généralement l'aspect d'un échange des meilleures solutions trouvées par chaque processus. La topologie de communication entre les processus, la fréquence d'échange sont autant de paramètres influants de la méthode.

D'autre critères de classification des approches d'hybridation font intervenir:

- *Hybridation globale versus partielle:* dans une hybridation global , les différentes méta-heuristiques travaillent sur l'entièreté de l'espace de recherche. Dans l'approche d'hybridation partielle, le problème initiale est décomposé en plusieurs sous-problèmes. Chaque algorithme est dédié à la résolution d'un sous problème et opère dans une partie de l'espace de recherche.

- *Hybridation homogène versus hétérogène :* dans une hybridation homogène la totalité des processus en coopération implémente la même méta-heuristique.

- *Générale versus Spécialiste:* dans une hybridation spécialiste les algorithmes sont combinés afin de résoudre différents problèmes. Chaque algorithme est désigné pour traiter un problème particulier.

II.7 Conclusion

Vue l'impuissance des méthodes de résolution classiques face au problèmes d'optimisation combinatoire, les méta-heuristiques se sont illustrées comme une alternative intéressante. Les méta-heuristiques sont des techniques de recherche générale capables de résoudre une variété de problèmes d'optimisation. Il incombe au concepteur d'adapter les principes généraux qui régissent ses méta-heuristiques afin de répondre aux exigences et particularités du problème traité.

On retrouve deux classes de méta-heuristiques: méthodes de recherche globale ou locale. Les procédures de recherche globale ont la particularité de travailler sur une population de solutions, cas des algorithmes génétiques et colonies de fourmis. Les procédures de recherche locale telles que le recuit simulé et la recherche tabou travaillent quant à elles sur une seule solution à un moment donné.

L'efficacité d'une méta-heuristique est mesurée en regard à la manière dont est géré le compromis entre l'intensification et la diversification, l'aptitude de la technique à échapper aux optimum locaux et la faculté de la méthode à offrir plusieurs solutions en cas de paysage multi-modales.

La puissance des méta-heuristiques est souvent améliorée par l'exploitation des possibilités de parallélisation et d'hybridation. La parallélisation d'une méthode permet de réduire le temps d'exécution de la recherche et améliore son caractère coopératif. L'hybridation offre quant à elle la possibilité de bénéficier de l'efficacité des différentes méta-heuristiques à des stades différents de la recherche, pour traiter des types différents de sous-problèmes.

Les approches citées lors de cette partie sont dédiées à la résolution des problème d'optimisation monocritère. Nous présentons dans la partie suivante une étude des problèmes d'optimisation multicritère, leurs particularités et les différentes approches de résolutions.

Nous nous attarderons surtout sur la réadaptation des techniques méta-heuristiques pour la résolution de ce genre de problèmes.

Partie III: Problèmes d'optimisation multicritère

III.1 Introduction

Pendant de nombreuses années, l'intérêt des chercheurs s'était porté sur les problèmes d'optimisation moncritère. Un constat qui ne concorde pas avec la réalité industrielle, où la majorité des problèmes notamment d'ordonnancement[Bil 99][T'ki 99][Nag 95] présentent plusieurs (souvent contradictoires) objectifs à optimiser. Dans un problème d'emploi du temps par exemple(Paragraphe I.11.3), des critères tels que la procuration de journées libres et l'équilibrage de charge d'étude sur la semaine peuvent figurer comme objectifs de l'optimisation[Dow 96]. Cependant, ces deux critères sont relativement contradictoires, du fait que la procuration de journées libres implique inévitablement la surcharge de certaines journées par rapport à d'autres.

Le rôle de l'optimisation dans ce cas consiste à générer un ensemble de solutions optimales plutôt qu'une seule. Il appartiendrait donc au décideur de choisir une solution parmi celles proposées. La façon dont s'effectue la coopération décideur/solveur s'inscrit parmi les critères d'estimation de la qualité de l'approche proposée.

Cette partie est consacrée à la présentation des concepts de base de l'optimisation multicritère. Nous dresserons un constat des différentes techniques utilisées dans ce cadre, telles que la programmation mathématiques, les techniques de recherche arborescente et les méta-heuristiques.

Dû à leur maniement d'une population de solutions, les algorithmes évolutionnistes ont été les plus utilisées dans le cadre de l'optimisation multicritère. Pour cela, nous nous étalerons sur l'étude des différents mécanismes qui ont servi à l'implémentation des AGs pour ce type d'optimisation.

III.2 Concepts de base

Mathématiquement, un problème d'optimisation multi-objectifs (POMO)[Ste 85] est définit comme suit:

$$\begin{cases} \text{optimiser } F(x) = [f_1(x), f_2(x), ..., f_n(x)]^T \\ \text{sous la condition } x \in C \end{cases} \quad \text{(III.1)}$$

$$x = [x_1, x_2, ..., x_k]^T \quad \text{(III.2)}$$

f_1, f_2,…,f_n désignent les différents critères à optimiser ($n \geq 2$). Les variables x_1, x_2,…,x_k représentent les variables de décision. L'évaluation d'une solution $\bar{x}$ est décrite par un vecteur objectif $F(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \ldots, f_n(\bar{x})]^T$. Nous serons donc en face de deux types d'espaces euclidiens:

- *L'espace de décision X de dimension k* qui désigne l'ensemble des solutions $x = [x_1, x_2, ..., x_k]^T$.

- *L'espace objectif de dimension n* qui désigne l'ensemble des évaluations possibles correspondant aux vecteurs objectifs $F(x)$.

A chaque point de l'espace de décision est associé un point de l'espace objectif désignant l'adéquation de la solution associée. Le POMO consiste à trouver une solution x^* qui satisfasse l'ensemble de toutes les contraintes C et qui optimise le vecteur objectif $F(x^*)$. L'optimisation du vecteur objectif F revient soit:

- A minimiser l'ensemble de toutes les fonctions objectifs.
- A maximiser l'ensemble de toutes les fonctions objectifs.
- Minimiser certaines fonctions objectifs tout en maximisant d'autres.

Pour des raisons de simplification nous convertissons toutes les fonctions à la forme de minimisation.

$$\min f_i(\bar{x}) = -\max(-f_i(\bar{x})) \tag{III.3}$$

Le fait qu'il soit rare de trouver une solution x^* qui représente le minimum de chaque composant f_i du vecteur objectif rend la notion d'optimalité vague dans ce cas. L'établissement de nouveaux critères d'optimalité est donc nécessaire.

III.2.1 Notion d'optimalité Pareto

Définition. 1: La notion d'optimalité Pareto a été introduite en économie pour la première fois par *Vilfredo Pareto* en 1896. Une solution x^* est dite Pareto optimale si pour toute autre solution x on a:

$$(\forall i \in [1..n] \;\; f_i(x^*) = f_i(x)) \text{ ou } (\exists i \in [1..n] \text{ tel que } f_i(x^*) < f_i(x)) \tag{III.4}$$

De façon plus claire, une solution x^* est dite Pareto optimale quand il n'est plus possible d'améliorer la qualité d'un objectif sans dégrader un autre. La notion d'optimalité Pareto engendre généralement un ensemble de solutions dites solutions non inférieures ou solutions non dominées.

Définition. 2: *La dominance* décrit une relation d'ordre partiel entre les points de l'espace de décision. Une solution x_1 domine la solution x_2 si la condition suivante est satisfaite:

$$(\forall i \in [1..n] \;\; f_i(x_1) \le f_i(x_2)) \text{ et } (\exists i \in [1..n] \text{ tel que } f_i(x_1) < f_i(x_2)) \tag{III.5}$$

Autrement dit, la solution x_1 est au moins identique à x_2 en regard à tous les objectifs et elle est meilleure que x_2 au moins pour un objectif donné(figure III.1).

Définition. 3: Une solution x^* est dite *faiblement non dominée* s'il n'existe aucune autre solution x tel que :

$$f_i(x) < f_i(x^*) \;\; \forall i \in [1..n] \tag{III.6}$$

Il est clair donc qu'une solution Pareto (non dominée) est forcement faiblement non dominée.

Définition. 4: Le front Pareto désigne la courbe décrite par les solutions non dominées du problème dans l'espace objectif (voir la figure III.2).

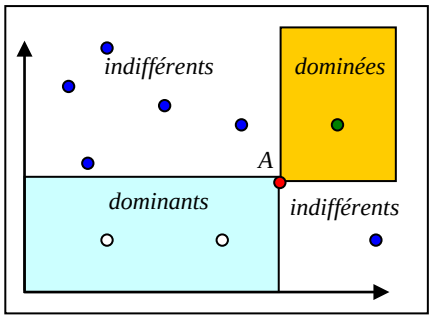


Figure II.1: relation de dominance entre le point *A* et les autres solutions.

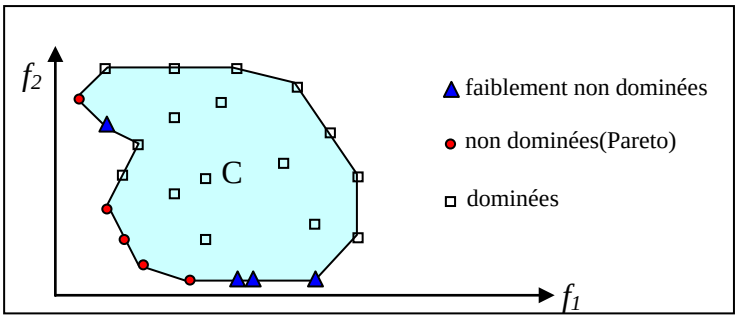


Figure II.2: solutions fortement et faiblement non dominées dans l'espace objectif

III.2.2 Notion d'optimalité Min-Max

Dans certains problèmes d'optimisation multicritère, le décideur n'est intéressé que par l'obtention d'une seule solution et le fait que le nombre de solutions Pareto soit grand, rend le processus de génération et de manipulation de cet ensemble très difficile[Coe 99].

La notion d'optimalité Min-Max[Coe 98] a été tirée de la théorie des jeux qui se rapporte à la résolution des cas de conflit. L'idée est de comparer les déviations relatives par rapport aux valeurs minimales atteintes. Considérant l'objectif f_i, la déviation de la solution *x* par rapport à cet objectif est donnée par la formule suivante:

$$z_i(x) = \frac{\left| f_i(x) - f_i^0 \right|}{f_i^0} \text{ ou } z_i(x) = \frac{\left| f_i(x) - f_i^0 \right|}{f_i(x)} \text{ où } f_i^0 \text{ désigne la plus petite valeur de } f_i \text{ atteinte} \qquad \text{(III.7)}$$

Les fonctions $z_i(x)$ définissent les augmentations relatives par rapport aux objectifs. Soit $z_i(x) = [z_1(x), z_2(x), ..., z_n(x)]^T$ le vecteur d'augmentation relative, un point *x** est dit Min-Max optimal si la condition récursive suivante est vérifiée :

Etape 1:

$$v_1(x^*) = \min_{x \in C} \left(\max_{i \in [1..n]} \{ z_i(x) \} \right) \qquad \text{(III.8)}$$

Soit i_1 l'objectif pour lequel $z_i(x)$ est maximal, $I_1 = \{i_1\}$ et C_1 l'ensemble des solutions *x** satisfaisant l'*étape 1*.

Etape 2:

$$v_2(x^*) = \min_{x \in C_1} \left(\max_{i \in [1..n], i \notin I_1} \{ z_i(x) \} \right) \qquad \text{(III.9)}$$

Soit i_2 l'objectif pour lequel $z_i(x)$ est maximal dans cette étape., $I_2 = \{i_1, i_2\}$ et C_2 l'ensemble des solutions *x** satisfaisant l'*étape 2*.

Etape k:

$$v_k(x^*) = \min_{x \in C_{k-1}} \left(\max_{i \in [1..n], i \notin I_{k-1}} \{ z_i(x) \} \right) \qquad \text{(III.10)}$$

Soit C_k l'ensemble des solutions *x** satisfaisant l'*étape k*.

Cette formule exprime le fait qu'ayant les extrêmes des fonctions objectifs obtenus par la résolution du problème réduit à chaque critère séparément, une solution est considérée désirable si elle donne la plus petite valeur des augmentations relatives de toutes les fonctions objectifs.

Une solution x^* qui satisfasse les deux étapes 1 et 2 peut être considérée comme présentant le meilleur compromis entre les objectifs. Vu la complexité de calcul engendrée par ces différentes étapes, souvent seule la première étape est prise en considération.

III.3 Optimisation et aide à la décision

La résolution d'un POMO revêt deux aspects de difficultés distinctes: l'optimisation et l'aide à la décision. L'aspect optimisation, concerne le processus de recherche ou d'approximation de l'ensemble des solutions optimales (Pareto ou Min-Max). L'aspect aide à la décision s'adresse au problème de sélection d'une solution représentant un bon compromis entre les différents critères. Une prise de décision humaine est donc nécessaire pour gérer les interdépendances entre les objectifs. Selon la façon suivant laquelle, les deux aspects sont combinés, trois approches principales sont possibles[Tal 99][Zit 99][Vel 98].

1. *Aide à la décision à priori:* Cette approche consiste à combiner les différentes fonctions objectifs en une seule fonction dite *fonction d'utilité*. Ce procédé suggère que le décideur a une connaissance des différents poids des objectifs ainsi que de la fonction d'utilité. Le problème est alors traité par un algorithme d'optimisation monocritère classique. Malheureusement, la fonction d'utilité n'est généralement pas connue par le décideur à priori, comme il lui est difficile d'associer des poids à des critères qui sont souvent non commensurables.

2. *Aide à la décision à posteriori:* Dans ce cas l'algorithme d'optimisation est lancé en premier générant plusieurs solutions. Le décideur choisit alors parmi celles-ci la solution qui lui convient le plus.

3. *Aide à la décision interactive:* Une coopération entre le solveur et le décideur est installée. A partir des connaissances acquises lors de la recherche, le décideur formule ces préférences. Ces dernières sont introduites par le solveur ultérieurement. Ce processus est réitéré plusieurs fois.

III.4 Problème d'optimisation multicritère

Les travaux de ces 30 dernières années ont surtout porté sur la programmation multicritère linéaire à variables réelles[Vin 76][Sak 90][Kuf 97][Ste 85][Wei 93]. Ceci est dû au développement de la méthode dans le cas monocritère, à la relative facilité de traitement de tels problèmes et à l'abondance de cas pratiques qui peuvent être formuler sous forme linéaire.

Les problèmes d'optimisation multicritère étudiés dans la littérature peuvent être organisés en deux grandes classes: d'une part les problèmes d'optimisation académiques et d'autre part les problèmes industriels. Les problèmes d'optimisation académiques multicritère se présentent comme des extensions de problèmes monocritère alors que ceux liés à l'industrie sont par nature multicritère.

III.4.1 Applications académiques

Le principal intérêt de tels problèmes est leur utilisation en tant que moyen de comparaison entre algorithmes d'optimisation multicritère. La plupart [Ehr 00] sont une simple formulation du problème original sous une version multicritère: sac à dos[Ben 99][Meu 99],voyageur de commerce, Flow Shop[Mur 98], routage de véhicules[Chr 75][Ben 00], etc...

- *Optimisation de fonction continues:* Les fonctions dues à *Scheffer* sont souvent utilisées par la communauté génétique pour tester et comparer l'efficacité des méthodes. La fonction bi-critères f_2 est la plus utilisée:

$$\begin{cases} f_{21}(x) = x^2 \\ f_{22}(x) = (x-1)^2 \\ \text{sous la contrainte } x \in [-10,10] \end{cases} \quad \text{(III.11)}$$

- *Voyageur de commerce:* Le PVC a été largement étudié dans la littérature[Zit 99]. Le but étant la recherche du plus petit chemin parcourant *nv* villes. L'extension du problème au cas multicritère consiste généralement à l'association de délai limite d'asservissement à chaque ville. Il s'agit alors de minimiser le nombre ou les délais de retards enregistrés. L'extension peut aussi se faire en introduisant plusieurs matrices de distance D_i de dimension $nv \times nv$. $D_i[v_1,v_2]$ représente la distance entre les villes v_1 et v_2 suivant l'objectif f_i.

- *Sac à dos:* La forme multicritère du problème de sac à dos a été étudiée par plusieurs auteurs [Ben 99][Zit 99][Sal 99]. Le problème consiste à determiner parmi un ensemble d'items de poids et de profits différents, ceux à introduire dans le sac afin de maximiser le profit sans dépasser la capacité du sac (la somme des poids ne doit pas excéder la capacité du sac). L'extension du problème au cas multicritère peut être réaliser en considérant plusieurs sacs à dos. A chaque item est associé un profit dépendant du sac lui-même.

- *Problème de couverture maximale*: l'application pratique d'un tel problème apparaît dans le domaine de positionnement des poste de secoure[Ehr 00]. Le problème est défini par *M* centres hurbains et *N* emplacements possibles pour les centres de soins. La construction d'un centre de soin à l'emplacement E_i induit un coût de construction c_i. Il s'agit donc de déterminer les emplacements où s'effectuerons les constructions afin de couvrir l'ensemble de tous les sites, en minimisant le coût de construction total. Les relations de couvertures des sites par les emplacements sont décrites par une matrice *A* de dimension $M \times N$. $A[i, j]=1$ si le site *i* est couvert par l'emplacement *j*. L'extension du problème au cas multicritère correspond à attribuer différents coûts de construction sur chaque emplacement en fonction de l'objectif pris en compte.

III.4.2 Problèmes d'ordonnancement multicritère

Il a été prouvé que dans un problème d'optimisation multicritère, il suffit que l'optimisation d'un seul critère soit Np-complet pour que le problème multicritère le soit aussi. *Chen* et *Bulfin*(1993)[T'ki 99] établirent des règles permettant le calcul de la complexité d'un problème bi-critères à partir de la complexité des problèmes monocritère correspondants.

Les problèmes d'ordonnancement d'ateliers multicritère à une machine ont fait l'objet de nombreuses tentatives de classification. Dans [T'ki 99], on trouve une classification de ces problèmes selon le degré de complexité qui leurs sont associés(problème de classe *P*, problème *Np-complet*, et problème de complexité inconnu). Le problème d'emploi du temps a été traité dans sa forme multicritère par *Dowsland* [Dow 96]. Dans [Ben 00] nous proposons une approche de résolution multicritère pour le problème de routage de véhicule.

Dans le paragraphe I.11.3 vous trouverez une description des différents critères d'optimisation pouvant être retenus pour les problèmes de Flow Shop, de Job Shop, d'emploi du temps et de routage de véhicules.

III.5 Approches de résolution

Les problèmes bi-critères en reçu une attention particulière. Des méthodes exactes telles que le *Branch & Bound*, l'algorithme *A** et la programmation dynamique ont subit des révisions afin de les adapter au cas multicritère. Cependant ces approches sont destinées à des problèmes de petites tailles, et leur efficacité est mise en question dès que la taille du problème augmente et que le nombre de critères retenus croît. La procédure du Simplex a aussi reçu des modifications afin de répondre aux exigences de l'optimisation à plusieurs objectifs, on trouve dans [Ste 85] une étude assez détaillée des travaux effectués dans ce sens.

Les approches de recherche approximative, telles que les méta-heuristiques, tentent d'approcher l'ensemble des solutions optimales. Ces méthodes sont applicables à une large gamme de problèmes et leur efficacité demeure relativement bonne quand la taille et le nombre d'objectifs augmentent.

III.5.1 Heuristiques de recherche exactes révisées

III.5.1.1 Algorithme A*

Stewart et *White* proposèrent la généralisation multicritère de l'algorithme *A** (*A*MO*) en 1991[Ste 91]. L'algorithme *A** est maintenu dans sa forme originale, les fonctions *k**, *g** et *h** prennent une forme non scalaire. Le poids d'un arc (u, v) dans le graphe de recherche, correspond à un tuple (c_1, c_2, …, c_n) où c_i désigne le coût relatif à l'objectif f_i induit par le passage de u à v(figure III.3). Le coût d'un chemin, correspond à la somme vectorielle des poids des arcs qui le composent.

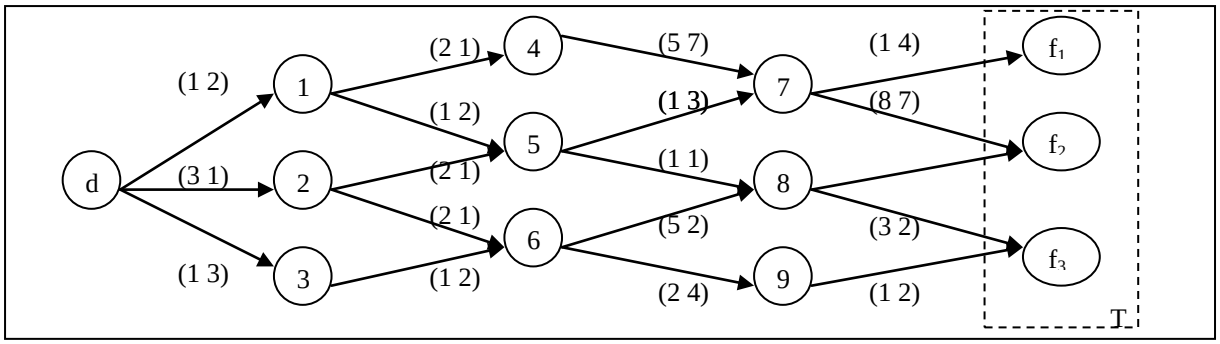


Figure III.3: Exemple d'arbre de recherche pour l'algorithme *A*MO* bi-critères

- *k**(u, v) désigne l'ensemble des coûts non dominés des chemins reliant u à v. *K**(u, v) = (+∞, +∞, …+∞) s'il n'existe aucun chemin reliant u à v.

- $g^*(u) = k^*(d, u)$ représentant le coût de la recherche ayant aboutit à la solution intermédiaire u.

- $h^*(u)$ l'ensemble des vecteurs coût non dominés parmi l'ensemble$\{ k^*(u, v)/ v \in T\}$ où T représente l'ensemble des états terminaux.

Les nœuds du graphe sont triés et parcourus comme pour l'algorithme *A** classique, l'ordre de trie est décrit par les relations de dominance entre les vecteurs coût *f*.

III.5.1.2 Programmation dynamique

L'application de la programmation dynamique dans le cadre de l'optimisation multicritère est rare. La difficulté est liée au principe de monotoniété exigée par la méthode. Ce principe réclame que la préférabilité d'une solution partielle reste préservée par récursion(voir le paragraphe I.8.3). Ce principe est difficilement vérifiable quand le nombre d'objectifs à optimiser est élevé(>2).

La technique de programmation dynamique multicritère a été implémentée par *Carraway*, *Morin* et *Moskowitz*[Car 90] pour la résolution du problème de routage dans les réseaux. Les critères à optimiser sont la minimisation des temps de communication et la maximisation de la fiabilité des transmissions. Dans cette approche les auteurs combinent les deux critères en une fonction d'utilité *U*.

III.5.1.3 Branch & Bound

Ramesh et *Zionts*[Ram 86] proposèrent une variante multicritère de l'algorithme de Branch & Bound appliqué aux problèmes d'optimisation linéaire à variables entières. Le problème est résolu en combinant la procédure de Branch & Bound avec une spécification interactive des préférences du décideur.

III.5.2 Approches de résolution approchées

Les années soixante ont vu se multipliés les situations de confit entre critères, particulièrement dans le domaine des finances. Des situations où les mots compromis et équilibre représentent la clé de la réussite. Les techniques alors élaborées procèdent soit par transformation du problème en un problème monocritère, soit en se basant sur la notion d'optimalité Pareto ou finalement par traitement séparé des différents critères.

III.5.2.1 Transformation du problème vers le monocritère

Cette approche[Tal 99] procède en transformant le problème du cas multicritère vers le monocritère. Parmi ces méthodes en trouvent les techniques d'agrégation, les méthodes ε-contrainte et la programmation par but.

III.5.2.1.1 Méthode d'agrégation

C'est certainement la technique la plus utilisée et la plus facile à implémenter. Les différents objectifs sont combinés au sein d'une même fonction d'utilité, généralement de façon linéaire.

$$f(x) = \sum_{i=1}^{n} \lambda_i f_i(x) \qquad \text{(III.12)}$$

Les poids λ_i déterminent le degré de priorité d'un objectif donné. L'aspect non-commensurable des différents critères nécessite l'introduction de poids c_i de changement d'échelle. La formule précédente prend alors la forme suivante:

$$f(x) = \sum_{i=1}^{n} c_i \lambda_i f_i(x) \qquad \text{(III.13)}$$

Les poids de changement d'échelle c_i sont généralement calculés suivant la formule:

$$c_i = \frac{1}{f_i^0} \qquad \text{(III.14)}$$

Il est simple de prouver que l'optimisation de la somme pondérée des fonctions objectifs revient à trouver une solution Pareto optimale. La nouveauté dans cette approche consiste à générer plusieurs solutions Pareto optimale en modifiant les valeurs des poids. Un tel procédé nous offre une idée sur la forme de la surface Pareto(région de l'espace de recherche contenant les solutions Pareto optimales). Ainsi que des informations sur l'interaction entre les différents objectifs.

Cette approche souffre de deux inconvénients :

- Une répartition uniforme des valeurs des poids produit rarement un ensemble de solutions uniformément réparties de l'espace Pareto. Souvent, ces points sont tellement rapprochés que la forme de l'espace ne peut être déduite.

- La partie non convexe de l'espace Pareto ne peut être obtenue à partir de l'optimisation de somme convexe des objectifs.

III.5.2.1.2 Méthode ε-contrainte

Dans cette technique[Zit 99] en s'efforce d'optimiser un but précis de l'ensemble des objectifs que compte le problème en exigeant que le reste des objectifs soient d'une adéquation assez bonne par rapports à des seuils fixés. Une telle approche est autant plus intéressante quand l'utilisateur désir privilégier un objectif donné. Par contre, la méthode perd de son intérêt quand le nombre d'objectifs est élevé.

Le critère le plus prioritaire est pris comme unique objectif de l'optimisation. Les autres critères sont alors transformés en des contraintes de restriction d'intervalle. Le problème est converti en un problème monocritère dont l'objectif est d'optimiser la fonction objectif f_p la plus prioritaire en contraignant les autres fonctions. A chaque objectif $i \neq p$ est associé une borne ε_i indiquant la plus mauvaise valeur admise pour la fonction f_i. Le problème est alors formulé comme suit:

$$\begin{cases} \min\ f_p(x) \\ x \in C \\ \text{sous la contrainte } f_i(x) \le \varepsilon_i, \forall i \in [1..n], i \ne p \end{cases}$$

L'inconvénient de cette méthode réside dans le fait que la connaissance à priori des intervalles appropriés pour les valeurs ε_i est exigée. Ceci induit le calcul des bornes inférieures des différents objectifs. La figure III.4 décrit le fonctionnement de la méthode.

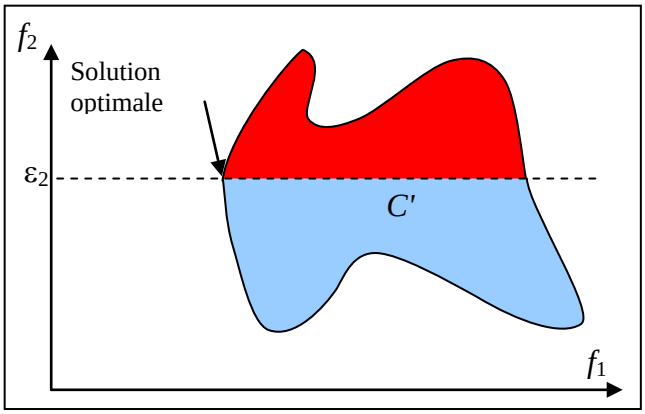


Figure III.4: Méthode ε-contrainte

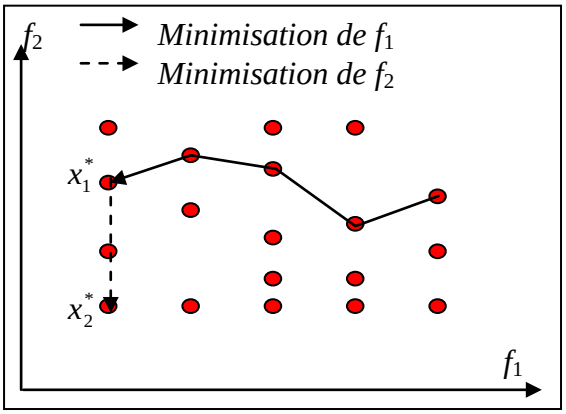


Figure III.5: Méthode d'optimisation séquentielle

A fin d'obtenir différentes solutions Pareto l'algorithme d'optimisation peut être lancé plusieurs fois avec des valeurs de ε_i distinctes. Contrairement à la méthode de transformation par somme pondérée des objectifs, cette technique est capable de générer des solutions non supportées.

III.5.2.1.3 Méthode d'optimisation séquentielle(Multi-niveaux)

Dans cette approche on opère en se concentrant chaque fois sur un seul objectif. Les objectifs sont alors triés par ordre décroissant de leur importance. La première étape consiste à trouver l'ensemble des points pour lequel l'optimum du premier objectif est atteint. L'étape suivante consiste à recenser parmi cet ensemble le sous-ensemble de points qui optimise le deuxième objectif ainsi de suite.

La méthode de programmation Multi-niveaux est intéressante quand l'ordre hiérarchique entre les objectifs est d'une importance primaire ce qui rend l'utilisateur non concerner par l'interaction entre les différents objectifs. Cependant les performances d'une telle approche se détériorent quand cet ordre devient très contraignant au point que les objectifs les moins importants deviennent d'une influence nulle sur le résultat final. Une telle approche est donc à éviter quand l'utilisateur souhaite un certain compromis et une certaine souplesse dans l'optimisation des différents objectifs.

Un ordre lexicographique est établi entre les objectifs, l'ensemble des solutions Pareto est obtenu en minimisant les objectifs l'un après l'autre dans l'ordre décroissant des priorités($f_{i_1}, f_{i_2}, \cdots, f_{i_n}$). Le premier problème résolu sera formulé de la manière suivante:

$$\min\ f_{i_1}(x), \text{sous la contrainte } x \in C \qquad \text{(III.15)}$$

Le résultat de cette première résolution est une solution x_1^*. La résolution du deuxième problème est alors lancée, formulée sous la forme suivante:

$$\min f_{i_2}(x), \text{ sous la contrainte } x \in C \text{ et } f_{i_1}(x) = f_{i_1}(x_1^*) \qquad \text{(III.16)}$$

A chaque itération, des contraintes d'égalité sont associées aux fonctions déjà optimisées. Le j^{ème} problème prend la forme suivante:

$$\min f_{i_j}(x), \text{ sous la contrainte } x \in C \text{ et } f_{i_1}(x) = f_{i_1}(x_1^*),\ f_{i_2}(x) = f_{i_2}(x_2^*), \cdots,\ f_{i_{j-1}}(x) = f_{i_j}(x_{j-1}^*) \qquad \text{(III.17)}$$

III.5.2.1.4 Programmation par but

Cette technique[Deb 99] se base sur une définition a priori d'un vecteur but. Le vecteur but $B = [b_1, b_2, \ldots, b_n]^T$ désigne un vecteur référence des valeurs que désir atteindre le décideur pour chaque objectif. Par exemple, la valeur b_i représente la valeur à ne pas dépasser pour l'objectif f_i. le problème est alors formulé comme suit:

$$\begin{cases} \min f(x) = \left(\sum_{i=1}^{n} \lambda_i \left| f_i(x) - b_i \right|^p \right)^{\frac{1}{p}} \\ \text{sous la contrainte } x \in C \end{cases} \qquad \text{(III.18)}$$

Les poids λ_i décrivent toujours la priorité entre les différents objectifs. Le but est donc de minimiser la distance séparant la solution trouvée de la solution idéale *B*. La métrique de *Tchebycheff* est utilisée pour calculer cette distance avec généralement 2 comme valeur de *p*. Quand *p* tend vers l'infini l'équation revient à la fonction Min-Max. de l'étape 1.

La recherche ainsi décrite est sensible au choix du vecteur but, un choix arbitraire de celui-ci peut conduire à des solutions non Pareto.

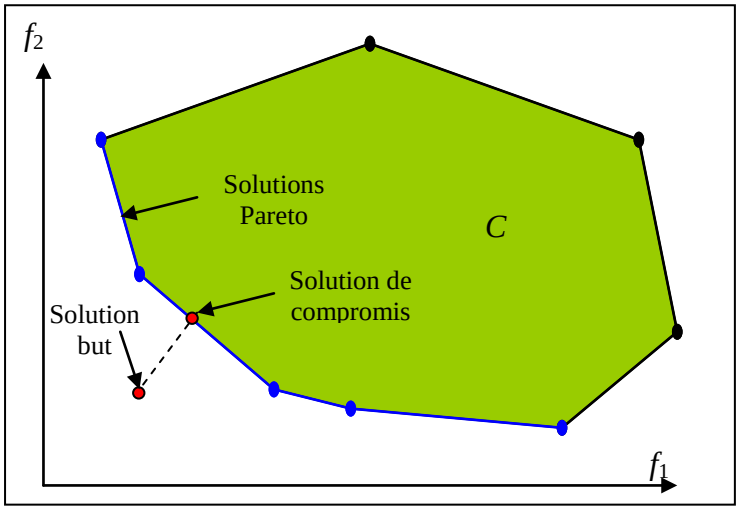


Figure III.6: Technique de programmation par but

III.5.2.1.5 Approche de théorie de jeux

Pour des raisons d'explication nous nous restreindrons à l'étude d'un exemple bi-critères à deux variables de décision. Soient $f_1(x_1, x_2)$ et $f_2(x_1, x_2)$ les deux fonctions objectifs à

optimiser, deux joueurs sont alors associés à chacun des deux objectifs. Le premier joueur tente d'optimiser la fonction f_1 en opérant sur la variable x_1, le deuxième joueur quant à lui tente d'optimiser la fonction f_2 en cherchant la valeur adéquate de la variable x_2. Dans la figure III.7 nous pouvons voir les lignes à valeurs égales de f_1 et f_2. Les lignes discontinues traversant O_1 et O_2 représentent les chemins des choix rationnels(minimisant) du premier et second joueur pour des valeurs de x_2 et x_1 respectivement. L'intersection des ces deux lignes, si elle existe, est candidate aux deux problèmes de minimisation. Dans la figure III.7 le point $N(x_1^*, x_2^*)$ représente un tel point. Ce point dit point d'équilibre représente la condition d'équilibre stable, c'est à dire qu'aucun mouvement unilatéral d'un joueur depuis ce point ne peut améliorer le critère correspondant. Ce point est caractérisé par:

$$f_i(x_1^*, x_2^*) = f_i(x_1, x_2^*) \text{ et } f_2(x_1^*, x_2^*) = f_2(x_1, x_2) \tag{III.19}$$

x_1 désigne un point à gauche ou à droite du point x_1^* et x_2 est un point au-dessous ou au-dessus du point x_2^*. L'extension de cette idée à n joueurs produit la définition suivante de la solution d'équilibre.

$$\begin{cases} f_1(x_1^*, x_2^*, ..., x_n^*) \le f_1(x_1, x_2^*, ..., x_n^*) \\ f_2(x_1^*, x_2^*, ..., x_n^*) \le f_1(x_1^*, x_2, ..., x_n^*) \\ ... \\ f_n(x_1^*, x_2^*, ..., x_n^*) \le f_n(x_1^*, x_2^*, ..., x_n) \end{cases} \tag{III.20}$$

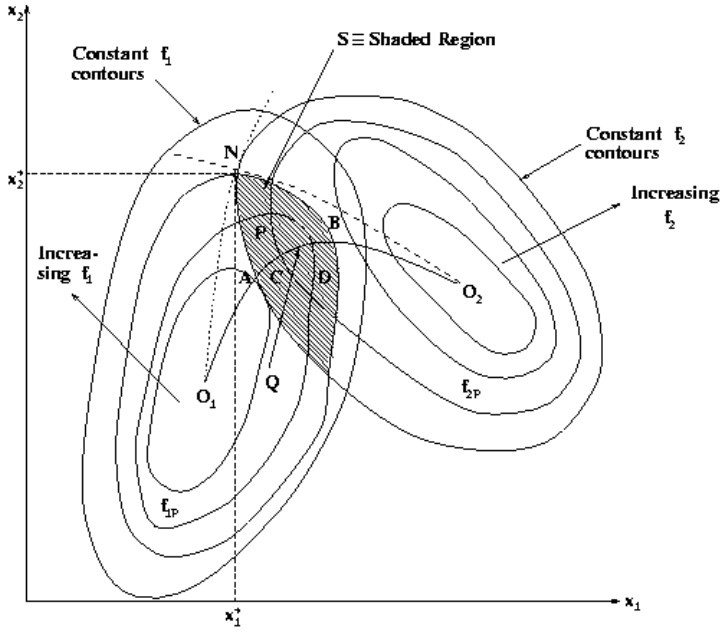


Figure III.7: Technique de programmation par but

III.5.2.2 Approches par traitement séparé des objectifs

L'optimisation des différents critères est réalisée en traitant les différents objectifs séparément. Dans les AGs la prise en compte des différents critères apparaît au niveau de la phase de sélection (sélection parallèle, sélection lexicographique) ou de la phase de reproduction

(reproduction multi-sexuelle). Nous reviendrons sur ces techniques lors de l'étude des algorithmes évolutionnistes multicritère.

III.5.2.3 Approches Pareto

Ces techniques partagent le fait de s'appuyer sur la notion d'optimalité Pareto. La notion de dominance est utilisée comme moyen de comparaison et d'ordonnancement des solutions. Le principal avantage de telles approches est qu'elles sont capables de générer un ensemble de solutions Pareto appartenant aux parties concaves de la frontière Pareto. Dans les AGs la prise en considération du caractère dominant ou dominé des solutions apparaît dans la phase de sélection. L'adéquation des solutions est estimée sur la base d'un ordre établi entre les individus de la population. Ce procédé qui consiste à ordonner les individus est nommé *ranking*.

III.6 Algorithmes évolutionnistes multicritère

Goldberg [Gol 89] indiquait que la notion de recherche génétique est multicritère dans sa nature. Cependant, les AGs requièrent une connaissance scalaire sur l'adéquation des solutions. La formulation du problème comme optimisation d'un critère global n'est souvent pas évidente. La conversion du problème par utilisation d'une somme pondérée des objectifs offre l'avantage de la simplicité d'implémentation. En plus, une seule solution est fournie, éliminant le besoin du décideur à intervenir. Cette technique reste inefficace puisqu'un choix inapproprié des poids induira un temps d'exécution additionnel jusqu'à ce qu'une solution convenable soit trouvée. L'étude des méthodes travaillant par transformation du problème vers le cas monocritère ne nous intéresse pas dans cette section.

III.6.1 L'élitisme et les AGs Multicritère

Dans les algorithmes évolutionnistes multicritère, l'élitisme joue un rôle encore plus important. A l'opposé de l'optimisation monocritère l'introduction de l'élitisme est plus complexe[Zit 99]. En effet au lieu d'avoir une seule meilleure solution, un ensemble de solutions non dominées sont à manier, dont la taille est comparable à une population. Cela nous conduit aux deux questions suivantes:

- *Population→Ensemble élite*: Quels sont les individus à maintenir dans l'ensemble élite et pour combien de temps?

- *Ensemble élite→Population:* Quand et de quelle façon sont réintroduites les solutions élites dans la population?

Deux approches peuvent se présenter. La première, copie systématiquement dans l'ensemble élite les individus de la population non dominés. Une variante plus restrictive consiste à ne tenir compte que des *n* individus ayant un vecteur objectif qui maximise l'un des *n* critères.

Généralement, les individus introduits dans la population élite ne sont retirés que quand une solution meilleure est insérée. De ce fait la population élite contient les meilleures solutions trouvées tout au long de la recherche. A chaque génération, un certain pourcentage des individus de la population sont remplacés par des membres de la population élite. Ces membres sont soit choisis de façon aléatoire ou suivant un autre critère comme la période pendant laquelle la solution est restée élite. Pour limiter le nombre de solutions élites d'autres

mécanismes doivent être implémentés tel que le maintient d'une certaine dissimilarité entre les individus de l'ensemble élite.

III.6.2 Niches écologiques et AGs Multicritère

Les travaux de *Obayashi, Takahashi, Takeguchi*[Oba 98], *Hiroyasu, Miki* et *Watanabe*[Hir 99] montrèrent l'intérêt qu'a le *Sharing* sur les performances des AGs multicritère. De point de vu utilisateur, il n'est pas du tout intéressant d'avoir un ensemble de solutions Pareto concentrées dans la même région. Le *sharing* permet dans ce cas de répartir les solutions trouvées à travers la frontière Pareto.

La fonction *sharing* prend souvent la forme suivante:

$$sh(dist(x,y)) = \begin{cases} 1 - \left(\frac{dist(x,y)}{\gamma} \right)^{\alpha} & Si \quad dist(x,y) < \gamma \\ 0 \text{ sinon} \end{cases} \tag{III.21}$$

Le paramètre γ désigne la taille des niches. Le vecteur objectif est révisé pour introduire la notion de *sharing* de la façon suivante:

$$m(x) = \sum_{y \in pop} sh(dist(x,y)) \tag{III.22}$$

$F'(x) = F(x)*m(x)$. (F est le vecteur objectif) (III.23)

Hiroyasu, *Miki* et *Watanabe* [Hor 99] proposèrent une nouvelle approche de sharing, n'opérant aucune modification sur la fonction objectif. La taille de la population est réduite, comme c'est illustré dans la figure III.8. La population est subdivisé en plusieurs sous ensemble représentant des niches de diamétre égal. Seules les solutions centrales des niches restent alors dans la population.

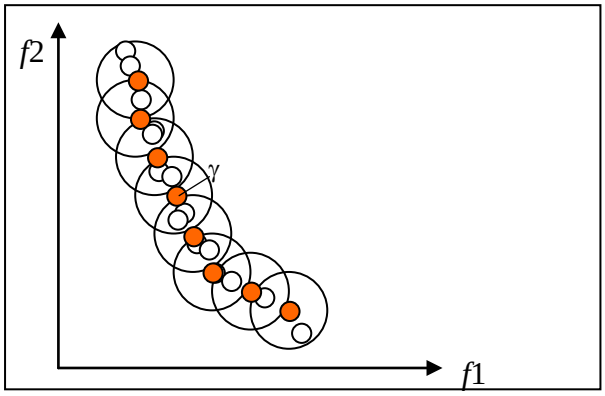


Figure III.8 :*Sharing* chez *Hiroyasu*(2 objectifs)

Récemment, Beaucoup d'études ont été faites afin de dresser un constat des différentes techniques employées et leurs performances respectives(*Talbi* [Tal 99], *Fonseca*[Fon 95], *Coello* [Coe 99], *Zitzler* [Zit 99], *Veldhuizeh*[Vel 99][Vel 98]).

III.6.3 Sélection parallèle(VEGA)

Schaffer [Zit 99] proposa en 1985 un algorithme évolutionniste multicritère appelé *VEGA*(vector evaluated genetic algorithm). La technique aussi connu sous l'appellation de sélection parallèle, opère par traitement séparé des différents critères. A chaque génération, la population est subdivisée en *n* sous-populations de tailles égales. A chaque sous-population est associée un objectif à optimiser. Les individus d'une même sous-population sont alors sélectionnés suivant l'objectif qui leur est affilié.

Nous donnons de ce qui suit la procédure de *Scheffer*:

1. *obj*=1, *P*' = ∅
2. **Pour** tout individu $i \in Pop$ **Faire** *f*(i) = f_{obj}(i)
3. **Pour** j=1 à *Taille_pop*/*n* **Faire** sélectionner un individu *i* sur la base de la fonction *f* et copier l'individu dans la population intermédiaire: *P'* = *P'*+{*i*}
4. Incrémenter *obj* = *obj* + 1
5. **Si** *obj* ≤ *n* alors aller à (2) sinon arrêter

Ce mécanisme est graphiquement représenté dans la figure III.8. Le procédé implique souvent la sélection des meilleures solutions par rapport à chaque objectif séparément. Cette technique est souvent référencée comme point de comparaison avec d'autres approches.

III.6.4 Somme pondérée variable de *Hajela* et *Lin*(HLGA)

Cette approche[Zit 99] appartient à la catégorie des techniques agrégatives. Proposée par *Hajela* et *Lin* en 1992 cette approche est basée sur la pondération des objectifs de l'optimisation au sein d'une même fonction globale. A fin de générer différentes solutions Pareto, les poids des objectifs varient d'un individu à un autre. Delà, chaque individu est évalué sur la base de sa propre fonction objectif globale. Les détails de l'évaluation des individus sont décrits dans la procédure suivante:

Pour tout individu *i* de la population *Pop*
Faire début
    Générer aléatoirement *n* nombres $\omega_1, \omega_2, \ldots, \omega_n$
    $f(i) = \omega_1 \times f_1(i) + \omega_2 \times f_2(i) + \ldots + \omega_n \times f_n(i)$

Pour améliorer la diversité de la recherche, *Hajela* et *Lin* proposèrent l'utilisation du *Sharing phénotypique* ainsi que la *restriction du voisinage* comme moyen d'accélération de la convergence. Cependant, et malgré sa simplicité d'implémentation, la méthode reste inappropriée face aux parties concaves de la frontière Pareto.

III.6.5 Algorithme génétique multicritère de Fonseca et Fleming(NDS)

En 1993, *Fonseca* et *Fleming* [Fon 95][Fon 95b] proposèrent un procédé de *ranking* basé sur la notion d'optimalité Pareto. Dans la *Non Dominated Sorting*(NDS) le rang d'un individu est égal au nombre de solutions qui le domine dans la population plus un. Le rang de l'individu jouera alors le rôle de la fonction d'utilité globale. La procédure suivante, décrit le principe d'évaluation dans cette approche.

1. **Pour** chaque individu *i* de la population *Pop* calculer $r(i)=1+|\{j \,/\, j \in Pop$ et *j* domine *i*$\}|$
2. Trier la population sur la base des rangs *r*.
3. Assigner à chaque individu *i* un niveau d'adéquation *f*(*i*) par interpolation depuis les meilleurs($r(i)=1$) jusqu'aux plus mauvais ($r(i) \leq Taille\_Pop$).

III.6.6 Niched Pareto Genetic Algorithm(NPGA)

Cette technique proposée par *Horn*, *Nafpliotis*[Hor 93] et *Goldberg* en 1994, combine les principes de la sélection par tournoi et de la dominance. Nous décrivons ci-dessous les deux étapes d'évaluation et de sélection:

γ : taille d'une niche

1. $P' = \emptyset$, ind=1
2. Sélectionner de façon aléatoire deux compétiteurs i et j et un ensemble de comparaison $P_{dom} \subseteq Pop$ de t_{dom} individus.
3. **Si** i est non dominer dans l'ensemble P_{dom} et que j ne l'est pas **alors** l'individu i est considéré vainqueur du tournoi : $P' = P' + \{i\}$
4. **Sinon Si** j est non dominer dans l'ensemble P_{dom} et que i ne l'est pas **alors** l'individu j est considéré vainqueur du tournoi : $P' = P' + \{j\}$
5. **Sinon** prendre une décision sur la base de partage phénotypique (*fitness sharing*)
   a) calculer les valeurs : $N(i) = |\{s / s \in P' \text{ et } dist(i, s) < \gamma\}|$ et
   $N(j) = |\{s / s \in P' \text{ et } dist(j, s) < \gamma\}|$
   b) **Si** $N(i) < N(j)$ **alors** $P' = P' + \{i\}$**sinon** $P' = P' + \{j\}$
6. $ind = ind + 1$. **Si** $ind \leq Taille\_Pop$ **alors** aller à (2).

Quand le procédé de sélection par tournoi n'arrive pas à départager les deux individus en compétition, le principe de *sharing* est alors utilisé. Il consiste à calculer pour chacun des deux compétiteurs le nombre d'individus déjà sélectionnés qui leur ressemblent(dans l'espace objectif). L'individu appartenant à une niche de population plus réduite sera finalement sélectionné.

III.6.7 Nondominated Sorting Genetic Algorithm(NSGA)

Dû à *Srinivas* et *Deb* (1994)[Deb 99], cette méthode se base sur la détermination progressive, des différents niveaux de dominance entre les solutions. Le partage phénotypique est réalise au niveau de chaque niveau séparément pour maintenir la diversité. L'évaluation des individus de la population passe par les étapes suivantes:

γ : taille d'une niche

1. $P_{reste} = Pop$ et
2. Calculer l'ensemble $P_{non\text{-}dominé}$ des solutions non dominées dans P_{reste}. Eliminer ces solutions de l'ensemble P_{reste} : $P_{reste} = P_{reste} - P_{non\text{-}dominé}$
3. Fixer le niveau d'adéquation des individus de $P_{non\text{-}dominé}$ à f_d puis effectuer la procédure du *sharing génotypique* seulement sur les dans $P_{non\text{-}dominé}$.
4. Augmenter la valeur de f_d pour qu'elle soit supérieure à la plus mauvaise évaluation dans $P_{non\text{-}dominé}$.
5. **Si** $P_{reste} \neq \emptyset$ **alors** aller à (2) **sinon** arrêter

Notons que dans un problème de minimisation, un individu a autant plus de chance d'être sélectionné que son évaluation est plus basse.

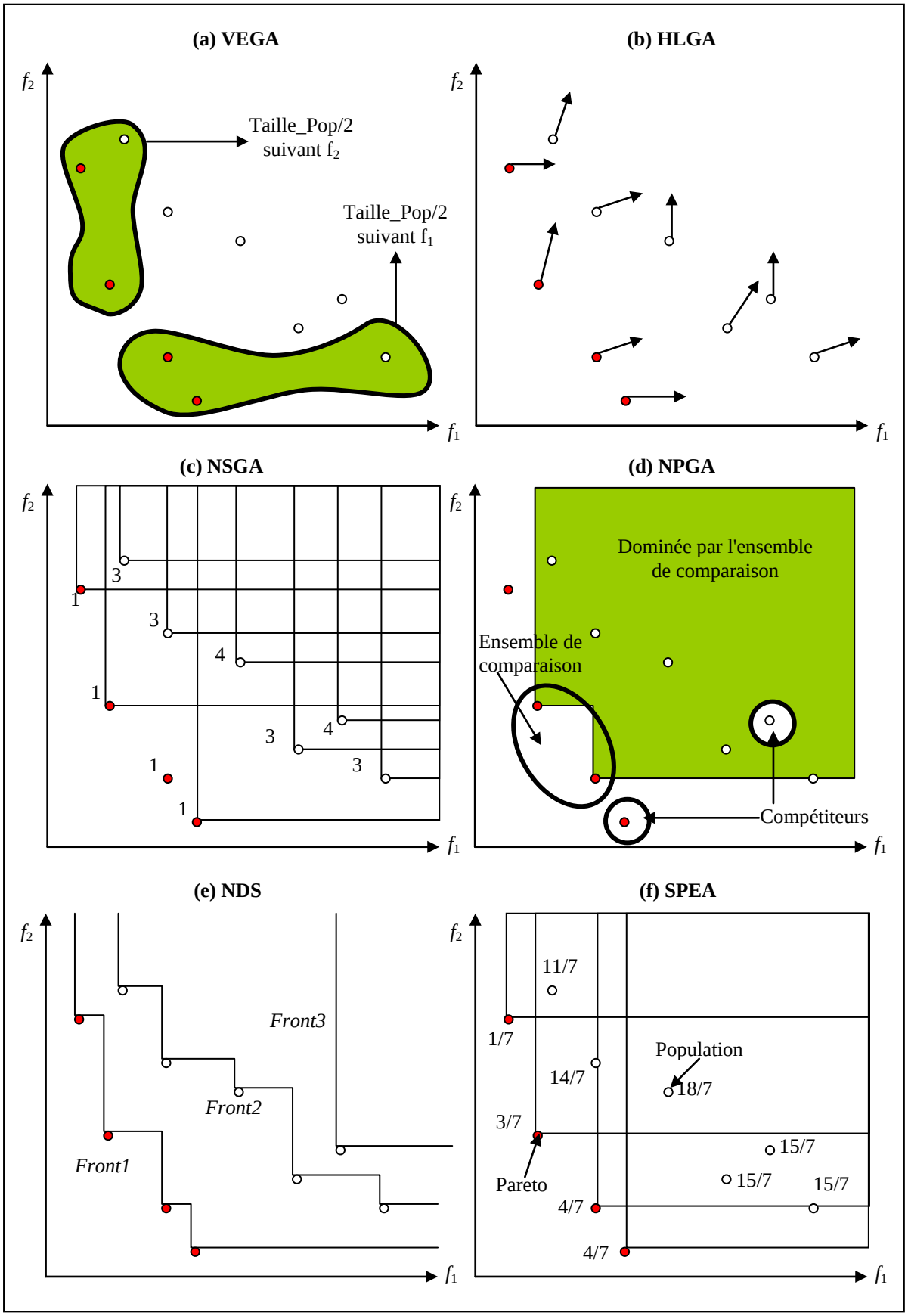


Figure III.9: Illustration de six stratégies de sélection dans l'espace objectif

III.6.8 Weighted Average Ranking(WAR)

De façon identique aux stratégies de ranking NDS et NSGA, cette approche consiste à ordonner les individus de la population selon leurs rangs. Les rangs des individus sont calculés en générant plusieurs listes ordonnées des individus, une par objectif. Le rang d'un individu revient alors à la moyenne de ses rangs par rapport à chaque objectif.

Bien que cette méthode n'utilise pas directement la notion de dominance, il est facile de prouver que :

$$(x \text{ domine } y) \Rightarrow \underset{i\in[1..n]}{moyenne}(r_i(x)) < \underset{i\in[1..n]}{moyenne}(r_i(y)) \qquad \text{(III.24)}$$

$r_i(x)$ désigne le rang de l'individu x suivant l'objectif i.

III.6.9 Strength Pareto Evolutionary Algorithm(SPEA)

Due à *zitzler* [Zit 98][Zit 99], la méthode dite *Strength Pareto Evolutionary Algorithm* est basée sur la combinaison d'anciennes et de nouvelles techniques afin de générer plusieurs solutions Pareto optimales en parallèle. Comme pour les autres techniques:

- Une population additionnelle est maintenue, elle archive les solutions Pareto trouvées toute au long de la recherche.
- Le concept de dominance est utilisé afin d'associer une valeur d'adéquation scalaire aux individus.
- Un clusturing des solutions archivées est réalisé afin de réduire leur nombre sans perdre la forme de la frontière Pareto.

D'autre part, la technique se distingue par

- La combinaison de trois méthodes en un seul algorithme.
- L'adéquation d'un individu est calculée à partir des solutions archivées seulement.
- Les solutions archivées participent aussi à la phase de sélection.
- Une nouvelle technique de maintien de niches est proposée, ne requirant aucune valeur de paramètre et se basant sur la notion d'optimalité Pareto.

Tout d'abord, la population Pareto archive est mise à jour. Ce qui revient à copier l'ensemble des solutions non dominées de la population dans l'ensemble Pareto et à supprimer éventuellement les éléments dominés de celle-ci. Qaund le nombre de solutions archivées dans l'ensemble Pareto dépasse une certaine limite, une représentation réduite est déduite par échantillonnage. Après l'évaluation des individus, la sélection par tournoi est effectuée sur les deux populations (Population courante et population Pareto) afin de produire la population intermédiaire P'. Les opérateurs de croisement et de mutation sont alors appliqués. La figure III.10 décrit schématiquement cette approche.

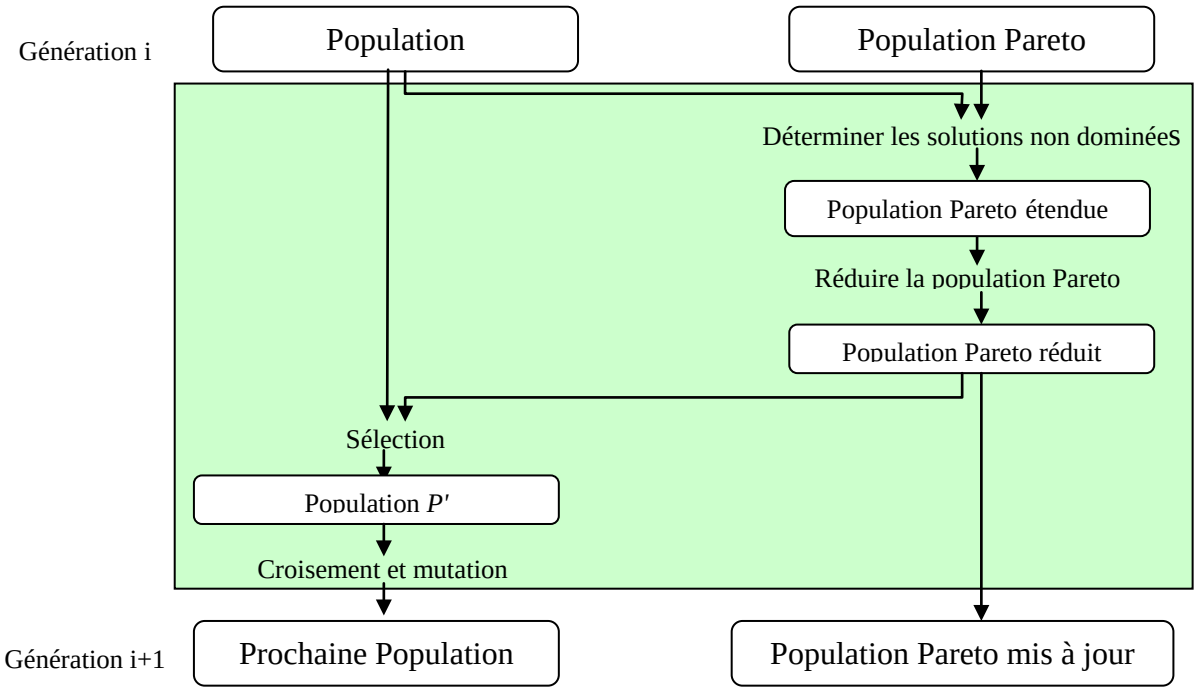


Figure III.10: Diagramme de l'algorithme SPEA

III.6.9.1 L'évaluation

La population Pareto correspond à l'ensemble des solutions élites. Cet ensemble contient les solutions non dominées trouvées pendant la recherche. L'évaluation d'un individu est déterminée en fonction des individus de la population courante et de la population Pareto. Ceci est réalisé suivant les étapes suivantes. Tout d'abord, une valeur réelle $s \in [0,1[$ appelée force(*strength*) est calculé pour chaque individu. La force d'un individu *i* de la population Pareto est proportionnelle au nombre d'individus de la population courante dominée par *i*. Par la suite, pour chaque individu de la population, les *strength* des solutions Pareto, par lesquelles il est dominé, sont sommés. Après quoi on rajoute 1 à la somme ainsi calculée.

```
Pour IndPareto ∈ Pareto
Faire
        i = 0
        Pour IndPop ∈ Population
        Faire
                Si Domine(IndPareto, IndPop) alors i = i+1
        Fait
        Strength(IndPareto) = i / (Taille_Pop + 1)
Fait
Pour IndPop ∈ Population
Faire
        somme = 0
        Pour IndPareto ∈ Pareto
        Faire
                Si Domine(IndPareto, IndPop) alors somme = somme + strength(IndPareto)
        Fait
        Strength(IndPop) = somme + 1
Fait
```

Une fois les forces des individus sont calculées, la sélection par tournoi est lancée ayant pour valeur d'adéquation les valeurs des forces.

III.6.9.2 Réduction de l'ensemble Pareto par clustering

Dans certains problèmes, l'ensemble Pareto optimale peut être extrêmement large. Dans ce cas la limitation de la taille de cet ensemble est nécessaire à cause des points suivants:

- De point de vu du décideur, la présentation de toutes les solutions Pareto trouvées est inutile quand leur nombre excède une certaine limite.
- Dans des cas d'optimisation de fonction continue, il est physiquement impossible et non nécessairement désirable de fournir toutes les solutions Pareto.
- Quand le nombre de solutions Pareto archivées est grand, ils tendent à avoir le même rang, entraînant une plus faible pression de sélection et du coup une lenteur de la recherche.

Ayant un nombre limite *Taille_Pareto* de solutions Pareto archivables, la technique du clusturing consiste en un regroupement itératif des solutions rapprochées en un nombre réduit de groupes($\leq$*taille_Pareto*). Une fois les groupes construit l'individu central du groupe est pris comme représentant du groupe entier dans l'ensemble Pareto. La procédure de clusturing est décrite par l'algorithme suivant:

```
Ensemble_cluster = ∅
Pour chaque IndPareto ∈ Pareto
Faire
        Ensemble_cluster = Ensemble_cluster ∪ {IndPareto}
Fait
Tanque |Ensemble_cluster| > Taille_Pareto
Faire
        Distance_min = + ∞
        Pour chaque {X, Y} ⊂ Ensemble_cluster
        Faire
                Si dist(X, Y) < Distance_min          alors
                        Cluster1 = X
                        Cluster2 = Y
                        Distance_min = dist(X, Y)
                Fin si
        Fait
        Nouveau_cluster = cluster1 ∪ cluster2
        Ensemble_cluster = Ensemble_cluster – {cluster1, cluster2}
        Ensemble_cluster = Ensemble_cluster ∪ {Nouveau_cluster}
Fait
```

III.6.10 Algorithme Génétique Multi-Sexuelle(MSGA)

Dans les algorithmes génétiques standards, les individus de la population sont du même sexe (ou n'ont pas de sexe). *Lis* et *Eiben* [Lis 96] proposèrent une variante des AGs où à chaque individu est associé un sexe. Dans cette approche, à tout objectif est associé un sexe donné. Initialement, les sexes sont attribués de façon aléatoire aux individus de la population initiale.

Les individus sont évalués suivant la fonction objectif associée à leur sexe. Du fait que la comparaison entre les individus ne peut être réalisée qu'au sein du même sexe, la sélection procède en deux étapes:

- Un sexe ***s*** est sélectionné aléatoirement, avec des probabilités égales.

- Un individu est sélectionné sur la base de son rang dans la communauté des individus de sexe ***s***.

L'opérateur de croisement multi-parents est adopté, portant sur *n* individus de sexes distincts. L'individu créé sera du même sexe que le parent dont il hérite le plus grand nombre de gênes. En cas d'égalité entre deux ou plusieurs parents, le sexe de l'un d'eux est pris aléatoirement. L'opérateur de mutation est restreint pour éviter à un individu de changer de sexe.

Contrairement aux deux méthodes de la sélection Parallèle(VEGA) et lexicographique, la prise en compte séparée des critères apparaît au niveau de la restriction de voisinage lors de la phase de reproduction.

Esquivel, Leiva et *Gallard*[Esq 99] adoptèrent une autre version de cette approche. Un croisement multiple par couple(Multiple crossover per couple) est adopté permettant plusieurs opérations de croisements par paire d'individus. La méthode présente une meilleure exploitation des solutions trouvées garantissant une recombinaison plus efficace des propriétés des parents.

III.6.11 Min-Max Algorithme Génétique (MOSES)

Coello[Coe 98] propose une approche qui se base sur la notion d'optimalité Min-Max. L'idée principale de cette approche est de générer les individus de façon à ce qu'ils soient tous faisables. Ceci, en contrôlant à chaque fois la satisfaction des contraintes et en adoptant des opérateurs de reproduction appropriés. Le décideur fixe plusieurs vecteurs de pondération, qui seront utilisés pour lancer différents processus. Chaque processus incorpore un algorithme de recherche génétique indépendant dont le but est d'optimiser la somme pondérée correspondante en conjonction avec le principe d'optimalité Min-Max et fournit une solution unique. Après la terminaison des *n* processus, l'ensemble des solutions non dominées parmi celles produites par les différents AGs sont recensées.

Coello propose un autre procédé[Coe 96], consistant en les étapes suivantes:

1. Une population initiale d'individus est générée, présentant que des individus faisables.

2. A chaque génération, l'ensemble des individus de la population est parcouru. Les meilleures évaluations obtenues pour chaque objectif sont recopiées au sein d'un même vecteur idéal.

3. Une variante de la sélection par tournoi est proposée. Au lieu de comparer l'adéquation des deux individus compétiteurs selon le critère de dominance, nous comparons les déviations maximales des deux individus par rapport au vecteur idéal. Si l'un des deux compétiteurs est meilleur que l'autre il sera le vainqueur du tournoi. En cas d'égalité le *sharing* est utilisé afin de départager les deux compétiteurs de façon similaire à la stratégie NPGA.

4. Les opérateurs de reproduction adaptés sont alors utilisés pour produire une nouvelle génération.

III.7 Algorithmes évolutionnistes Parallèles multicritère

Plusieurs implémentations des Algorithmes évolutionnistes multicritère parralèles ont été proposées[Vel 98]. Notre discussion prendra la forme d'une analyse des points parrallèlisables dans les AGs.

III.7.1 Evaluation parallèle

Premièrement, l'étape d'évaluation consiste à calculer l'adéquation des individus selon différents critères. Un temps additionnel est requis dû à la nécessité de scalairisation des vecteurs coûts. Les techniques d'attribution d'adéquation sont : le *ranking*, la scalairisation, l'échantillonnage indépendant et la recherche coopérative. L'évaluation des individus peut soit se faire en attribuant l'évaluation de chaque objectif à un processus à part, en répartissant l'évaluation des individus de la population sur plusieurs processus, ou en assignant la tâche d'évaluation de chaque individu à un processus à part.

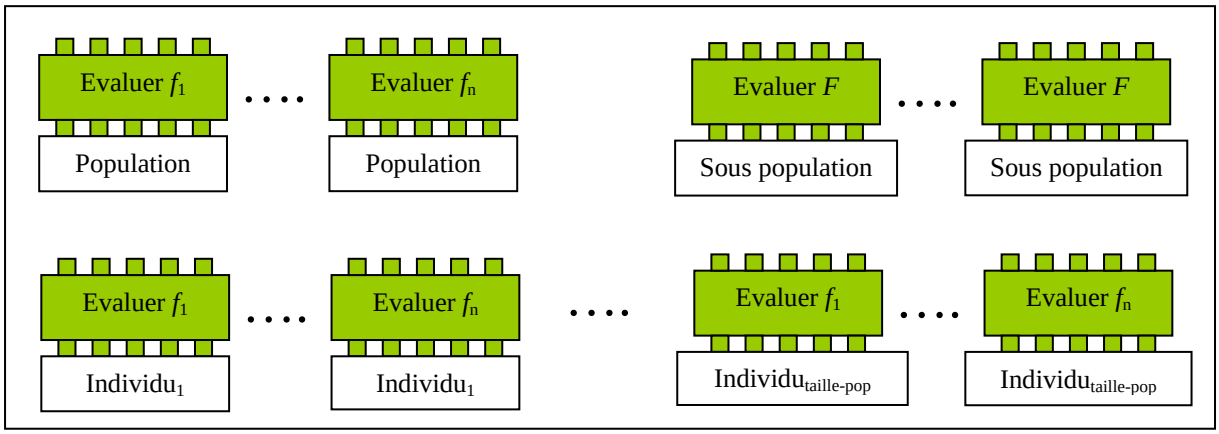


Figure III. 11: Les différentes possibilités de l'évaluation parallèle

III.7.2 Ranking parallèle

L'étape d'assignation de l'adéquation est aussi susceptible d'être parallélisée. Dans le cas du *ranking* par exemple, le rang de chaque individu implique la comparaison de celui-ci avec l'ensemble des individus de la population. De la même façon on peut choisir soit d'assigner le calcul du rang de chaque sous population à un processus différent ou de parallèliser totalement cette opération en attribuant le calcul du rang de chaque individu à un processus indépendant.

III.7.3 Sharing parallèle

Le procédé du *sharing* ne se prête pas bien à la parallélisation. En effet, la révision du vecteur coût des individus, revient au calcul des distances (phénotypique ou génotypique) séparant les solutions entre elles. La distance séparant une solution x de y étant identique à la distance entre y et x , la parallèlisation peut soit induire la redondance des calculs ou une synchronisation(donc des temps d'attente) entre les processus. Certaines variantes du *sharing* telle que celle proposée par *Fonseca* peut être parallèlisée de façon efficace. Dans cette version du *sharing*, les distances ne sont calculées qu'entre des individus de même rang. Delà, ont peut associer un processus chargé d'effectuer le *sharing* sur chaque sous population d'individus de même rang.

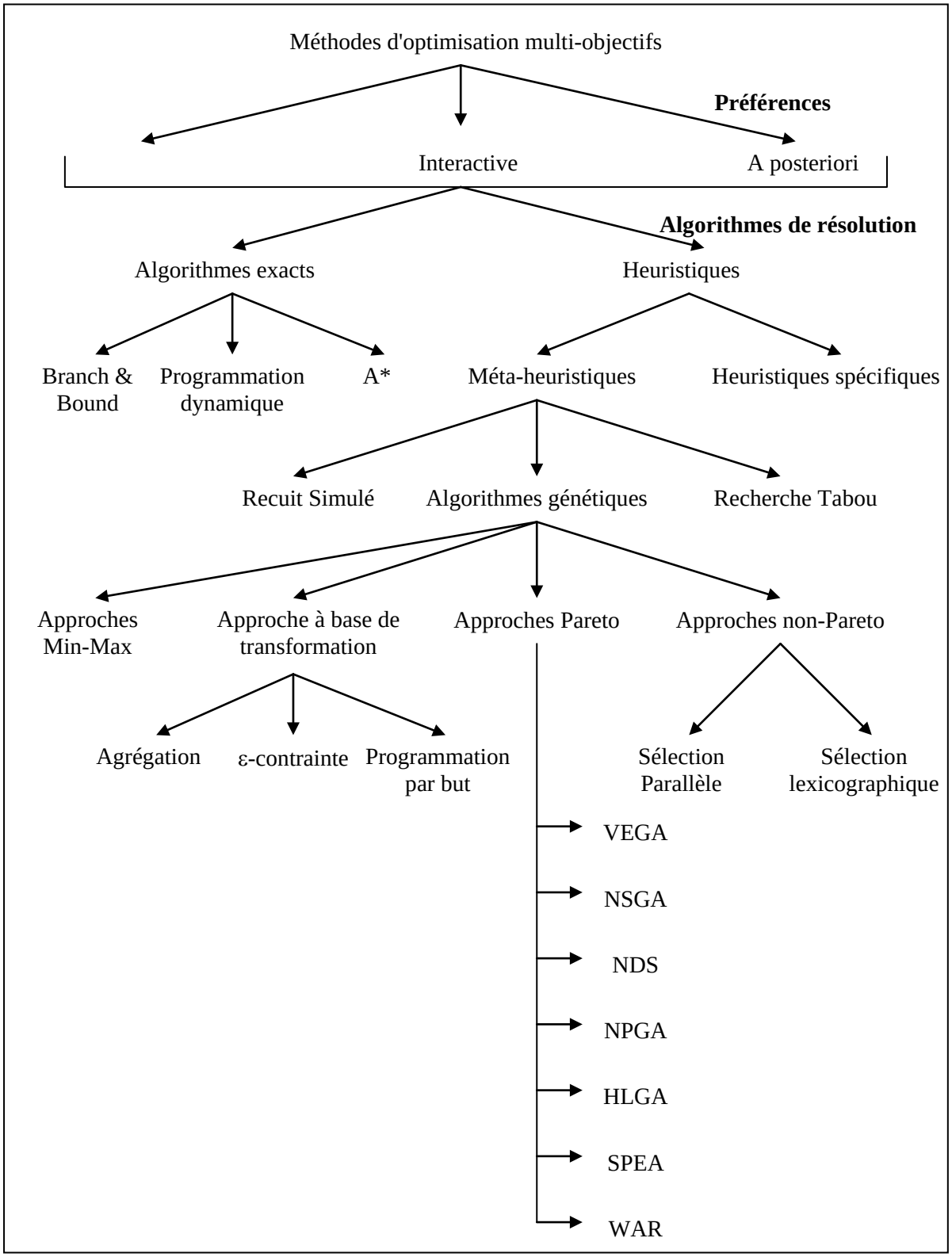


Figure III. 12: Classification des méthodes d'optimisation multicritère

III.8 Recherche Tabou multicritère

Les premiers travaux portants sur l'utilisation de la RT pour l'optimisation multicritère opéraient par transformation vers le monocritère. Des approches ultérieures, étendent les principes de la RT afin de produire une bonne approximation de la frontière Pareto.

III.8.1 Transformation vers le cas monocritère

Dans [Cos 94], la Recherche Tabou est appliquée pour résoudre le problème d'emploi du temps. L'algorithme est exécuté une seule fois moyennant un jeu de poids fixe. Delà une seule solution est produite.

III.8.2 Approche lexicographique

Dans [Her 94], la recherche tabou est utilisée pour résoudre le problème de formation de cellules de fabrication dans les ateliers. La méthode est appliquée pour optimiser une séquence de sous-problèmes monocritère sous contraintes. Les objectifs sont traités de façon séquentielle suivant l'ordre d'importance des critères. Le premier sous-problème résolu consiste à optimiser la fonction objectif la plus prioritaire. Le deuxième problème revient alors à optimiser la deuxième fonction objectif la plus importante. Ceci en maintenant satisfaite une contrainte supplémentaire qui consiste à ne pas détériorer, avec une certaine marge, la valeur obtenue pour la première fonction. Les autres critères sont ainsi traités en réitérant se procédé.

III.8.3 Approche de programmation par but

Gandibleux, Mezdaoui et *Fréville* proposèrent l'application de la RT pour des problèmes d'optimisation où les préférences de l'utilisateur sont connues dès le départ. Ces préférences sont alors formuler sous forme d'un vecteur idéal (point utopique selon la terminologie des auteurs) $F^*(f_1^*, f_2^*, ... f_n^*)$. L'objectif est alors de minimiser la fonction objectif globale f suivante :

$$\text{minimiser } f(x) = \left\| F(x) - F^*(x) \right\|_{p,\omega} = \left(\sum_{j=1}^{n} \left((\omega_j)^p \left(f_j(x) - f_j^* \right)^p \right) \right)^{\frac{1}{p}} \qquad \text{(III.25)}$$

Les poids ω_j sont introduits pour opérer un changement d'échelle ou pour signaler l'importance de certains critères par rapport à d'autres.

Lors de chaque itération, la solution voisine du point courant ayant la plus petite valeur de f et qui n'est pas tabou est choisie comme nouvelle solution courante.

L'objectif principal de l'algorithme est de générer une bonne approximation de l'ensemble Pareto optimal. A cet effet, la liste des M meilleures solutions voisines est considérée ($1 \leq M \leq |V(x)|$. A chaque itération, ces M solutions sont comparées aux différentes solutions présentes dans la population Pareto. Les solutions dominées de la population Pareto dans la liste sont éliminées. Si une solution de la liste est non dominée dans la population Pareto elle est insérée dans celle-ci.

III.8.4 Approche Pareto

Ben Abdelaziz, Krichen, Chaouchi [Ben 99]proposèrent une approche classique de la RT basée sur la notion de dominance. A chaque itération *it*, l'ensemble Pareto est mis à jour en fonction de l'ensemble non dominé des voisins de la solution courante $N(x_{it})$. La prochaine solution courante est choisie de façon aléatoire dans l'ensemble $N(x_{it})$.

Après chaque *d* itérations, une procédure de diversification est lancée. Cette procédure consiste à calculer le vecteur coût de la solution courante x_{it} est les deux critères les plus lésés i_1 et i_2 sont alors déterminés. L'algorithme *Greedy* est exécuté avec pour objectif:

$$\text{optimiser } f(x) = \lambda_1 f_{i1}(x) + \lambda_2 f_{i2}(x) \qquad \text{(III.26)}$$

Les paramètres $\lambda_1$et λ_2 sont pris dans l'une des valeurs suivantes : (0.8, 0.2), (0.5,0.5), (0.2,0.8).

L'algorithme de la Recherche Tabou est arrêté quand le nombre d'itérations *itmax* expire sans qu'aucune amélioration sur l'ensemble *Pareto* n'est observée.

III.9 Recuit Simulé multicritère

L'utilisation du recuit simulé dans le cadre de l'optimisation mutli-critères a premièrement été étudiée par *serafini* [Sar 92]. L'idée principale dans l'application du RS pour l'optimisation multi-objectifs consiste en l'utilisation d'une norme pondérée des composants du vecteur objectif, afin d'accepter ou rejeter une solution de coût inférieur.

III.9.1 Règle de calcul des probabilités de transition

En fonction de la manière de calculer la probabilité d'acceptation d'une solution, deux approches peuvent être envisagées. Dans la première approche dite à critère d'acceptation fort, seule les solutions dominantes sont sûres d'être acceptées. La deuxième approche (à faible critère d'acceptation) adoucit cette contrainte en acceptant directement les solutions non dominées(une solution non dominée n'est pas obligatoirement dominante).

III.9.1.1 Trie agrégatif

Ce mécanisme [Sar 92] s'apparente aux techniques agrégatives où les différents critères sont combinés au sein de la même fonction objectif globale. Delà le principe d'acceptation classique est utilisé.

- *Règle Scalairisation linèaire:* La probabilité d'acceptation d'une solution *y* partant de la solution *x* à la température *T* est:

$$P(y,T) = e^{\sum_{i=1}^{n} \omega_i (f_i(x) - f_i(y))/T} \qquad \text{(III.27)}$$

III.9.1.1.1 Approche de Tuytten

Tuytten [Tuy 98] propose cette technique pour la résolution du problème du sac à dos multi-objectifs. La solution initiale est construite en utilisant l'algorithme de *Greedy*. Pour obtenir une approximation de l'ensemble Pareto optimale, une population archive est maintenue qui contient les solutions non dominées trouvées tout au long de la recherche. Initialement cet ensemble ne contient que la solution initiale x_0. L'algorithme du RS multicritère est décrit dans la procédure suivante:

1. Générer la solution initiale x_0.
2. Pareto={x_0}; *it*=0; $T_0 = T_{init}$

A l'itération *it*

3. Générer une solution $y \in V(x_{it})$
4. Evaluer $F(y)$.
5. Générer un nombre aléatoire $p \in [0,1]$.
6. Si $p < e^{\sum_{i=1}^{n} \omega_i (f_i(x_{it}) - f_i(y))/T_{it}}$ alors accepter y: $x_{it+1}=y$.
7. Mettre à jour l'ensemble Pareto
8. $it = it + 1$; si it modulo $N_{étape} = 0$ alors $= \alpha\, T_{it-1}$ sinon $T_{it} = T_{it-1}$.

III.9.1.1.2 Approche de Ulungu, Teghem et Ost

L'approche proposée[Ulu 98] est identique à la précédente, excepté dans les points suivants:

1. L'algorithme est lancer plusieurs fois avec des jeux de poids différents. Les poids sont calculés suivant les préférences du décideur ce qui fait de cette méthode une approche fortement interactive. Le calcul des poids λ_i se fait sur la base de la formule suivante:

$$\lambda_i = \frac{m_i - g_i}{m_i - M_i} \tag{III.28}$$

   m_i et M_i désignent respectivement la plus mauvaise et la meilleure valeur obtenue pour l'objectif i. Les paramètres g_i sont fournis par le décideur est désignent les bornes supérieures des objectifs ($f_i(x) \leq g_i$).

2. Plusieurs ensembles Pareto sont alors générés, ces ensembles sont fusionnés afin de ne laisser que les solutions non dominées.

- *Règle de programmation par but:*

$$P(y,T) = e^{\left(\max_{i \in [1..n]} \omega_i (f_i(x) - r_i) - \max_{i \in [1..n]} \omega_i (f_i(y) - r_i) \right)/T} \tag{III.29}$$

Le vecteur $r=(r_1, r_2, \ldots, r_n)$ représente le vecteur idéal à approcher.

III.9.1.2 Trie Pareto

Le principe d'acceptation est révisé pour introduire la notion d'optimalité Pareto. Dans les approches à critère d'acceptation fort, le critère d'acceptation peut être formulé sous la forme suivante:

- *Règle du Produit Simple:*

$$P(x,T) = \prod_{i=1}^{n} \min\left\{1; e^{(f_i(x) - f_i(y))/T}\right\} \tag{III.30}$$

- *Règle de Cebisev:*

$$P(x,T) = \min\left\{1; \min_{i \in [1..n]} e^{(f_i(x) - f_i(y))/T}\right\} \tag{III.31}$$

Avec l'approche à critère d'acceptation faible, toute solution qui n'est pas strictement dominée serait certainement acceptée.

- *Règle de critère faible:*

$$P(x,T) = \min\left\{1; \max_{i\in[1..n]} e^{(f_i(x)-f_i(y))/T}\right\} \quad \text{(III.32)}$$

III.10 Conclusion

La difficulté des problèmes d'optimisation multicritère est principalement dueA à l'ambiguïté qui caractérise la notion d'optimalité. Deux principales définitions se retrouvent dans la littérature: optimalité Pareto et optimalité Min-Max. Cependant la notion Pareto semble bénéficier d'une certaine unanimité.

La classification des techniques d'optimisation multi-objectifs fait intervenir trois aspects. D'une part, le type d'interaction entre algorithme de recherche et décideur (aspect aide à la décision). D'autre part, l'objectif de la recherche elle-même: recherche exacte de l'ensemble des solutions Pareto optimale ou recherche approchée de cet ensemble. Le troisième aspect, concerne l'utilisation ou non de la notion d'optimalité Pareto lors de la recherche.

Parmi les méta-heuristiques utilisées pour la résolution des problèmes multicritère, les techniques de recherche globale (à base de population) telles que les algorithmes génétiques semblent les plus adéquates. En effet, l'utilisation d'une population permet d'approximer par une seule exécution l'ensemble des solutions Pareto contrairement au techniques de recherche locale (RT et RS) où il faut exécuter la recherche plusieurs fois afin d'obtenir un ensemble de solutions Pareto.

Les stratégies de diversification notamment le sharing sont des mécanismes déterminant de l'efficacité de la recherche. La diversification permet de mieux échantillonner la frontière Pareto. En plus et dans certains cas, l'ensemble des solutions Pareto est tellement grand qu'un procédé tel que le sharing permet d'éliminer les solutions rapprochées.

Notre intérêt à porté sur l'étude de l'application des AGs dans le domaine de l'optimisation multicritère. Plusieurs stratégies de sélections seront comparées. Pour l'analyse de l'apport de la diversification sur la recherche, les trois variantes du sharing sont développées et leurs performances respectives testées et comparées. D'autres mécanismes tels que l'hybridation, la génération heuristique de la population initiale et la parallélisation seront aussi le sujet de cette étude. Le problème de flow shop à permutation est pris comme exemple pour l'implémentation de ces mécanismes.

Partie IV: Algorithmes génétiques Multicritère pour les problèmes de Flow Shop

IV.1 Introduction

Les problèmes d'ateliers tels que le Flow Shop ou le Job Shop sont des problèmes dont la résolution vêtit une grande importance dans le domaine industriel. Les critères à optimiser sont généralement la minimisation du temps de terminaison ou celui des retards. Cependant, rares sont les approches de résolution qui prennent en compte ces différents critères à la fois. Cette partie de notre travail présente une étude comparative entre plusieurs approches basées sur les Algorithmes Génétiques adaptés au cas Multicritère. Diverses stratégies de sélection, de maintien de la diversité de la recherche et d'hybridation seront implémentées. Leurs performances seront testées et comparées. Un modèle d'algorithmes génétiques parallèle est proposé, permettant d'augmenter la taille de la population et le nombre de génération limite et conduisant ainsi à de meilleurs résultats. Finalement, l'heuristique NEH est utilisée lors de la génération de la population initiale.

La difficulté du cas Multicritère réside dans l'absence d'une relation d'ordre totale qui lie l'ensemble des solutions du problème. Sur le plan des AGs, ce manque apparaît dans la difficulté de concevoir un opérateur de sélection qui affecte des probabilités de sélection proportionnelles aux degrés de souhaitabilité des individus dans la population. Un autre inconvénient est lié à la sensibilité quant au choix de la population initiale qui caractérise les AGs ainsi qu'aux mauvais échantillonnages lors de la sélection. Cette sensibilité entraîne souvent la perte prématurée de la diversité ainsi que l'instabilité de la recherche. D'où la nécessité de concevoir des techniques de maintien de la diversité au sein de la population.

La première partie de ce travail est consacrée à la présentation du problème de Flow Shop Multicritère. Nous discuterons des différents paramètres à optimiser ainsi que des contraintes à satisfaire. Dans la deuxième partie, nous exposons les différentes possibilités d'extension des AGs au cas multicritère. Nous présentons aussi les différents choix de codage, de fonctions objectifs et d'opérateurs Génétiques. Dans la troisième partie, différentes stratégies de sélection seront présentées et leurs performances comparées. L'apport de l'élitisme sera particulièrement analysé. Lors de la quatrième partie, nous discuterons des méthodes de maintien de la diversité implémentées ainsi que leurs apports sur la qualité des solutions trouvées. La cinquième partie sera consacrée à la présentation de l'hybridation de l'AG Multicritère avec la recherche locale. L'apport de l'hybridation sera ainsi souligné. La sixième partie compare notre algorithme à d'autres travaux de la littérature, pour cela nous proposons un algorithme générateur de benchmarks de flow shop bi-critères. Dans la septième partie, nous exploitons la possibilité de parallélisation des AGs. La dernière partie est dédiée à l'introduction d'une nouvelle approche de génération de la population initiale basée sur l'heuristique NEH.

IV.2 Problème de Flow Shop Multicritère

Le problème de flow shop est décrit dans le paragraphe I.11.3. Notre intérêt a porté sur l'étude du problème de Flow Shop à permutation *F/perm,d_i/(C_{max} ,T),* où les jobs doivent être ordonnés dans le même ordre sur toutes les machines. La figure ci-dessous présente un ordonnancement de 7 jobs sur 3 machines. Comme on peut l'observer les jobs sont traités dans la même disposition.

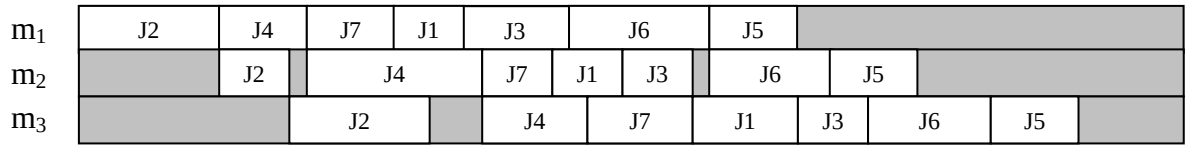


Figure IV.1 : Exemple d'ordonnancement

IV.3 AG et Problème de Flow Shop Multicritère

IV.3.1Codage

L'application des AGs à un problème donné nécessite dans un premier temps une représentation chromosomique d'une solution (dans notre cas un ordonnancement des jobs). La séquence de passage des jobs dans les machines étant identique il suffit donc de présenter le séquencement des jobs en une seule machine. Par conséquent, un ordonnancement est vu comme une permutation définissant l'ordre de passage des jobs sur les machines. L'exemple de la figure IV.1 est représenté suivant ce codage par un vecteur associant une entrée à chaque job. La position d'un job dans le chromosome définit son numéro d'ordre dans la séquence.

2	4	7	1	3	6	5

Figure IV.2 : représentation chromosomique de l'exemple de la figure IV.1

IV.3.2Détermination des dates de début des tâches

L'évaluation d'une séquence donnée, nécessite le calcul des dates de début et de fin des tâches. Etant donné que les critères à optimiser sont réguliers, ce calcul est réalisé par construction de l'ordonnancement au plus tôt des tâches. Le calcul se fait d'une manière récursive, à commencer par les tâches planifiées en premier, comme suit:

$$P_{ii} = \begin{cases} 0 & \textbf{Si } J_i \text{ est le premier job de la permutation et } j=1 \\ P_{i\,(i-1)} + d_{i\,(i-1)} & \textbf{Si } J_i \text{ est le premier job de la permutation et } j\neq 1 \\ P_{i`j} + d_{i`j} & \textbf{Si } J_i \text{ n'est pas le premier job de la permutation et } j=1 \\ max(P_{i\,(i-1)} + d_{i\,(i-1)},\, P_{i`j} + d_{i`j}) & \textbf{Sinon} \end{cases} \quad \text{(IV.1)}$$

P_{ij} désigne la date de début de la tâche t_{ij} . $J_{i'}$ désigne le job qui précède J_i dans la permutation.

Cette formule exprime le fait qu'une tâche t_{ij} ne peut être lancée que suite à la libération de la machine m_j par la tâche précédente $t_{i'j}$ et la fin de la tâche antérieure dans le même job $t_{i(j-1)}$.

La procédure de calcul des temps de début P_{ij} des tâches t_{ij} est décrite par l'algorithme suivant :

```
/* calcul des temps de début des tâches t_i1 s'effectuant sur la première machine */
libre : = 0 /* date de libération de la machine une */
pour i : = 1 à N      /* S[i] désigne la i ème entrée dans le chromosome S */
faire début
        p S[i] 1 : = libre
        libre := p S[i] 1 + d S[i] 1
    fin
/* calcul des temps de début des taches tij s'effectuant sur les autres machines */
pour j : = 2 à M  /* pour chaque machine */
faire début
   libre : = 0
        pour i : = 1 à N
        faire début
          p S[i] j : = maximum ( libre, p S[i] j-1 + d S[i] j-1 )
          libre : = p s[i] j + d s[i] j
              fin
    fin
```

IV.3.3Fonctions Objectifs

Les critères d'optimisation pris en compte se résument à deux objectifs : la minimisation du temps de terminaison total (*makespan*) ainsi que de la somme des retards enregistrés. Ces critères se présentent comme les critères les plus considérés dans le domaine de l'ordonnancement.

$$f_1 = C_{\max} = \max_{i \in [1..N]} (P_{iM} + d_{iM}) \qquad \text{(IV.2)}$$

$$f_2 = T = \sum_{J_i \in jobs} \left[\max(0, P_{iM} + d_{iM} - l_i) \right] \qquad \text{(IV.3)}$$

A chaque ordonnancement possible S est associé un vecteur objectif $F(S) = (f_1(S),\ f_2(S))$ qui représente l'évaluation de la solution.

IV.3.4Génération de la population initiale

Notre première procédure de génération de la population initiale consiste en une production aléatoire de plusieurs permutations. L'aspect aléatoire nous garantit relativement une bonne répartition des individus dans l'espace de recherche. La procédure suivante décrit le procédé de génération de la population initiale.

```
Pour tout individu ind de la population
Faire Début
    Ind = ∅
    Possibilité = {1, 2, ..,N}
    Tant que Possibilité ≠ ∅
    Faire Début
        Choisir aléatoirement un job J parmi ceux de l'ensemble Possibilité
        Ind  = ind→J
        Possibilité = Possibilité – {J}
    Fin
Fin
```

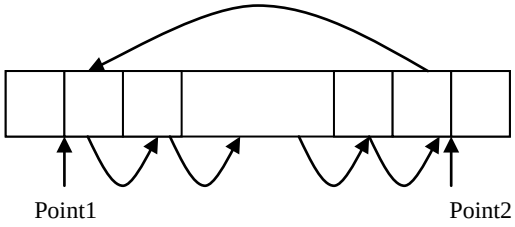


Figure IV.3 : Opérateur de Mutation

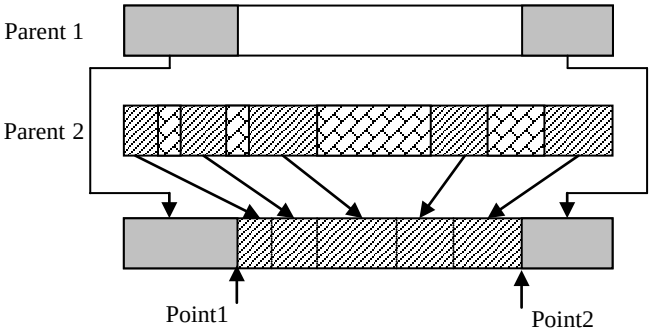


Figure IV.4 : Opérateur de Croisement

IV.3.5Opérateurs Génétiques

Appliquer la méthode des AGs à un problème donné nécessite aussi le choix des opérateurs génétiques qui servirons à faire évoluer la recherche. Nous, nous sommes inspirés des opérateurs définis par *Murata* et *Ishibuchi*[Mur 98]. L'opérateur de mutation et de croisement sont décrit dans les figures IV.3 et IV.4.

L'opérateur de Mutation consiste au choix aléatoire de deux points dans le chromosome. Une rotation est alors effectuée comme indiquée dans la figure IV.3. L'opérateur de croisement aussi nommé "*croisement à deux points*" consiste au choix aléatoire de deux points de croisement. Un individu fils est alors généré. Il est similaire au niveau de ses extrémités, au *parent1* et dont le reste des jobs est pris dans l'ordre rencontré dans le *parent2* (figure IV.4).

Nous donnant ici les procédure de mutation et de croisement:

```
Procédure mutation (individu : INDIVIDU)
Début
    Choisir un nombre aléatoire p∈[0,1[
    Si p<Pm
    alors Début                                          { effectuer la mutation }
          choisir aléatoirement un job Ji où i ∈ [1..N]
          choisir aléatoirement un job Ji où j ∈ [1..N] et j ≠ i   { i et j représentent les points de coupure}
          si i > j alors x = i ; i = j ; j = x           {mettre i < j }
          x = j
          pour k allant de j à i                         {k est décrémenté chaque fois }
              faire individu[k] = individu[k-1]          {effectuer la rotation }
          individu[i] = x
        Fin
Fin

Fonction croisement (individu1, individu2 : INDIVIDU) : INDIVIDU
Début
    Choisir un nombre aléatoire p∈[0,1[
    Si p<Pc
    alors Début                                          { effectuer le croisement}
          choisir aléatoirement un job Ji où i ∈ [1..N]
          choisir aléatoirement un job Ji où j ∈ [1..N] et j ≠ i   { i et j représentent les points de coupure}
          si i > j alors x = i ; i = j ; j = x           {mettre i < j }
          x = j
          pour k allant de 1 à i et allant de j à N      {copier les extrémité du premier parent dans
                                                          l'individu fils}
              faire FILS[k]=individu1[k]
          pos = i + 1                                    {remplir le centre de l'individu fils avec les jobs
                                                          non encore pris}
          pour k allant de 1 à N
```

```
            faire Début
                    pris = faux
                    pour r allant de 1 à i
                        faire si FILS[r] = individus2[k] alors pris = vrais et sortir de la boucle
                    si pris = faux
                    alors pour r allant de j à N  faire si FILS[r] = individus2[k] alors pris = vrais et sortir de la
                    boucle
                    si pris = faux alors FILS[pos] = individus2[i] et incrémenter pos
                  Fin
          Fin
    Retourner FILS
Fin
```

IV.4 Opérateur de sélection

L'absence d'une relation d'ordre total entre les différentes solutions possibles d'un problème multicritère, fait que la notion d'optimalité tel que conçu pour le cas uni-critère n'est plus utilisable. La notion d'optimalité Pareto tirée de la notion de dominance mathématique des vecteurs, jouit d'une grande adhésion de la part des chercheurs.

IV.4.1Dominance et optimalité Pareto

Une définition qui a fait son chemin est la notion de dominance ou d'optimalité Pareto. La dominance représente une relation d'ordre partiel sur l'ensemble des points de l'espace de recherche.

S_i domine S_j SSI $\forall\ k \in [1..nb\_obj]\ f_k(S_i) \leq f_k(S_j)$ et $\exists\ k \in [1..nb\_obj] / f_k(S_i) < f_k(S_j)$ (IV.4)

S_i est Pareto optimale SSI $\forall S_j / S_j \neq S_i$ on a S_j ne domine pas S_i (IV.5)

Où *nb_obj* désigne le nombre d'objectifs à optimiser.

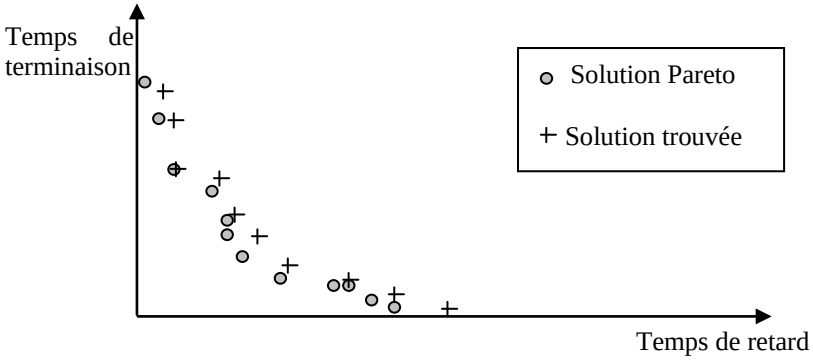


Figure 5 : Solutions non dominées

De façon intuitive, une solution S_i est dite "*dominant*" la solution S_j si et seulement si : S_i est au moins meilleure que S_j sur tous les plans (par rapport à l'ensemble des objectifs d'optimisation) et qu'elle soit préférable à S_j au moins pour un objectif. Le but de l'optimisation multicritère est d'aboutir à un ensemble de solutions non dominées qui approxime au mieux l'ensemble des solutions Pareto du problème.

IV.4.2AGs et Problèmes d'optimisation multicritère

Un vaste éventail de techniques d'adaptation des AGs au cas multicritère se retrouvent dans la littérature[Tal 99][Zit 99]. Les différences principales entre ces méthodes consistent en le moment où les différents objectifs sont pris en compte :

1. *La phase de sélection* : l'aspect multicritère apparaît dans la façon dont les individus sont triés suivant leur ordre d'adéquation en vu d'être sélectionnés
    a. *La sélection lexicographique* : se base sur un ordre de priorité entre les objectifs préétablie par le décideur[Fou 85].
    b. *La sélection par Ranking* : se base sur le calcul des rangs des individus de la population en s'appuyant soit sur la notion de dominance tel que la sélection *NSGA*[Sri 95] et *NDS*[Fon 95b], soit sur la valeur des fonctions objectifs telles que la sélection *WAR*[Ben 97] ou la sélection par somme pondérée des objectifs[Haj 92].
    c. *La sélection parallèle* : A chaque génération, la population est subdivisée en *nb_obj* sous-populations de tailles égales. A chaque sous-population est associée un objectif à optimiser. Les individus d'une même sous-population sont alors sélectionnés suivant l'objectif qui leur est affilié[Sch 85].
    d. *La sélection par somme pondérée des objectifs à poids variables:* Adopté par *Murata* et *Ishibuchi*[Mur 98], cette méthode consiste lors de chaque génération à générer de façon aléatoire un ensemble de *nb_obj* nombres ω_1, ω_2, ... ω_{nb_obj} compris entre [0,1] tel que $\omega_1+\omega_2+ ...+\omega_n = 1$. Les individus de la population sont alors sélectionnés suivant la formule $\omega_1 f_1(S)+\omega_2 f_2(S)+... +\omega_{nb\_obj} f_{nb\_obj}(S)$ pendant cette génération là.
2. *La phase de reproduction* (*Reproduction Multi-sexuelle* [All 92]): Un objectif correspond dans ce cas à un sexe donné. A chaque individu de la population est associé un sexe dès sa génération (après génération de la population initiale ou suite à une reproduction). Les individus sont admis à la reproduction (croisement) seulement s'ils appartiennent à des sexes différents.

IV.4.3Stratégies de sélection implémentées

Pour notre étude nous avons implanté six stratégies de sélection multicritère. Les différences principales entre ces méthodes résident dans la façon dont les individus de la population sont ordonnés, ainsi que l'expression permettant le calcul des probabilités de sélection.

IV.4.3.1 Sélection par somme pondérée des objectifs

C'est l'une des premières méthodes utilisées dans le cadre de l'optimisation multicritère. Basée sur la transformation du problème en un problème uni-critère, cette méthode consiste à combiner les différentes fonctions objectifs en une seule fonction généralement de façon linéaire.

$$f(S_i) = \sum_{k\in[1..2]} \lambda_k f_k(S_i) \qquad \text{(IV.6)}$$

Les poids λ_k sont pris dans l'intervalle [0..1] de façon expérimentale, tel que $\sum_{k\in[1..2]} \lambda_k = 1$. Le rang d'un individu est dans ce cas égal à $f(S_i)$.

$$Rang(S_i) = f(S_i) \qquad \text{(IV.7)}$$

Un individu S_i a alors une probabilité d'être sélectionné égale à :

$$\wp(S_i) = \frac{Rang(S_i)}{\sum_{j\in[1..tp]} Rang(S_i)} \qquad \text{(IV.8)}$$

Où *tp* désigne la taille de la population courante.

IV.4.3.2 Sélection Parallèle

Dans la sélection parallèle, la moitié des éléments sélectionnés le sont sur la base de leurs coûts de temps de terminaison. Les *tp*/2 autres individus sont sélectionnés sur la base de leur coût de retard. La procédure ci-dessous décrit ce principe.

- Trier les individus de la population courante selon l'ordre croissant de f1 et soit Rang1(i) le rang de l'individus i.
- Trier les individus de la population courante selon l'ordre croissant de f2 et soit Rang2(i) le rang de l'individus i.
- **Pour** i : = 1 à *tp*/2
  Faire début
    Sélectionner un individu de la population courante avec la probabilité $Rang1(i) \Big/ \sum_{j\in[1..tp]} Rang1(j)$
    Insérer cet individu dans la population intermédiaire
    fin
- **Pour** i : = 1 à *tp*/2
  Faire début
    Sélectionner un individu de la population courante avec la probabilité $Rang2(i) \Big/ \sum_{j\in[1..tp]} Rang2(j)$
    Insérer cet individu dans la population intermédiaire.
    fin
- Effectuer la reproduction sur la population intermédiaire.

IV.4.3.3 Sélection NSGA

Dans la sélection *NSGA* (*Non Dominated Sorting Genetic Algorithm*) les rangs des individus sont calculés d'une manière récursive à commencer par les individus dominant de la population.

On associe le rang 1 à l'ensemble des individus non dominés E_1 de la population courante.

On associe le rang 2 à l'ensemble des individus E_2 qui ne sont dominés que par des individus appartenant à E_1.

On associe le rang 3 à l'ensemble des individus E_3 qui ne sont dominés que par des individus appartenant à $E_1 \cup E_2$.

.....

On associe le rang *k* à l'ensemble des individus E_k qui ne sont dominés que par des individus appartenant à $E_1 \cup E_2 \cup \ldots E_{k-1}$.

La probabilité de sélection d'un individu S_i de rang *n* dans la population suit l'expression de *Baker* [Tal 99] suivante :

$$\wp(S_i) = \frac{S(tp+1-R_n)+R_n-2}{tp(tp-1)} \qquad \text{(IV.9)}$$

Où S désigne la pression de sélection et

$$R_n = 1 + |E_n| + 2 * \sum_{j \in [1..n-1]} |E_j| \quad \text{(IV.10)}$$

IV.4.3.4 Sélection NDS

Dans la sélection *NDS* (*Non Dominated Sorting*) le rang d'un individu est identique au nombre de solutions dominant l'individu plus un.

$$Rang\ (S_i) = |\ S_j \in Population / S_j \text{ domine } S_i| + 1 \quad \text{(IV.11)}$$

La probabilité de sélection est calculée suivant le même procédé que celui de la formule (IV.9)

IV.4.3.5 Sélection WAR

La *Weighted Average Ranking* consiste à calculer le rang de chaque individu de la population par rapport aux différents objectifs séparément. Le rang d'un individu est alors pris comme étant la somme des rangs. Pour cela on commence par ordonner les individus par ordre croissant de la fonction *f1* et par ordre croissant de la fonction *f2*.

$$Rang(S_i) = Rang\_Terminaison(S_i) + Rang\_Retard(S_i) \quad \text{(IV.12)}$$

Les probabilités de sélection sont alors calculer comme pour la sélection *NSGA*.

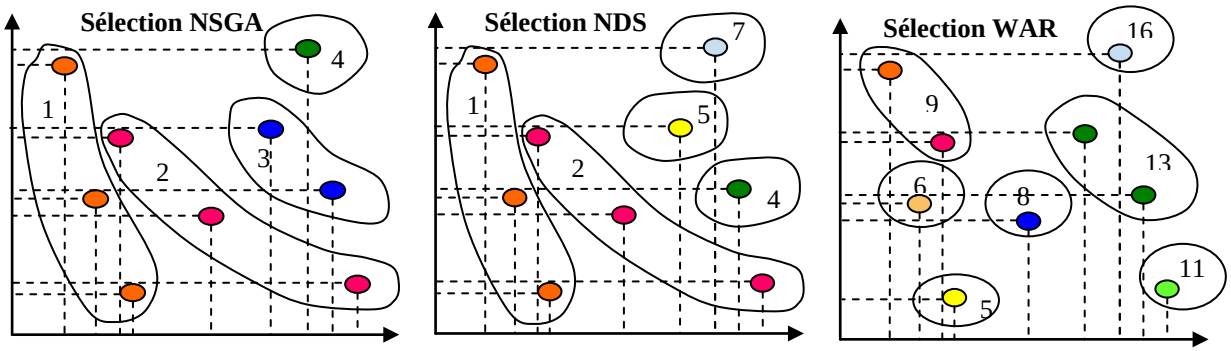


Figure IV.6 : Comparaison entre Sélection *NSGA* et *NDS* et *WAR*

IV.4.4Sélection élitiste

La sélection élitiste consiste à maintenir une sorte de population archive *PO** qui contiendra les meilleures solutions non dominées rencontrées tout au long de la recherche. Cette population participera aux étapes de sélection et de reproduction. Dans ce cas la probabilité de sélection d'un individu S_i de la population courante de rang *n* correspondra à l'expression suivante :

$$\wp(S_i) = \frac{(tp - A)}{tp} \times \frac{S(tp - 1 - R_n) - 2 + R_n}{tp(tp - 1)} \quad \text{(IV.13)}$$

Ce qui laisse une probabilité de *A*/*tp* de choisir un élément de la population Pareto. *"A"* détermine de ce fait l'espérance du nombre d'individus sélectionnés à partir de l'ensemble *PO**.

En réalité toutes les méthodes de sélection garde un archive contenant les meilleures solutions rencontrées pendant la recherche(Ensemble Pareto). La particularité de l 'élitisme est de faire participer cette population lors de la phase de sélection. Le rang des individus est calculé suivant la technique *NSGA*. La procédure suivante décrit le processus de sélection d'un individu de la population courante dans le cas de la sélection élitiste.

Choisir un nombre aléatoire p compris dans l'intervalle [0, 1]
Si p>(tp - A) / tp
Alors début
i=1 ; T = $\wp(S_1)$
Tant que p≥T
Faire début
i = i+1; T = T + $\wp(S_i)$
Insérer l'individu S_i dans la population intermédiaire
Fin
Fin
Sinon sélectionner de façon aléatoire un individus quelconque de la population Pareto (PO*)

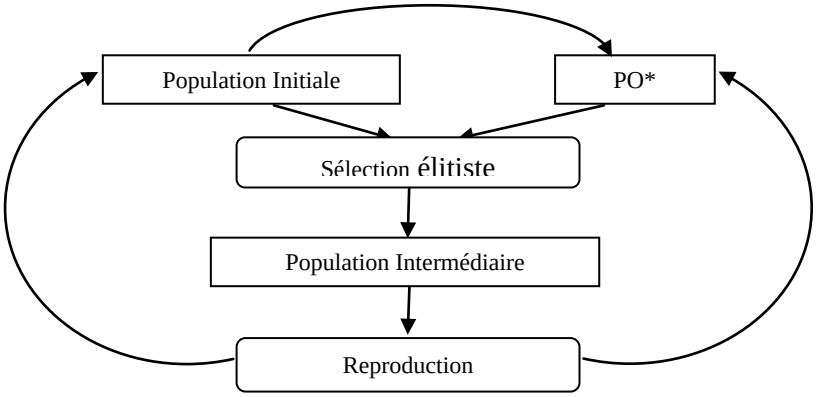


Figure IV.7 : Stratégie de sélection élitiste

Nous décrivons ici les principaux composants de l'AG.

1. Générer les individus de la population initiale de façon aléatoire.
2. Evaluer les individus de la population {Calcul des vecteurs coût F de chaque individu}
3. Calculer l'ensemble Pareto Optimal PO* de la Population Courante.
4. **Tant que** le nombre de générations limite n'est pas atteint
   Faire début
   a. Calculer les rangs des individus de la population selon le type de sélection adopté et sur la base des vecteur F.
   b. Calculer les probabilités de sélection de chaque individu.
   c. Sélectionner *tp* individus de la population courante et éventuellement de la population PO* (sélection élitiste) suivants leurs probabilités, et constituer la population intermédiaire.
   d. **Pour** i := 1 à *tp* */* Phase de croisement */*
   Faire début
   e. Sélectionner une paire d'individus de la population intermédiaire. Suivant un probabilité P_c choisir soit
   (a) réaliser le croisement et insérer l'individu fils dans la nouvelle population ou
   (b) insérer l'un des deux individus pères dans la nouvelle population.
   f. **Pour** i := 1 à *tp* **Faire** muter l'individu S_i de la population avec un probabilité P_m.
   g. Evaluer les individus de la nouvelle population.
   h. Mettre à jour PO* : PO* = *Solutions Non Dominé* (PO*+ Population)
   Fin

IV.4.5Performances des différentes stratégies de sélection

Pour comparer les performances des 6 stratégies de sélection implémentées Des tests ont été effectués sur le problème de *Heller* à 20 jobs, 10 machines[Hel 60]. Les tests font état d'une amélioration considérable de la recherche avec l'introduction de l'élitisme dans la phase de

sélection. Les stratégies Non-Pareto représentées ici par la sélection par somme pondérée et la sélection parallèle semblent être non adaptées au cas Multicritère. Les trois stratégies de sélection *NSGA*, *NDS*, *WAR* sont de performances presque identiques, avec une légère suprématie pour les méthodes *NSGA* et *NDS*. Les paramètres des différentes méthodes sont décrits dans le tableau1, la taille de la population est de 200 individus et le nombre de générations limite est de 15000. Les paramètres P_c et P_m sont respectivement égalaux à 0.8 et 0.7.

Somme pondérée	λ1= 0.5	λ2= 0.5
NSGA	S = 1.7	
NDS	S = 1.7	
WAR	S = 1.7	
Elitiste	S = 1.7	A = 5

Tableau 1: Paramètres des différentes stratégies de sélection

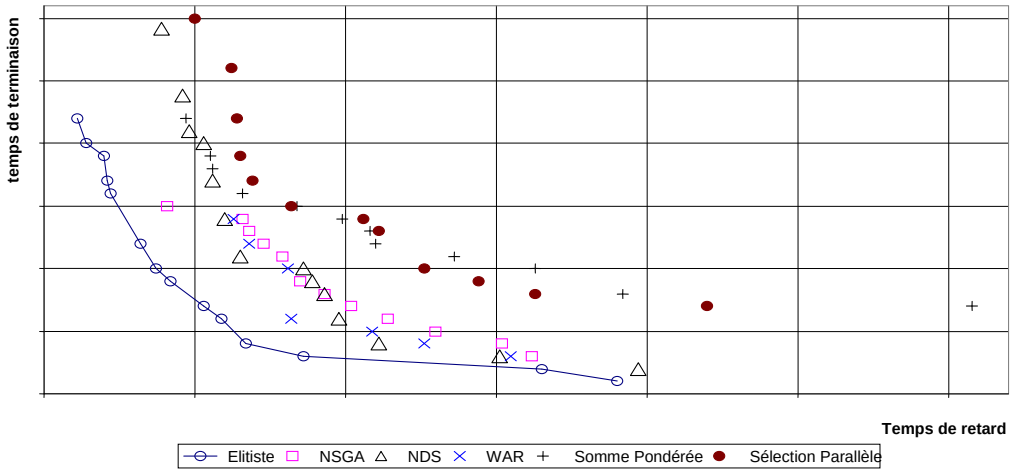


Figure IV.8 : Comparaison des différentes stratégies de sélection

IV.4.6Effet du paramètre "A" sur la stratégie de sélection élitiste

L'apport de l'élitisme étant prouvé, il est donc intéressant de connaître l'impact qu'a le choix du paramètre *A* sur l'évolution de la recherche dans l'algorithme génétique. Le choix de la valeur de ce paramètre influe considérablement sur la balance exploitation/exploration. En effet une valeur élevée de *A* a pour effet d'intensifier l'exploitation des bonnes solutions trouvées par la recherche. Une valeur petite de *A* a pour effet de favoriser l'exploration de nouveaux horizons de l'espace de recherche.

Le choix d'une valeur adaptée de *A* est assez déterminant pour l'efficacité de la recherche. Les figures IV.9, IV.10 montrent l'évolution de la recherche le long des générations 1000, 3000, … avec des valeurs du paramètre *A* = 1 et 4. Pour une valeur de *A* petite(*A*=1) la convergence vers la frontière Pareto se fait lentement contrairement aux résultats obtenus avec *A* = 4.

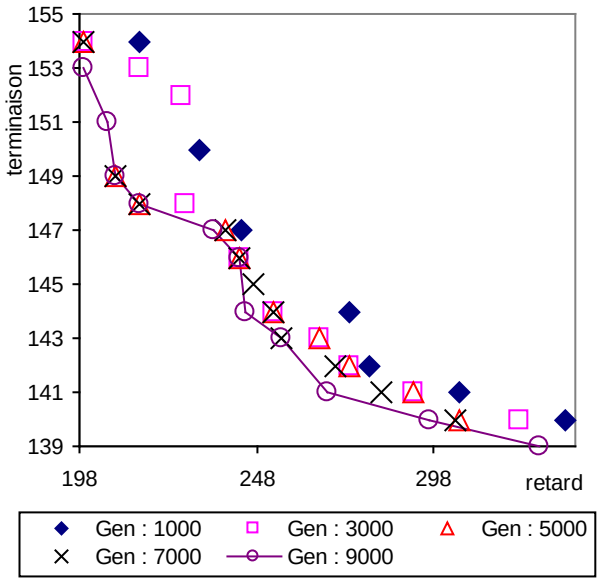


Figure IV.9 : Evolution de la recherche pour A = 1

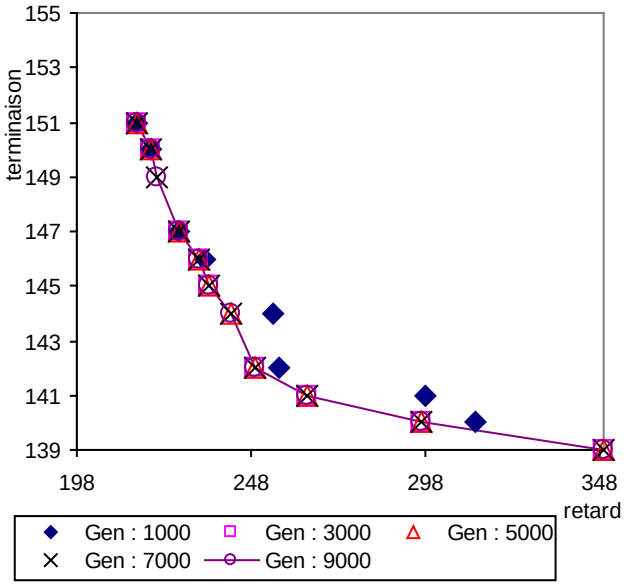


Figure IV.10 : Evolution de la recherche pour A = 4

IV.5 Le maintien de la diversité

Les AGs classiques sont réputés pour être très sensibles quant au choix de la population initiale ainsi qu'aux mauvais échantillonnages lors de la sélection. Cette fragilité est observable sur le plan de la perte de diversité ou ce qu'on appelle aussi la dérive génétique. Pour palier à cet inconvénient plusieurs approches visant à maintenir la diversité dans la population ont été proposées : *Introduction de nouveau individus, Maintien de la distance, Crowding, Restriction de Voisinage, Niches écologiques.*

Le principe du *Sharing*[Gol 87] est la dégradation de l'adéquation des individus appartenant à des espaces de recherche de forte concentration de solutions. Ce procédé à pour effet de :

- Changer le paysage du problème.
- Favoriser la dispersion des solutions de la population dans l'espace de recherche.
- La formation de sous populations d'individus semblables.

La dégradation de l'adéquation d'un individu est réalisée grâce à une fonction appelée fonction de partage (*sh*). La fonction de coût révisée d'un individu S_i notée $f'(S_i)$ est égale à la fonction de coût originale f multipliée par le compteur de niche de l'individu $m(S_i)$.

$$F'(Si) = F(S_i) \times m(S_i) \qquad \text{(IV.14)}$$

Le compteur de niche calcule le degré de similarité qu'a un individu avec le reste de la population. La fonction de partage *sh* est définie comme suit :

$$m(x) = \sum_{y \in Population} sh(dist(x,y)) \qquad \text{(IV.15)}$$

$$sh(dist(x,y)) = \begin{cases} 1 - \left(\frac{dist(x,y)}{\gamma}\right)^{\alpha} Si \quad dist(x,y) < \gamma \\ 0 \text{ sinon} \end{cases} \qquad \text{(IV.16)}$$

Dans la formule (IV.16) la constante γ désigne le seuil de dissimilarité (taille des niches). C'est à dire la distance au bout de laquelle deux individus S_i et S_j ne sont plus considérés comme appartenant à la même niche. La constante α par contre permet de contrôler et réguler la forme de la fonction *sh*.

Après l'évaluation des individus, Le vecteur objectif *F* de chaque individu est dégradé suivant le procédé de la formule (IV.14). Une fois les vecteurs *F*' sont calculés, les rangs des individus sont assignés et les probabilités déduites.

Selon que la distance entre les individus est calculée dans l'espace de décision (la représentation chromosomique de l'individu) ou dans l'espace objectif (adéquation de l'individu) trois possibilités peuvent se présenter.

IV.5.1 Sharing Génotypique

Dans ce cas la distance entre individus est calculé sur la base de la différence entre chromosomes. Etant donné qu'un ordonnancement est représenté par une permutation, la distance entre deux ordonnancements, donc deux permutations, est prise comme étant égale à :

$$d1(x, y) = |\{(i, j) \in J \times J / i \text{ précède } j \text{ dans la solution } x \text{ et } j \text{ précède } i \text{ dans } y\}| \qquad \text{(IV.17)}$$

Ce qui signifie que la distance entre deux individus *x* et *y* est égale au nombre de ruptures d'ordre existantes entre *x* et *y*.

IV.5.2Sharing Phénotypique

La distance dans ce cas est prise comme étant la différence entre les coûts des deux individus.

f1 et f2 désignent les deux fonctions objectives temps de terminaison final et le temps de retard total.

$$d2(x,y) = |f_1(x) - f_1(x)| + |f_2(x) - f_2(x)| \qquad \text{(IV.18)}$$

IV.5.3Sharing combiné

Ce dernier cas représente la combinaison des deux premières approches dans le sens où le calcul de la distance fait intervenir les deux sortes de distances, génotypique et phénotypique. La fonction *sh* prend dans ce cas la forme suivante :

$$sh(x, y) = \begin{cases} 1 - \dfrac{d1(x, y)}{\gamma 1} & \text{si d1(x, y)} < \gamma 1,\ \text{d2(x, y)} \geq \gamma 2 \\ 1 - \dfrac{d2(x, y)}{\gamma 2} & \text{si d1(x, y)} \geq \gamma 1,\ \text{d2(x, y)} < \gamma 2 \\ 1 - \dfrac{d1((x, y)d2(x, y)}{\gamma 1 \gamma 2} & \text{si d1(x, y)} < \gamma 1,\ \text{d2(x, y)} < \gamma 2 \\ 0 & \text{sinon} \end{cases} \qquad \text{(IV.19)}$$

L'étape du *sharing* suit directement l'étape d'évaluation et précède celle de la sélection. Une fois les vecteurs coût des individus de la population sont calculés, La procédure de dégradation du coût décrite dans la formule VI.14 est exécutée. L'algorithme suivant reprend la procédure de recherche génétique tout en détaillant l'étape de *sharing*.

```
Générer la population initiale
Evaluer les individus ind de la population (calculer les F(ind)
Générer la population Pareto PO*
Tant que le nombre de génération limite n'est pas atteint
Faire Début
    Pour chaque ind appartenant à la population
    Faire Début
        Calculer le compteur de niche m(ind)
        F'(ind) = F(ind)*m(ind)
        Calculer les rangs des  individus de la population sur la base des vecteurs F' et de la stratégie de sélection choisie
        Calculer les probabilités de sélection des individus
        Sélection
        Reproduction
        Evaluation
        Mise à jour de l'ensemble PO*
Fin
```

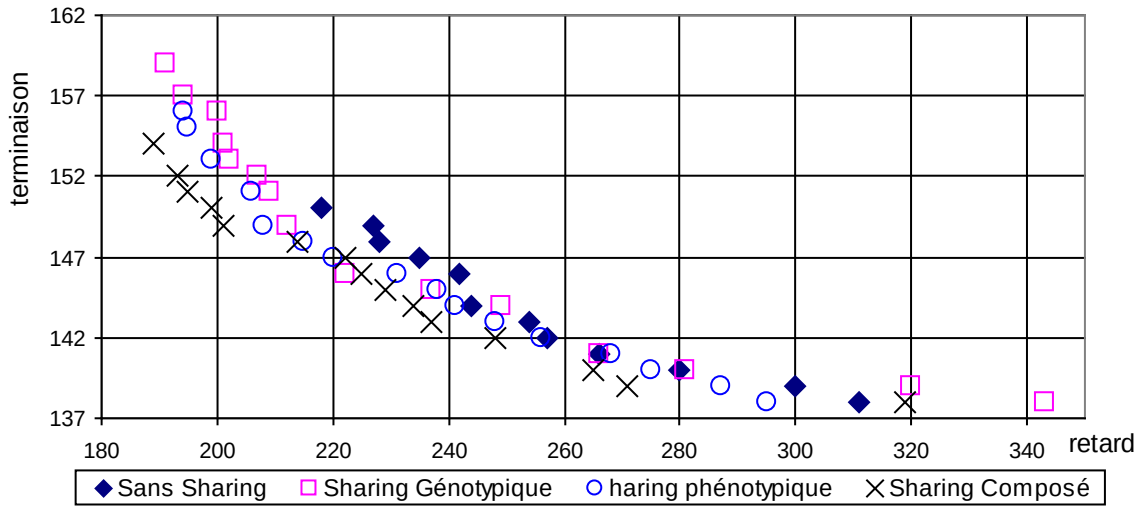


Figure IV.11 : Apport de la diversification

La figure IV.11 représente l'apport des différentes variantes de diversification sur la recherche. Les tests ont été effectués sur le problème de *Heller* 20*10. Avec un nombre de générations limite de 50000 et en utilisant les mêmes paramètres que ceux décrit précédemment pour la stratégie de sélection élitiste. Les paramètres concernant la stratégie de diversification sont : α=0.9, γ1=4 et γ2=1,0. L'apport de la diversification génotypique est peut appréciable comparé aux résultats obtenus par la diversification phénotypique. Cependant, la diversification dans l'espace de décision se distingue par un ensemble de solutions efficaces non trouvées par la diversification phénotypique. La diversification composée présente une meilleure qualité de solutions que les deux précédentes méthodes. A noter, que l'apport de la diversification n'apparaît qu'après un nombre de générations considérable d'où la nécessité de prendre un nombre de génération limite très élevé (> 40000).

IV.6 Hybridation avec la recherche locale

Notre intérêt s'est porté sur l'utilisation de la recherche locale (RL) comme moyen d'accélération et de raffinement de la recherche Génétique. L'idée dans ce cas est de lancer l'AG en premier lieu afin d'approcher la frontière Pareto, après quoi la recherche locale s'occupera du raffinement des solutions trouvées par l'AG afin de mieux approcher l'ensemble des solutions Pareto. Le principe de l'hybridation est simple, une fois l'AG terminé (le nombre de générations limite atteint), la recherche locale est alors lancée ayant pour entrée l'ensemble Pareto obtenu(voire figure IV.12).

L'utilisation de la recherche locale nécessite premièrement le choix de la manière par lequel le voisinage d'une solution donnée est généré. Nous, nous sommes inspirés pour cela de l'opérateur de mutation. Le voisinage d'une solution correspond à l'ensemble des permutations générées suite à la rotation dans les deux sens d'une partie de la solution(voir la figure IV.13).

Pour générer l'ensemble des voisins d'une permutation *O*, on choisit chaque fois une paire de jobs de position *i* et *j* respectivement. L'insertion du job présent à la position *i* à la position *j* donne un premier voisin O_1. De même, l'insertion du job présent à la position *j* à la position *i* donne un deuxième voisin O_2. De ce fait, le nombre de voisins d'un individu *O* est égal à : $2 \times C_N^2 = N(N-1)$.

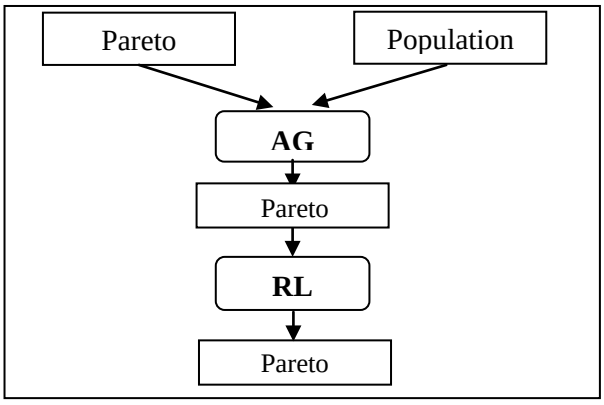


Figure IV.12 : Hybridation de l'AG avec la RL

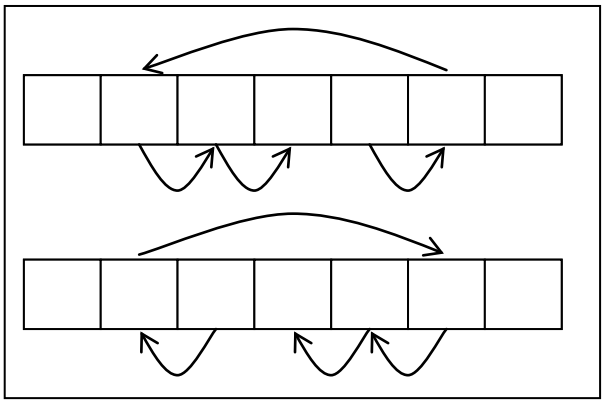


Figure IV.13 : Génération des voisins dans la RL

Le procédé d'hybridation consiste à générer pour chaque individu de la population Pareto finale, l'ensemble de son voisinage. Les voisins non dominés dans la population Pareto sont insérés dans celle-ci. Les solutions appartenant à Pareto et dominée par le voisinage d'une solution sont supprimées. Ce procédé est réitérer jusqu'à ce qu'aucun voisin d'aucune solution Pareto ne soit inséré dans la population Pareto. La figure IV.14 reprend sous forme schématique le procédé de la recherche locale.

Les mesures de performance de l'AG hybride montre que la recherche locale ne présente aucun intérêt pour les problèmes de petites tailles notamment *Heller* 20*10. Cependant l'apport de l'hybridation se fait sentir dès que la taille du problème augmente. Les tests représentés dans la figure IV.15 sont réalisés sur le problème de *Heller* avec 100 jobs et 10 machines.

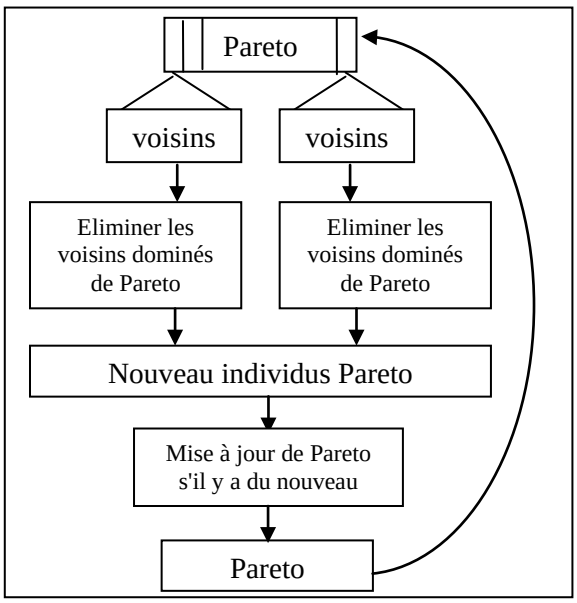


Figure IV.14 : Recherche locale en action

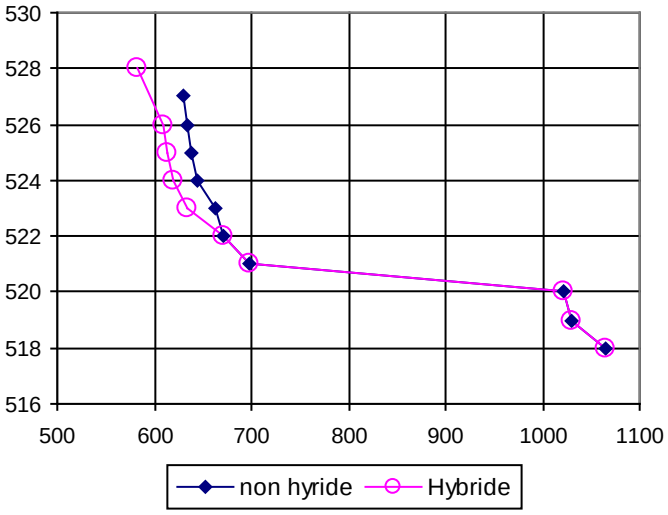


Figure IV.15 : Apport de la recherche locale

La procédure de recherche locale est décrite ci-dessous

```
Nouveau := vrais
Tant que Nouveau = vrais
Faire Début
    Nouveau = Faut
    Pour tout individu ind de la Population Pareto PO*
    Faire Début
        Voisins = ∅
        Pour i = 1 à N-1
        Faire  Pour j = i à N
                Faire Début
                    Ind1 = ind
                    Insérer le job de la position j à la position i dans ind1
                    Voisins = Voisins ∪ {ind1}
                    Ind2 = ind
                    Insérer le job de la position i à la position j dans ind2
                    Voisins = Voisins ∪ {ind2}
                Fin
        Mettre à jour l'ensemble PO* à partir de PO* et Voisins
        Si PO* est modifié alors Nouveau = vrais
    Fin
Fin
```

Dans [Mur 98], la procédure de la recherche locale est lancée suite à chaque étape de reproduction. Les individus de la nouvelle population sont considérés comme solutions de départ pour la recherche ascendante(Hill Climbing). Chaque individu est amélioré suivant le procédé de transition d'une solution à une autre solution avoisinante meilleure. Le processus est arrêté quand aucune amélioration n'est possible. Afin de réduire le temps de calcul induit par l'utilisation de la recherche locale suite à chaque génération, *Murata* et *Ischibuchi* fixe le nombre de voisins d'une solution examinés lors de chaque itération de la recherche locale. La figure IV.16 suivante décrit le mécanisme d'hybridation suivi par les deux auteurs.

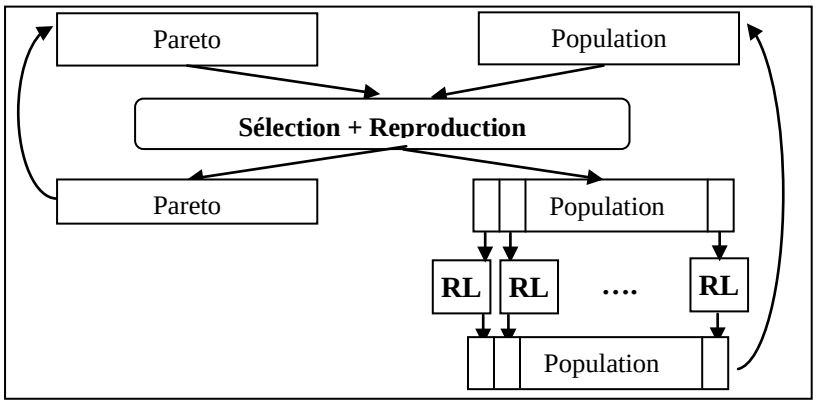


Figure IV.16 : Hybridation itérative l'AG avec la RL

Autre le fait que cette approche est très sensible quant au choix du paramètre *k* du nombre de voisins à explorer lors de chaque itération, l'algorithme est considérablement ralenti. Les tests comparatifs entre les deux approches d'hybridation montrent une égalité dans la qualité des solutions trouvées se qui ne justifie en rien le temps d'exécution plus long dû à la deuxième approche.

IV.6.1Tests et comparaisons avec d'autres travaux

L'étude comparative des travaux d'optimisation multicritère se heurte généralement au problème d'absence de benchmarks. Ce qui est le cas notamment du problème du Flow shop. Ceci est dû aux différences entre les critères à optimiser et les spécificités du problème traité. Seul le temps de terminaison final peut être considéré comme un critère de comparaison effective. Nous avons donc opté pour l'étude de l'habilité de l'algorithme à trouver des solutions extremums (se trouvant aux extrémités de la frontière Pareto) avoisinants celles trouvées par des algorithmes monocritère.

Les problèmes (ta*xx*) dus à *Taillard* [Tai 93], offre un bon support de comparaison. D'une part, ces problèmes sont très référencés dans la littérature[Ben 98][Now 99]. D'autre part, chaque problème est caractérisé par une valeur *LB* correspondant à la borne inférieure du *makespan* et d'une valeur *UB* correspondant à la meilleure solution obtenue par *Taillard* pour ce problème.
L'algorithme proposé par *Taillard* consiste en la génération d'une instance de problème à partir des trois données d'entrées : *N*, *M* et un nombre entier *SEED*. Nous étions donc contraints de raffiner ce procédé afin de générer des instances comportant des temps de retard. L'algorithme suivant décrit notre générateur d'instances de flow shop bi-critère, où *N*, *M*, *SEED*, *UB* sont des variables globales données par l'utilisateur. Cet algorithme génère les mêmes instances que celles de *Taillard* munies en plus des temps limite l_i.

```
fonction unif1: entier
début
    m = 2147483647;   a = 16807;   b = 127773;   c = 2836;
    k = SEED / b;                                   { division entière }
    SEED = a * (SEED modulo b) - k * c;
    si (SEED < 0) SEED = SEED + m;
    valeur_0_1 = SEED / m ;                         { division non entière }
    retourner inf + (valeur_0_1 * (sup - inf + 1));
fin
fonction unif2 : entier
début
    m = 2147483647;   a = 16807;   b = 127773;   c = 2836;
    k = SEED2 / b;                                  { division entière }
    SEED2 = a * (SEED2 modulo b) - k * c;
```

```
    si ( SEED2 < 0)  SEED2 = SEED2 + m;
    valeur_0_1 = SEED2 / m ;                                  { division non entière }
    retourner 12.0 * UB / 30.0 + valeur_0_1 * 17.0 * UB / 30.0);
fin
procédure Générer_instance
début
   SEED2 = SEED * 2;
   pour j allant de 1 à M
      pour i allant de 1 à N faire d[i][j] = unif1;           { d[i][j] désigne la durée de la tâche t_ij }
   pour i allant de 1 à N faire limit[i]=unif2;               { limit[i] désigne la date limite l_i du job J_i }
fin
```

Le tableau 2 présente les résultats de tests effectués sur 8 instances de *Taillard étendu* au cas bi-critère notées ma_ta*xx*_bi. *MM* désigne le meilleur *makespan* obtenu, *MR* le temps minimal de retard enregistré. Le paramètre *dév* mesure l'écart entre le meilleur *makespan* obtenu et *UB*. |*PO*| calcul le nombre d'individus du front Pareto obtenus.

Problème	**Dimension (N x M)**	**SEED**	**LB**	**UB**	**MM**	**Dév**	**MR**	**|PO|**	**Nb de génération**
Ma_ta01_bi	20x5	873654221	1232	**1278**	**1278**	0 %	453	4	50 000
Ma_ta02_bi	20x5	379008056	1290	**1359**	**1359**	0 %	491	6	50 000
Ma_ta11_bi	20x10	587595453	1448	**1582**	**1586**	0.25%	1508	28	80 000
Ma_ta12_bi	20x10	1401007982	1479	**1659**	**1674**	0.9%	1342	21	80 000
Ma_ta21_bi	20x20	479340445	1911	**2297**	**2330**	1.43%	1062	32	200 000
Ma_ta31_bi	50x5	1328042058	2712	**2724**	**2735**	0.4%	3629	11	200 000
Ma_ta41_bi	50x10	1958948863	2907	**3037**	**3126**	2.93%	6653	24	200 000
Ma_ta51_bi	50x20	1539989115	3480	**3886**	**3990**	2.67%	11379	32	300 000

Tableau 2: performance de l'algorithme génétique hybride

Les résultats affichés par le tableau 2 démontrent la capacité de l'algorithme à trouver des solutions de faible compromis donc extremums. Ceci dit, l'ensemble *Pareto* trouvé est assez dispersé ce qui nous offrira un bon échantillonnage du front *Pareto*. Les déviations toutes en étant petites restent aussi justifiables. En effet l'algorithme proposé reste très indépendant du problème traité puisque aucune heuristique garantissant des solutions de faible *makespan* ou minimisant le temps de retard n'est utilisée. La petite taille de l'ensemble |PO| est due au caractère fortement corrélé des deux objectifs à optimiser.

L'algorithme ainsi proposé n'étant pas guidé par d'autres heuristiques adaptées au problème de flow shop, reste donc très lent. La parallélisation de l'algorithme s'impose donc comme moyen judicieux pour combler cette lacune. La réduction de temps d'exécution d'une génération rendra moins coûteux l'augmentation de la taille de la population ou le nombre de générations limite.

IV.7 Algorithmes Génétiques Parallèles

Deux modèles de parallèlisation ont été élaborés l'un se basant sur un modèle centralisé, l'autre sur un modèle distribué. Le modèle centralisé se base sur une architecture maître esclave. Le processus maître s'occupe du lancement des processus esclave et de la distribution du travail. Le modèle distribué s'appuie sur une distribution équitable du travail sur les différents processus.

Le modèle centralisé s'appuie sur la parallèlisation des étapes de sélection, d'évaluation et de reproduction. Le processus maître contrôle la distribution des données d'entrée et la récupération des résultat. A chaque itération, la population est subdivisée en plusieurs sous-populations. Un processus esclave s'occupe de l'évaluation des individus de chaque sous-

population. Le processus central récupère les vecteurs coût des individus et les distribue sur un ensemble de processus esclaves qui réalise la sélection. La sélection étant réalisé par chaque processus de façon partielle sur un sous ensemble de la population ceci engendre une plus faible pression de sélection. Finalement, les individus sélectionné par chaque processus sont retransmit au processus maître et sont redistribuer sur des processus esclaves qui effectuent la reproduction. Ce procédé est repris de façon shématique dans la figure ci-dessous.

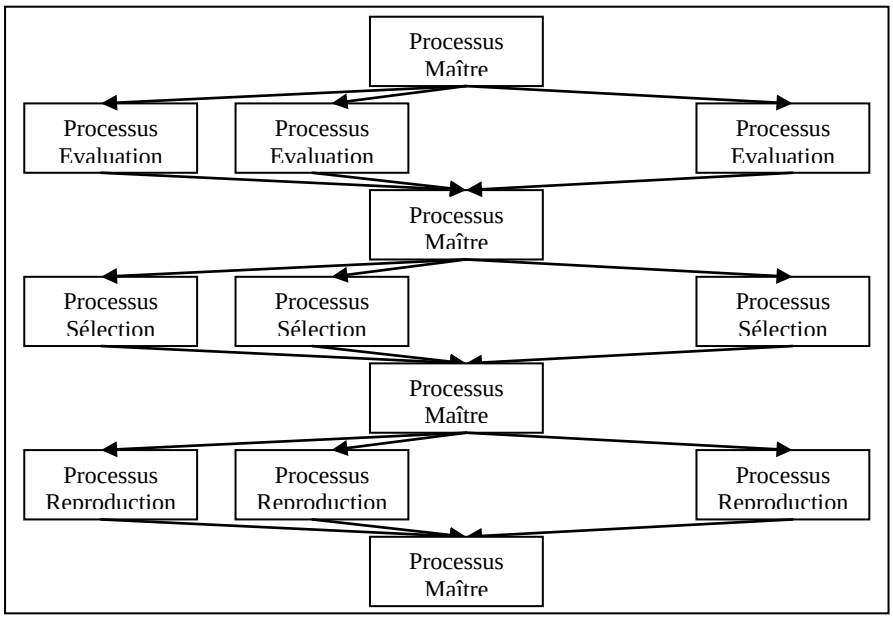


Figure IV.17 : Modèle centralisé

Ce modèle est caractérisé par un taux de communication élevé ainsi qu'un degré de synchronisation considérable. En effet, le lancement de la phase de sélection n'est possible qu'une fois le processus maître ait récupéré la totalité des vecteurs coûts des processus esclaves. De ce faite la vitesse d'execution est conditionnée par la vitesse du processus s'exécutant sur la machine la mois rapide ou la plus surchargée. Un inconvénient de ce modèle réside dans la sensibilité face au cas de pannes.

Les tests effectués sur la base de ce modèle ne fournissent aucune amélioration de la qualité des solutions trouvées. Le temps d'exécution n'est pas amélioré de façon remarquable du faite que le processus de *sharing* et de *mise à jour de l'ensemble Pareto* n'est pas parallèlisé en plus des volume de message transmit chaque fois(sous ensemble d'individus, sous ensemble de vecteurs coût).

Le modèle distribué[Tal 95] prône la subdivision de la population en sous-populations de tailles égales. Chaque processeur exécute l'AG sur la sous population qui lui est affiliée. A des périodes de temps déterminées les différents AGs procèdent à un échange de leurs meilleurs individus présents dans leurs sous-populations respectives. Nous avons opté pour une topologie de communication en anneau (voir figure IV.18). Ceci afin de minimiser au maximum le taux de communications inter-processus tout en maintenant l'aspect fortement connexe du graphe. Ce qui garantie qu'un bon individu pourra se propager à toutes les sous-populations après un certain nombre de générations. Nous décrivons ci-dessous sous forme algorithmique et schématique le procédé de migration.

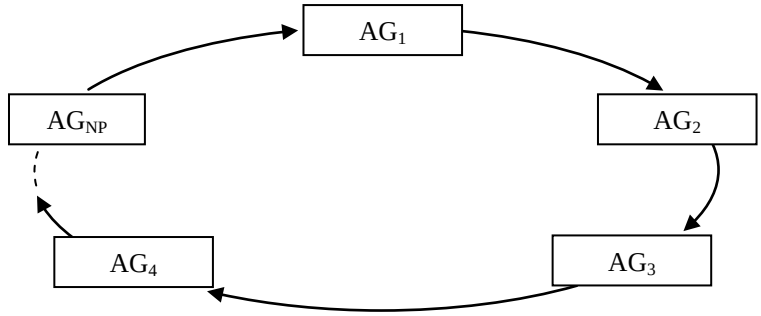


Figure IV.18 : Topologie de communication en anneau

1. Générer la sous-population initiale P_0, i = 0,
2. **Tant que** le nombre de génération limite n'est pas atteint
   Faire - Sélection
   - Reproduction
   - **Si** (i **modulo** Période) = 0 **alors**
   - **si** l'individu émit précédemment est reçu **alors** choisir un individu non dominé et l'envoyé au processus droit
   - **si** c'est disponible **alors** recevoir un individu du processus gauche
   - accuser réception au processus gauche.
   - insérer l'individus reçu dans la population en choisissant une victime de façon aléatoire
   - i = i + 1
   Fait

Les avantages d'un tel modèle sont nombreux:

- *Fiabilité & stabilité*: Le blocage d'un processus ou sa lenteur n'influe pas sur le comportement du reste des processus.
- *Taux de communication réduit*: lors de chaque vague de migration en aura au maximum 2×*NP messages échangés*. *NP* individus émis et *NP* accusés de réception.
- *Indépendance vis à vis de l'environnement* : l'algorithme est indépendant de la plate-forme d'exécutions (homogénéité, nombre de machines, caractéristiques).
- *Un degré de synchronisation faible* : l'immigration des individus entre les sous-populations n'exige aucune synchronisation entre processus. En effet, un processus envoi son meilleur individu à son voisin même si celui-ci n'est pas prêt à le recevoir. Cependant, il ne renvoi d'autres individus que si le précédent ai été consommé (insérer dans la population voisine). D'autre part, aucun processus ne se bloque en attente d'un individu migrateur.

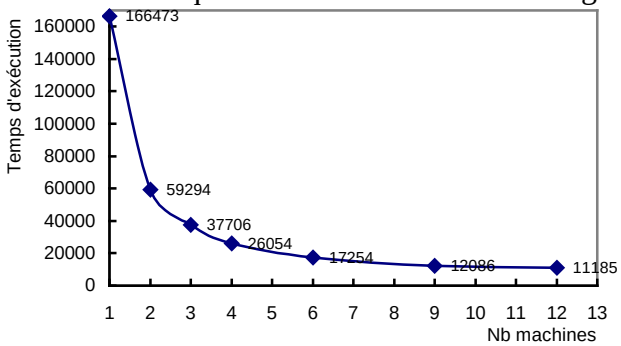


Figure IV.19 : Variation de la vitesse d'exécution de l'AG parallèle pour le problème ma_ta21_bi

La figure IV.19 montre la variation des temps d'exécution (en secondes) de l'AG parallèle quand le nombre de machines donc de processus augmente. Les tests ayant étés effectués sur un cluster de Stations *SUN ULTRA* 1. Le gain en temps d'exécution étant prouvé, ceci pourra

nous inciter à augmenter les tailles des populations ainsi que le nombre de générations limite dans le but de trouver des solutions encore meilleures.

Le calcul du rang ou de la fonction de partage d'un individu ne tient compte des individus appartenant à la même sous-population. Ce qui explique la pente super linéaire de la figure IV.19.

		AG Séquentiel						AG Parallèle					
Problème	**UB**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**
ma_ta11_bi	1582	1586	0.25%	1508	28	200	80000	1583	0.06%	1431	32	300	300000
ma_ta21_bi	2297	2330	1,43%	1062	32	200	200000	2305	0.34%	1057	29	300	300000

Tableau 3 : Amélioration de la qualité des solutions pour le problème ma_ta11_bi et ma_ta21_bi

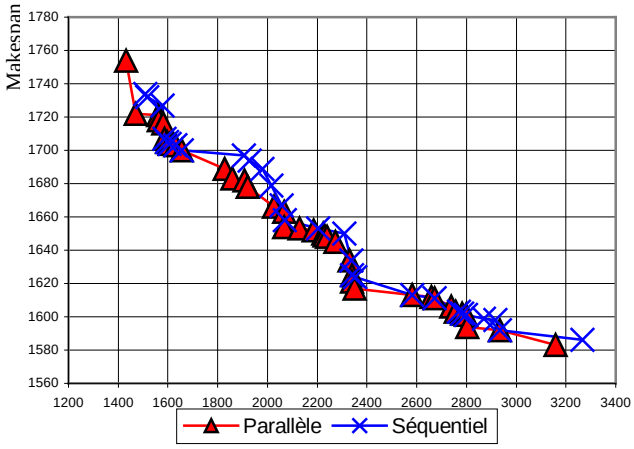


Figure IV.20 : Amélioration du front Pareto par l'introduction du parallélisme pour le problème ta11_bi

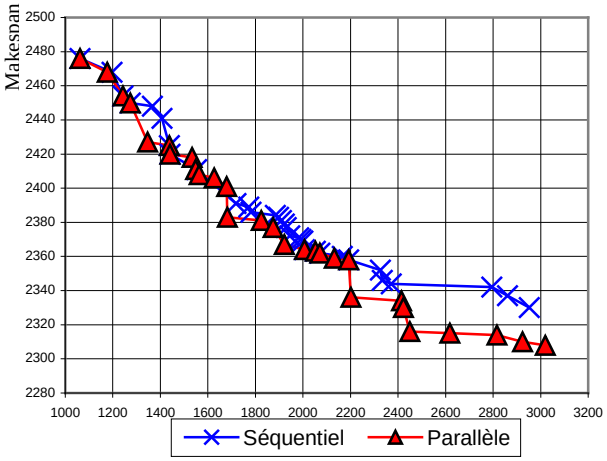


Figure IV.21 : Amélioration du front Pareto par l'introduction du parallélisme pour le problème ma_ta21_bi

Comme le montrent les figures IV.20 et IV.21, l'utilisation de populations de grandes tailles réparties sur différents AGs et l'augmentation du nombre de générations limite améliore la qualité des solutions trouvées.

L'algorithme est réalisé sous l'environnement PVM (Parallel Virtual Machine). Cette environnement permet à des machines de nature différentes et fonctionnant sous des architectures système distinctes de travailler exactement comme s'ils faisaient partie d'une même machine à plusieurs processeurs. Aucun équilibrage de charge n'est effectué dans cette environnement, de ce fait la vitesse d'exécution de l'algorithme dépendras de la vitesse de la plus faible machine ainsi que de la charge externe qui circule dans le système.

IV.8 Heuristique *NEH* pour la génération de la population initiale

Nous proposons dans cette section une variante probabiliste de l'heuristique *NEH* due à *Nawaz*, *Escore* et *Ham*, conduisant à la génération d'une population d'individus de bonne qualité. Dans cette variante, une solution est générée par élaboration progressive de plusieurs séquencements partiels. A chaque itération un job J_i est sélectionné pour être inséré dans la séquence partielle δ avec une probabilité égale à :

$$\left(\sum_{j=1}^{M} d_{ij}\right)^{\alpha} \Big/ \sum_{k\in\Phi}\left(\sum_{j=1}^{M} d_{kj}\right)^{\alpha} \qquad \text{(IV.20)}$$

où Φ correspond à l'ensemble des jobs non encore planifiés. Une fois un job est sélectionné, il est inséré dans la séquence δ à la position engendrant le meilleur makespan où le meilleur temps de retard. Le nombre α qui figure dans la formule ci-dessus présente un paramètre de contrôle de l'heuristique dont la valeur est comprise entre $]0, +\infty[$. Une valeur petite de α conduit à des solutions de modeste qualité bien répartie dans l'espace. Une valeur élevé du paramètre génère des solutions de bonnes qualité concentrées dans des régions attractives de l'espace de recherche. Nous donnant ci-dessous la procédure de génération de la population initiale à base de l'heuristique NEH.

pour chaque individus *ind* de la population initiale
faire début
  Φ = {l'ensemble de tout les jobs}
  δ = **nil**
  pour chaque job J_i calculer les temps de traitements D[i]= somme(D[k])
  choisir un job J_m avec la probabilité $D[m]^{\alpha}$/somme($D[k]^{\alpha}$)
  D[m]= 0 $\Phi = \Phi - J_m$
  choisir un job J_n avec la probabilité $D[n]^{\alpha}$/ somme($D[k]^{\alpha}$)
  D[n]= 0 $\Phi = \Phi - J_n$
  Si $J_m \rightarrow J_n$ est meilleure que $J_n \rightarrow J_m$ alors $\delta = J_m \rightarrow J_n$ sinon $\delta = J_n \rightarrow J_m$

  Pour k allant de 3 à N
  Faire début
    choisir un job J_m avec la probabilité $D[m]^{\alpha}$/somme($D[k]^{\alpha}$)
    D[m]= 0 $\Phi = \Phi - J_m$
    Générer l'ensemble des séquences construites par introduction de J_m dans δ à l'une des k positions possibles
    Si *ind* < *tp*/2 **alors** δ = la séquence avec le meilleur makespan
    Sinon δ = la séquence avec le meilleur temps retard
  fait
fait

La moitié des individus de la population initiale sont générés sur la base de la minimisation du *makespan*, l'autre moitié se base sur la minimisation du temps de retard. Les tests repris dans le tableau ci-dessous représentent une comparaison entre l'algorithme séquentiel décrit dans le paragraphe VI.6. Le paramètre α est pris égal à 1.

		AG Séquentiel avec génération aléatoire de la population initiale						**AG Séquentiel avec génération de la population initiale basé sur l'heuristique NEH**					
Problème	**UB**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**
Ma_ta11_bi	1582	1586	0.25%	1508	28	200	80000	1586	0.25%	1392	37	200	80000
Ma_ta12_bi	1659	1674	0.9%	1342	21	200	80000	1672	0.78%	1342	21	200	80000
Ma_ta21_bi	2297	2330	1.43%	1062	32	200	200000	2303	0.26%	1097	30	200	200000

Tableau 4 : Comparaison des méthodes de génération aléatoire et NEH

Pour ces deux problèmes, l'utilisation de l'heuristique NEH comme outil de génération de la population initiale fournit de meilleurs résultats. L'analyse des frontières Pareto produit par chacune des deux approches(figure IV.22, IV.23), fait apparaître un plus large front dans le cas d'une génération NEH de la population initiale.

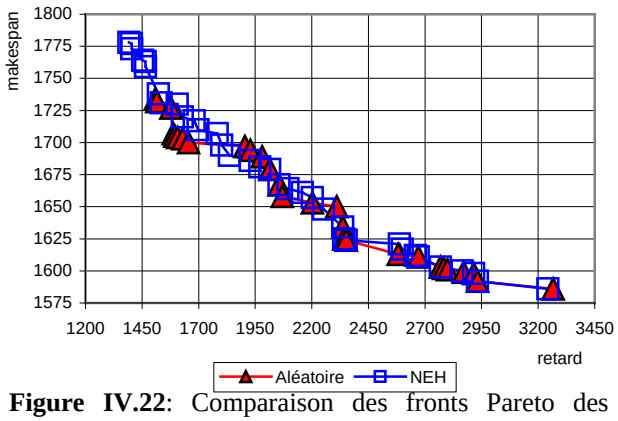


Figure IV.22: Comparaison des fronts Pareto des approches de génération de la population initiale aléatoire et NEH pour le problème ma_ta11_bi

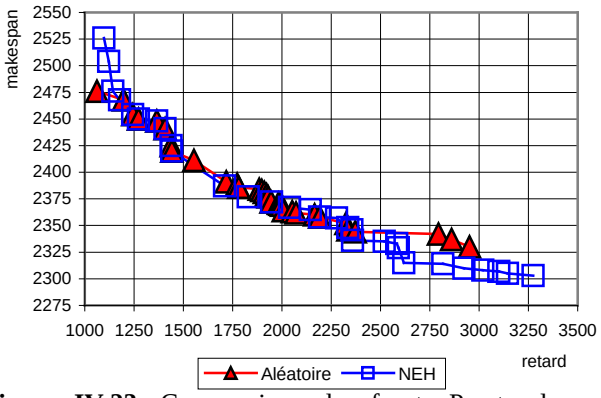


Figure IV.23: Comparaison des fronts Pareto des approches de génération de la population initiale aléatoire et NEH pour le problème ma_ta21_bi

IV.9 Conclusion

Nous avons tenté dans cet partie de construire notre approche par introduction progressive de concepts tels que la sélection, la maintenance de la diversité l'hybridation et la parallèlisation. A chaque étape nous avons illustré l'apport du mécanisme introduit, ce qui nous permet de formuler les conclusions suivantes.

Les stratégies de sélection Pareto (NSGA, NDS, WAR) semblent être mieux adaptées au cas multicritère. L'efficacité de telles méthodes est améliorée avec l'introduction de l'élitisme lors de la phase de sélection. N'empêche que l'élitisme peut conduire à la convergence prématurée de recherche d'où la nécessité de bien choisir les paramètres A et S. Cependant, le risque de la dérive génétique et de l'instabilité de la recherche reste présent. Les stratégies de diversification semblent être le moyen privilégié pour prévenir de tels problèmes. Trois variantes de la méthode de sharing ont été développées, basées sur les niches écologiques. Le Sharing Phénotypique (diversification dans l'espace objectif) apparaît le plus intéressant vu l'intérêt que porte le décideur à l'obtention d'une frontière Pareto plus large est mieux dispersée. Ce qui n'empêche pas la diversification génotypique de fournir de bons résultats. La combinaison des deux concepts améliore considérablement la recherche.

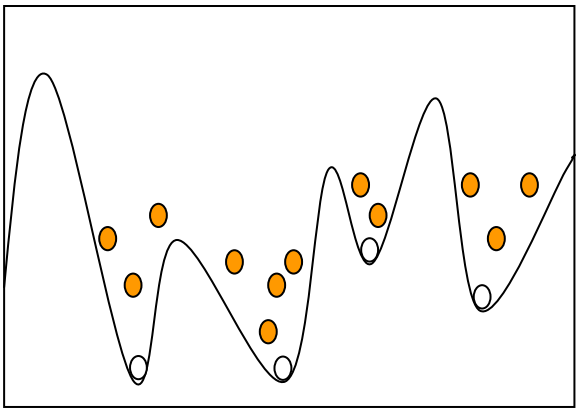


Figure IV.24 : Après l'AG Après la RL

La recherche locale a été utilisée comme moyen de raffinement et d'accélération de la recherche(Figure IV.24). L'idée et de lancer l'AG en premier afin d'avoir une première approximation de la frontière Pareto. La recherche locale se charge alors d'améliorer ces solutions (trouver les optimums locaux des régions en la quelles se trouve la recherche). Le

lancement itératif de la recherche locale suite à chaque génération de l'AG ne donne pas de meilleurs résultats et ne justifie donc pas le coût important d'exécution induit. L'apport de l'hybridation n'apparaît que pour des problèmes de grande taille.

Pour tester l'efficacité d'une méthode le concepteur est face à deux problèmes. D'une part, le problème de Flow Shop multicritère n'est pas standard (les critères à optimiser ne sont pas identiques). D'autre part, l'absence de benchmarks dans ce cadre. Nous avons proposé dans ce travail, l'extension des problèmes de Taillard[Tai 93] au cas bi-critère. Les tests effectués montrent la grand habilité de l'AG hybride à atteindre des solutions de faible makespan.

L'AG ainsi conçu reste d'une très grande lenteur dû au mécanismes de sélection et de diversification. La parallèlisation se présente alors comme moyen intéressant pour surmonter cet inconvénient. La réduction du temps d'une génération, pousse à l'utilisation de populations de taille plus grande et à l'augmentation du nombre de générations limite. Ce qui a pour effet d'améliorer la qualité du front Pareto trouvé.

Nous avons aussi illustré l'impact qu'à le choix de la population initiale sur la qualité du front Pareto trouvé. Nous avons proposé une nouvelle variante probabiliste de l'heuristique NEH assurant à la fois de bonnes solution de départ et de bonnes répartitions dans l'espace.

Conclusion Générale

L'objectif de cette thèse à été l'étude et l'amélioration des approches heuristiques proposées dans le cadre de l'optimisation multicritère. L'accent a particulièrement été mis sur l'utilisation des Algorithmes Génétiques pour la résolution des problèmes d'ordonnancement. Un choix justifié par la qualité et le volume des travaux dans ce sens qui font des AGs l'heuristique multicritère par excellence.

L'optimisation multicritère revêt plusieurs aspects. Le premier a trait à la manière dont le décideur coopère avec l'algorithme de résolution. Nous observons trois approches principales:

1. L'utilisateur formule ces préférences à priori. La recherche est alors lancée, son résultat est une solution unique répondant aux exigences de l'utilisateur.

2. La recherche est lancée en premier, conduisant à la production de plusieurs solutions. L'utilisateur choisit une solution parmi celles proposées.

3. Une coopération progressive est engagée entre le décideur et l'algorithme de recherche. L'algorithme est lancé sur la base des préférences du décideur. Les connaissances acquises lors de la résolution sont alors utilisées par le décideur afin de reformuler ses préférences.

Le deuxième aspect se rapporte à la signification donnée à la notion d'optimalité. Selon cette distinction, les approches de recherche multicritère peuvent être classées en quatre catégories : les approches opérant par transformation du problème en un problème mono-critères, les approches non-Pareto, les approches Pareto et les approches Min-Max. Parmi celles-ci, les approches Pareto semblent jouir d'une certaine unanimité au sein de la communauté d'optimisation.

Le problème de Flow Shop à permutation objet de notre travail est traité avec pour objectifs : la minimisation du temps de terminaison(*makespan*) et la somme des retards. Les tests effectués sur 5 stratégies de sélection implémentées montrent que les stratégies Pareto(NSGA, NDS, WAR) sont celles qui fournissent les meilleurs résultats. Ces stratégies partagent en commun le fait de s'appuyer sur la notion de dominance afin de dégager un rangement (*ranking*) des solutions présentes dans la population.

L'objectif d'une optimisation multicritère est de produire un ensemble de solutions efficaces, généralement Pareto optimales. Pour cette raison, l'algorithme développé stocke les meilleures solutions trouvées tout le long de la recherche. Dans les AGs ceci est réalisé en maintenant une population supplémentaire dite population Pareto ou archive. L'élitisme consiste à faire participer la population Pareto lors de la phase de sélection. Ce mécanisme sert généralement comme moyen d'intensification de la recherche. L'implémentation de l'élitisme nous a permit une réelle amélioration de la qualité des solutions trouvées

Dans le cadre de l'optimisation mono-critères le maintient de la diversité sert à faire face au risque de convergence prématurée et le désir de générer plusieurs optimums locaux en cas de fonction multi-modales. Dans l'optimisation multicritère, ces objectifs s'étendent à la volonté de bien échantillonner les solutions du front Pareto. L'utilisation de la notion des

niches écologiques vise à produire une bonne dispersion des solutions dans l'espace de recherche. Deux types de diversification peuvent être envisagés: la diversification dans l'espace de décision ou dans l'espace objectif. La pratique montre que le décideur voue un intérêt particulier à pouvoir disposer de solutions présentant des qualités de compromis différentes(dispersées dans l'espace objectif). L'implémentation des deux approches de diversification, montre une légère suprématie de la diversification phénotypique. La combinaison des deux approches fournit de meilleurs résultats.

Les AGs sont souvent critiqués pour leur lenteur, l'hybridation de la méthode avec celle de la recherche locale offre une solution à ce problème. L'algorithme Génétique est lancé en premier fournissant une première approximation de la frontière Pareto, la recherche locale est alors lancée ayant pour entrée l'ensemble Pareto fourni par l'AG. L'effet de l'hybridation n'est remarquable que pour des problèmes de grandes tailles.

Pour accélérer encore plus l'algorithme deux modèles d'AGs parallèles sont proposés, l'un se basant sur une approche centralisée l'autre sur une approche distribuée. Le modèle distribué offre un degré de synchronisation faible est un niveau de sûreté plus élevé. Dans ce cas, plusieurs AGs sont lancés chacun ayant pour entrée une sous-population d'individus initiale. A des périodes de temps déterminées une migration circulaire est effectuée entre les différents processus de recherche. La réduction du temps d'une génération, nous permet d'agrandir la taille de la population et d'augmenter le nombre de génération limite ce qui conduit à une recherche plus efficace.

Les AGs sont aussi connus pour leur sensibilité quant au choix de la population initiale. Comme il a été démontré que l'utilisation d'heuristiques pour la génération de cette population améliore les performances de l'AG, nous avons proposé une adaptation probabiliste de l'heuristique *NEH*[Ben 99] assurant de bonnes solutions de départ. La moitié des individus de cette population sont générés de façon à présenter de bons temps de terminaison(*makespan*), l'autre moitié est générée afin d'offrir de bon temps de retard.

L'absence de benchmarks pour les problèmes multicritère est souvent une difficulté auquel doit faire face le concepteur d'algorithme d'optimisation. Nous avons proposé dans cette thèse un générateur d'instance de problèmes de type flow shop bi-critères. Ce générateur se présente comme une adaptation de l'algorithme de *Taillard* souvent référencé dans la littérature. Nous estimons que ses instances fourniront une bonne base de comparaison pour les travaux futurs.

Une batterie de tests ont été effectués démontrant la grande habilité de notre algorithme à trouver des solutions de faibles compromis (solutions extrêmes). Pour l'algorithme génétique hybride séquentiel, la déviation du meilleur *makespan* trouvé par rapport à la solution exacte n'a jamais dépassé 2.93%(problème ma_ta41_bi) L'introduction du parallélisme fait passer le meilleur *makespan* trouvé de 1586 à 1583 pour le problème ma_ta11_bi et de 2330 à 2305 pour le problème ma_ta21_bi. La somme des retards passe de 1508 à 1431 pour le problème ma_ta11_bi et de 1062 à 1057 pour le problème ma_ta21_bi. Ce qui donne des déviations de 0.06% pour le problème ma_ta11_bi et 0.34% pour le problème ma_ta21_bi. L'utilisation de l'heuristique *NEH* pour la génération de la population initiale, fait passer le meilleur *makespan* trouvée de 2330 à 2303 pour le problème ma_ta21_bi. La somme des retards passe de 1508 à 1392 pour le problème ma_ta11_bi et de 1062 à 1097 pour le problème ma_ta21_bi.

Un volume important de travail reste à faire. Premièrement, nous proposons l'extension de l'étude vers un nombre d'objectifs supérieur à 2. Ce qui permettra d'analyser les performances de la méthode quand le nombre d'objectifs augmente. De nouveaux critères de mesure de la performance tels que l'entropie[Bac 98b] et la contribution[Tal 99], seront alors nécessaire puisque la comparaison graphique s'avérera impossible. D'autres mécanismes d'hybridation sont à étudier notamment avec la recherche tabou. L'adaptation probabiliste d'autres heuristiques utilisées dans la résolution du problème de flow shop sont aussi à envisager.

Nous proposons aussi:

1. L'étude d'autres problèmes d'ordonnancement multicritère tel que le Job Shop ou l'emploi du temps.

2. L'étude de problèmes où un ordre de préférence subsiste entre les différents critères.

3. L'analyse des mécanismes de prise de décision.

Références bibliographiques:

[Abr 91]	D.Abramson "**Construction school time tabling using simulated annealing : Sequential and parallel algorithm**" *Management science, volume 37, Janvier 1991, USA.*
[Abr 92]	D.Abramson, J.Abela "**A parallel Genetic Algorithm for solving the school time tabling problem**" *15 Australian computer science conference, Fevrier 1992.*
[All 92]	R. Allenson. "**Genetic Algorithms with gender for multi-function optimisation**" *Technical Report EPCC-SS92-01, Edinburg Parallel Computing Center, Edinburg, Scotland, 1992.*
[Ans 97]	Nirwan Ansari, Edwin Hou " **Computation Intelligence for Optimization**" *Kluwer Academic Publishers.1997.*
[Bac 98]	B. D. Backer, V. Furnon "**Local search in constraint programming: experiements with tabu search on the vehicle routing problem**" *Meta-Heuristics Advances and trends in local search paradigms for optimization, Kluwer Academic Publishers, Page 63, 1998.*
[Bac 98b]	V.Bachelet, Z.Hafidi, P.Preux, E-G.Talbi "**Vers la coopération des métaheuristiques**" *www.lifl.fr, 1998*
[Bak 85]	J.E.Baker "**Adaptive selection methods for Genetic Algorithms**" *Proceeding of International conference on Genetic Algorithms and their application, 1985, Page 101.*
[Bak 87]	J.E.Baker "**Reducing bias and inefficiency in the selection algorithme**" *Second International Conference on Genetic Algorithms, 1987, Page 14.*
[Bea 93]	D.Beasley, D. Bull, R. Martin "**A sequential Niche Technique for Multimodal Function Optimization**" *Evolutionary Computation Vol 1, N° 2, 1993, Page 101.*
[Ben 97]	P.J.Bentley and J.P. Wakefield "**Find an acceptable Pareto-optimal solutions using multiobjective Genetic Algorithms**". *Springer Verlag, London Page 231, June 1997.*
[Ben 99]	F. Ben Abdelaziz, S. Krichen, J. Chaouchi " **A hybrid heuristic for Multiobjective Knapsack problems**" *Met-heuristics advances and trends in local search paradigms for optimization, Klewer Academic Publishers, 1999, page 205.*
[Ben 98]	M.Ben-Daya, M. Al-Fawzan "**A tabu search approch for the Flow shop scheduling problem** " *European Journal of Operational Research, Vol 109, Page 88, 1998.*
[Ben 00]	F.Benameur, B.Kitoun "**Résolution du problème de routage de véhicules uni et multicritère à l'aide des algorithmes génétiques**" *Thèse d'ingénieur Encadré par M. Rahoual et H.Mabed, soutenu en Juin 2000.*
[Bil 99]	J.C. Billaut, V. T'kindt "**Les problèmes d'ordonnancement d'atelier multicritères**" *Rapport interne N° 206, Université François Rabelais, Tours, France.*
[Bra 91]	S.B.Brah, J.L.Hunsucker " **Branch & Bound algorithm for the flow-shop whith multiple processors**" *European Journal of Operational Research, Vol 51, 1991; page 88.*
[Car 88]	J.Carlier, P.Chrétienne "**Problèmes d'ordonnancement modélisation/ Complexité/ algorithmes**" *Edition Masson 1988.*
[Car 89]	J.Carlier, E. Pinson "**An algorithm for solving the Job-Shop Problem**" *Managment science, Vol 35, N° 2, Fév 1989. Page 164.*
[Car 90]	R. Carraway, T. L. Morin, H. Moskowitz "**Generalized dynamic programming for multicriteria optimization**" *European journal of operational research, Vol 44, page 95, 1990.*
[Cas 95]	Y. Caseau, F. Laburthe "**Improving Branch and Bound for Jobshop Scheduling with Constraint propagation**" *Rapport Interne de l'Ecole Normale Supérieure de Paris, France, 1995.*
[Cau 95]	C. Caux, H. Pierreval, M-C Portmann "**Les algorithmes génétiques et leur application aux problèmes d'ordonnancement**"

	RAIRO, Automatic Control Production Systems, Numéro Spécial Ordonnancement, Volume 29 N°4-5, Page 409, 1995..
[Che 91]	M. Chergui "**Modèle structuré d'un algorithme d'affectation des enseignements aux locaux**" *Thèse de Magister, Institut de mathématique à l'université de sciences et de technologie, Alger.*
[Chr 76]	N. Christofides "**The vehicle routing problem**" *Revue Française d'Automatique, Informatique et Recherche Opérationnelle, Vol 10, N° 2, Page 55, 1976.*
[Coe 96]	C. A. C. Coello "**An empirical study of evolutionary techniques for multiobjective optimization in engeneering design**" *Un résumé de thèse de doctorat soumis au departement of computer science of the graduate school of Tulane University, avrile 1996.*
[Coe 98]	C.A. C. Coello "**Using the Min-Max method to solve multiobjective optimization problems with genetic algorithms**" *LNCS, Springer Verlag, 1998.*
[Col 97]	A. Colorni, M. Dorigo, V.Maniezzo, G. Roghini, M. Trubian "**Heuristics From Nature For hard Combinatorial Oprtimization Problems**" *International Transaction In Operational Research, Vol 3, N°1, Page 1.*
[Cos 94]	D. Costa "**A tabu search algorithm for computing an operational timetable**" *European journal of operational research, Vol 76, page 98, 1994.*
[Das 97]	I. Das "**Multi-objective optimization**" *http://www.owlnet.rice.edu, Rice University, Janvier 1997.*
[Deb 99]	K. Deb "**Solving Goal Programming Problems using multi-objective Genetic Algorithms**" *IEEE, Page 77, 1999.*
[Dia 92]	B. Adenso-Diaz, "**Restricted neightborhood in the tabu search for flow-shop problem** " European Journal of Operational Research, Vol 62,N° 1, 1992, page 27.
[Dow 96]	K.A.Dowsland "**Simulated Annealing solutions for multi-objective scheduling and timetabling**" *Moderne Heuristic search methods, Editor V.J.Rayard-Smith, I.H.Osman, C.R.Reeves and G.D.Smith©19936 John Wiley & Sons Ltd.*
[Ehr 00]	M. Ehrgott, X. Gandibleux "**An annotation bibliography of multiobjective combinatorial optimization**" *Tecnical Repport N° 62/2000, Université de Kaiserslautern, avril 2000.*
[Esh 91]	L.J.Eshelman, J.D. Shaffer " **Preventing premature convergence in genetic algorithms by preventing incest** " *Fourth international conference on genetic algorithms, San Mateo, California, Morgan Kaufmann Pub, 1991, page 115.*
[Esq 99]	S. C. Esquivel, H. A. Leiva, R. H. Gallard "**Multiplicity in genetic algorithms to face multicriteria optimization**" *IEEE, Page 85, 1999.*
[Far 85]	Henry Farreny, Malik Ghallab "**Eléments d'intelligence artificielle**" *Edition Hermes 1985.*
[Far 97]	Henri Farreny "**Recherche heuristiquement ordonnée : générations compatibles avec la complétude et l'admissibilité**" *Technique et science informatiques. Volume 16- N° 7 1997. page 925.*
[Fle 95]	G. Fleury " **Applications de méthodes stochastiques inspirées du récuit simulé à des problèmes d'ordonnancement**" *RAIRO, Automatic Control Production Systems, Numéro Spécial Ordonnancement, Volume 29 N°4-5, Page 445, 1995*
[Fon 95]	C. M. Fonseca, P. J. Fleming "**An overview of Evolutionary algorithms in multiobjective optimization**" *Evolutionary Computation, Vol 3, N° 1, page 1, Spring 1995.*
[Fon 95b]	C. M. Fonseca, P.J. Fleming "**Multiobjective genetic algorithms made easy: selection, sharing and mating restrictions**" *In IEEE Int. Conf On Gentic Algorithms in Engineering System: Innovations and Applications, Page 45, Sheffield, UK, 1995.*
[Fon 98]	Cyril Fontlupt, Denis Robilliard, Philippe Preux, El-Ghazali Talbi "**Fitness Landscapes and performance of meta-heuristics"**

	Meta-Heuristics Advances and trends in local search paradigms for optimisation, Kluwer Academic Publishers, 1998. Page 257.
[Fou 85]	M. P. Fourman. "**Compaction of symbolic layout using genetic algorithms**". *Conf on Getic Algorithms and their Applications,Pittesburgh, 1985.*
[Fog 66]	L.J.Fogel, A.J.Owens, M.J.Walsh "**Artificial Inteligence through adaptation**", *Wiley, New York, 1998.*
[Fre 82]	E. C. F. Freuder "**A sufficient condition for backtrack-free search**" *Journal of ACM, Vol 29, N° 1, 1982.*
[Fri 94]	D.Frigioni, A.Spaccamela, U.Nanni "**Dynamization of backtrack-free search for the constraint satisfaction problem**" *Lecture note in computer science 778 Février 1994. page 136.*
[Ger 66]	W. S. Gere "**Heuristic in job shop scheduling**" *Managment science, Vol 13, N° 3, Nov 1966. Page 167.*
[Gen 96]	Mitsuo Gen, Runwei Cheng "**Genetic Algorithms & Engineering Design**" A wiley-Interscience Publication, John Wiley & Sons, Inc. 1996.
[Glo 95]	Fred Glover "**Tabu search** " *Rapport interne, Colorado University, USA.*
[Glo 96]	Fred Glover "**What's in an adaptive landscape?**" *Journal of heuristic, Rapport de travail 1996.*
[Gol 87]	D. E. Goldberg, J. Richardson, "**Genetic algorithms with sharing for multimodal function optimisatio**n". *In second Int. Conf. on Genetic Algorithms ICGA'2, page 41, 1987*
[Gol 89]	D. E. Goldberg "**Genetic Algorithms in Search, Optimization, and Machine Learning**" *The university of Alabama, Eddison Wesley, 1989.*
[Gon 78]	T.Gonzalez, S.Sahni " **Flowshop and Job-shop Schedules : Complexity and Approximation** " *Operational Research, Vol 26, N°1, 1978, page 36.*
[Gra 99]	E.Grave, A.Liagre "**Gestion de colonisation de fourmis pour le problème de clique maximale**" Mémoire de Maîtrise d'informatique à l'université de Lille, France, 1999.
[Gre 98]	Peter Greistorfer "**Hybrid Genetic Tabu Search For A Cyclic Scheduling Problem**" *Met-heuristics advances and trends in local search paradigms for optimization, Klewer Academic Publishers, 1999, page213.*
[Gué 98]	Christelle Guéret, Nerendra Jussien "**Using intelligent backtracking to improve Branch-and-bound methods:an application to Open-Shop problems**" *International Workshop on project management and scheduling, 1998.*
[Gup 69]	N.D.Gupta " **An improved Combinatorial Algorithm For The Flowshop-Scheduling Problem** " *Oprerational Research, Vol 19, 1969, page 1753.*
[Haj 92]	P. Hajela, C. Y. Lin "**Genetic search strategies in multicriterion optimal design**" *Structural Optimisation, (4) Page 99, 1992.*
[Har 94]	G. Harik "**Finding Multiple Solutions in Problems Of Bounded Difficulty**" *Rapport Interne du Illinois Genetic Algorithms Laboratory, Departement of General Engineering, University of Illinois at Urbana-Champaign.*
[Hao 98]	K-K Hao, Jèrome Pannier "**Simulated Annealing and Tabu Search for Constraint Solving**" *Fifth International Symposium on Artificial Intelligence and Mathematics (AIM'98), Florida, USA, Janvier, 1998.*
[Hel 60]	J.Heller " **Some Numerical Experiments For a MxJ Flow Shop And Its Decision Theoretical Aspects** " *Operational Research, Vol 8, 1960, page 178.*
[Her 91]	A. Hertz "**Tabu search for large scale timetabling problems**" *European journal of operational research, Vol 54, page 39, 1991.*
[Her 94]	A. Hertz, B. Jaumard, C. C. Ribero, W. P. Filho "**A multi-criteria tabu search approch to cell formation problems in group technology with multiple objectives**" *Recherche Opérationnelle, Vol 28, N°3, Page 303, 1994.*
[Her 95]	A. Hertz, M. Widmer "**La méthode Tabou appliquée aux problèmes d'ordonnancement**" *Automatique-Productique. Informatique Industrielle Numéro Spècial ordonnancement, Vol 29, N° 4-5, page 353, 1995.*

[Hir 99]	T. Hiroyasu, M. Miki, S. Watanabe "**Distributed Genetic algorithms with a new sharing approch in multiobjective optimization problems**" *IEEE 1999.*
[Hol 75]	J.Holland "**Adaptation in natural and artificial systems**" *Michigan Press University, Ann Arbor, MI, 1975.*
[Hor 94]	J. Horn, N. Nafpliotis "**Multiobjective Optimization Using The Niched Pareto Genetic Algorithm**" *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Volume 1, 1994 (ICEC '94)*
[Hou 96]	C.R.Houck, J.A Joinies, M.G. Kay "**A genetic Algorithm for function optimization: A Matlab implementation**" *North Carolina State University www.ncsr.edu.*
[Hou 97]	C. R. Houck, J. A. Joinies, M. G. Kay. "**Characterizing Search Space for Tabu Search**" *North Carolina State University www.ncsr.edu.*
[Kou 98]	C. Koulamas "**A new constructive heuristic for flowshop scheduling problem**" *European journal of operational research, Vol 105, Page 66, 1998.*
[Küf 98]	K.H Küfer "**On the asymptotic average number of efficient vertices in multiple objective linear programming**" *Journal of complexity. Volume 14 1998. page 333.*
[Kul 85]	R. V. Kulkarni, P. R. Bhave "**Integer programming formation of vehicle routing problems**" *European Journal of Operational Research, Vol 20, Page 58, 1985.*
[Le 98]	Patrick Le Goeuëslier "**Un ordonnancement asymptotiquement optimal sur un domaine semi-infini multidimensionnel**" *Technique et science informatiques. Volume 17-n° 7. 1998. Page 895.*
[Lis 96]	J. Lis, A. E. Eiben "**A multi-sexual genetic algorithm for multi-objective optimization**" *International Conference on Genetic Algorithms ICGA, Pages 59, Nagoya, Japon, 1996.*
[Lou 99]	S. J. Louis, X. Yin , Z. Y. Yuan "**Multiple Vehicle Routing With Time Windows Using Genetic Algorithms**" *Congress on Evolutionary computation, 6-9 Juillet, Washington, USA, Page 1804, 1999.*
[Mab 97]	M. H. Mabed, R. Kameche "**Contribution à la résolution du problème de l'emploi du temps par les algorithmes génétiques**" *Mémoire d'ingénieur à l'université de USTHB dirigé par M. Rahoual, Algérie, 1997.*
[Mah 95]	S. W. Mahfoud "**A Comparison of Parallel and Sequential Niching Methods**" *Proceeding of the sixth International Conference on Genetic Algorithms, 1995, Page 136.*
[Mar 98]	Espinouse Marie-Laure "**Flow-Shop et extension : chevauchement des tâches, indisponibilité des machines et système de transport**" *Thèse de doctorat de l'université Joseph Fourier, soutenue le 18 Déc 1998.*
[Mau 84]	M.L.Mauldin "**Maintaining diversity in genetic search**" *Proceedings of the national conference on artificial inteligence, 1984, Page 247.*
[Min 93]	Steven Minton "**An analytic learning system for specializing heuristics**" *13th international joint conference on AI. Volume 3. Septembre 1993. Page 922.*
[Mun 91]	Traian Muntean, El-Ghazali Talbi "**Méthodes de placement statique des processus sur architectures parallèles**" *Technique et science informatiques. Volume 10-n°5. Page 355. 1991.*
[Mur 94]	T.Murata, H.Ishibuchi "**Genetic Algorithm and neighborhood Search For Fuzzy Flowshop Scheduling Problems**" *Fuzzy Sets Syst, Volume 67, 1994, Page 81.*
[Mur 98]	T.Murata, H.Ishibuchi " **A Multi-objectives Genetic Loacal Search Algorithm and Its Application Flow-shop Scheduling** " *IEEE Transaction System. Vol 28, N°3, 1998, Page 392.*
[Nag 95]	A. Nagar, J. Haddock, S. Heragu "**Multiple and bicriteria scheduling: a literature survey**" *European Journal of Operationnal Research, Vol 81, Page 88, 1995.*
[Naw 83]	M. Nawaz, E. E. Enscore, I. Ham "**A heuristic algorithm or the m-machine, n-job flow-shop sequencing problem**" *Omega, N° 11, Page 91, 1983.*
[Now 99]	E. Nowicki "**The permutation Flow shop with buffers : A tabu search approch**" *European Journal of Operational Research, Vol 116, Page 205, 1999.*
[Oba 98]	S. Obayashi, S. Takahashi, Y. Takeguchi "**Niching and elitist models for MOGAs**"

	In parallel problem solving from nature PPSN'5, Page 160, 1998.
[Oei 91]	C. K. Oei "**Tournament Selection, Niching, and the Preservation of Diversity**" *University if Illinois at Urbana-Champaign, Urbana, IlliGAL Report N° 91011 Décembre 1991.*
[Ozc 96]	E.Ozcan, C.K.Mohan "**Simulated Annealing and Genetic Algorithms for Partial Shape Matching**" *The Eleventh International Symposium on Computer and Information Sciences, 1996, page 173.*
[Pae 95]	B. Paechter "**Optimising a presentation Timetable using evolutionary algorithms**" *Departement of computer studies Napier, University of Edinburgh, 1995.*
[Pap 95]	C. H. Papadimitriou "**Computational Complexity**" *University of California, Addison-Wesley Publishing Company, 1995.*
[Paz 00]	F. Pezzella, E. Merelli "**A tabu search method guided by shifting bottleneck for the job shop scheduling problem**" *European Journal of Operational Research, Vol 120, Page 297, 2000.*
[Pre 95]	P. Preux, E-G. Talbi. "**Assessing the Evolutionary Algorithm paradigm to solve hard problemes**" *Rapport Interne Université de sciences et Technologies de Lille, France.1995.*
[Rah 99]	M. Rahoual, H. Mabed "**Genetic algorithms to solve Timetabling problem** " *Fourth Conference on Evolutionary Algorithm EA'99, Denkerque, France, 1999.*
[Raj 91]	C.Rajendran, D.Chaudhuri " **An efficient heuristic approach to the scheduling of jobs in flow-shop** " *European Journal Of Operational Research, Vol 61, 1991, page 318.*
[Ram 96]	D.J.Ram, T.H.Sreenivas, K.G.Subramaniam "**Parallel Simulated Annealing Algorithms**" *Journal of Parralel and Distributed Computing, Vol 37,Page 207, 1996.*
[Ren 94]	J. M. Renders, "**Algorithmes génétiques et réseaux de neurones**" *Editions Berti, 1994.*
[Ree 95]	C. Reeves "**A Genetic algorithm for flow shop sequencing**" *Computers and Operations Research, Vol 22, Pages 7, 1995.*
[Ric 96]	E. b. Richards, Y. h. Li, Y. J. Jiang, N. A. Azarmi "**Localized Simulated annealing in Constraint satisfaction and Optimization**" *Moderne Heuristic search methods, John Wiley & Sons Ltd, Page 28, 1996.*
[Ros 95]	P. Ross, D. Corne "**Improving evolutionary time tabling with Delta evaluation and directed mutation**" *Departement of Artificial intelligence, University of Edinburhg, 1995.*
[Sak 90]	Masatoshi Sakawa, Hitoshi Yano "**An interactive fuzzy satisficing method for generalized multiobjective linear programming problems with fuzzy parameters**" *Journal of the Operational Research Society, Vol 46, Page 958, 1990.*
[Sal 99]	F. S. Salman, J. Kalagnanam, S. Murthy "**Cooperative strategies for solving the bicriteria sparse multiple knapsack problem**" *IEEE, Page 53, 1999.*
[Sar 92]	P. Sarafini "**Simulated annealing for multiple objective optimization problems**" *Tenth International Conference on multiple criteria decision making, Page 87, Taipei, Juillet 1992.*
[Sia 95]	Patrick Siarry "**La méthode du récuit simulé: théorie et applications**" *RAIRO, Automatic Control Production Systems, Numéro Spécial Ordonnancement, Volume 29 N°4-5, Page 535, 1995..*
[Sim 82]	Simon French, D.Phil "**Sequencing and sheduling: An introduction to the mathematic of the Job-Shop"** *Department of Decision Theory, University of Manchester. John Wiley & Sons Edition.*
[Sch 85]	J. D. Schaffer. "**Multiple objective optimisation with vector evaluated genetic algorithms**", *In J. J. Grefenstette, editor, IGGA Int. Conf on Genetic Algorithms, Page 93. Lawrence Erbaum, 1985.*
[Sha 91]	A.B.Shaukat, J.L.Hunsucker " **Branch & Bound algorithm for the flowshop whith multiple processors**" *European Journal of Operational Research, Vol 51, 1991; page 88.*
[Son 99]	L.Sondergeld, S.Vob "**Cooperative Intelligent Search Using Adaptive Memory Techniques**" *Met-heuristics advances and trends in local search paradigms for optimization, Klewer Academic Publishers, 1999. Page 297.*
[Sri 95]	N. Srinivas, K. Deb "**Multiobjective optimisation using non-dominated sorting in genetic algorithms**" *Evolutionary Computation 2(8):Page 221, 1995.*

[Ste 85]	Ralph.E.Steuer "**Multiple criteria optimization: Theory, computation and application**" *John Wiley & Sons Edition, 1985.*
[Ste 91]	B. S. Stewart, C. C. White "**Multiobjective *A**" *Journal of association for computing machinery, Vol 38, N° 4, Page 775, 1991.*
[T'ki 99]	Viencent T'Kindt "**Etude des problèmes d'ordonnancement multicritères**" *Thèse de doctorat de l'université de Tours, France, Soutenue le 6 déc 1999.*
[Tai 93]	E. Taillard "**Benchmarks for basic scheduling problems**" *European Journal of Operational Research, Vol 64, Page 278, 1993.*
[Tal 91]	E.G.Talbi, P.Bessiere "**Mise en œuvre et évaluation d'un Algorithme Génétique sur les machines SuperNode**" *First Maghribin Symposium on programming and System, octobre 1991, Algerie, Page 11.*
[Tal 95]	E.G.Talbi "**Algorithmes Génétiques parallèles: techniques et applications"** *Publication université de Lille, Janvier 1995.*
[Tal 98a]	E.G.Talbi, Z.Hafidi, J.M.Geib "**Parallel Tabu Search For Large Optimization Problems**" *Meta-Heuristics Advances and trends in local search paradigms for optimisation, Kluwer Academic Publishers, 1998. Page 345.*
[Tal 98b]	E.G.Talbi, Z.Hafidi, J.M.Geib "**A Parralel Adaptive Tabu Search Approch**" Avant-impression a soumettre à Elsevier Preprint, Septembre 1998
[Tal 99]	El-ghazali Talbi **"Métaheuristiques pour l'optimisation combinatoire multi-objectifs: Etat de l'art"** *Rapport interne, Université de sciences et Technologies de Lille, France. Non encore apparu, juin 1999.*
[Tal 99b]	E-G Talbi "**A Taxonomy of hybrid metaheuristics**" *Journal of combinatorial optimization, Kluwer academic publishers, 1999.*
[Tuy 98]	D. Tuyttens "**An improved MOSA method for solving multi-objective combinatorial optimization problems**" *Rapport interne, Laboratory of mathematics operational research, Faculté Polytechnique de Mons, Belgique, 1998.*
[Ulu 98]	E. Ulungu, J. Teghem, C. Ost "**Efficiency of interactive multi-objective simulated annealing through a case study**" *Journal of the operational research society, Vol 49, N° 10, Page 1044, 1998.*
[Var 98]	R. Varela, A. Gomez, C. R. Vela, J. Puente, C. Alonso "**Heuristic Generation of the initial Population in solving job shop problem by evolutionary strategies**" *Centro de Inteligencia Artificial, Université de Oviedo, Espagne.*
[Vel 98]	D. A. V. Veldhuizen "**Multiobjective evolutionary algorithm research: a history and analysis**" *Rapport technique N° 98-30 du Department of Electrical and computer engineering, air force institute of technology, USA, Décembre 1998.*
[Vel 99]	D. A. V. Veldhuizen "**Multiobjective evolutionary algorithms: classification, analyses, and new innovations**" *Thèse de doctorat de la Faculty of the the graduate school of engineering of the air force institute of technologiy, USA, Juin 1999.*
[Vin 76]	P.H. Vincke "**Une méthode intéractive pour la programmation linéaire à plusieurs fonctions économiques**" *Revue Française d'Automatique, Informatique et Recherche Opérationnelle, Page 5, juin 1976.*
[Wer 89]	A. de Werra, A. Hertz "**Tabu Search Techniques: A Tutorial and an Application to Neural Networks**" *OR Spektrum, Vol 11, page 131, 1989.*
[Wez 95]	M. C. Van Wezel, G. N. Kok "**Genetic Improvement of railway timetables**" *Departement of computer scienc, Ultrecht, The Netherlands, 1995.*
[Wie 93]	Andrzej Wierzbicki "**Augmented simplex: a modified and parallel version of simplex method based on multiple objective an subdifferentiel optimisation approch"** *Rapport de travail International Institute for applied system Analysis, octobre. 1993.*
[Woo 99]	David. L. Woodruff "**A Chunking Based Selection Strategy For Integrating Meta-heuristics With Branch and Bound**" *Met-heuristics advances and trends in local search paradigms for optimization, Klewer*

	Academic Publishers, 1999. Page 499.
[Zit 98]	E. Zitzler, Lothar Thiele "**An evolutionary algorithm for multiobjective optimization: the strength Pareto approach**" *Technical Report N° 43, Computer Engineering and Communication Networks, Swiss Federal Institute of Technology, mai 1998.*
[Zit 99]	E. Zitzler "**Evolutionary algorithms for multiobjective optimization: methods and application**" *Thèse de doctorat de Swiss federal institute of technology, Zurich, Switzerland, Novembre 1999.*
[Zwe 89]	Monte Zweben, Megan Eskey "**Constraint satisfaction with delayed evaluation**" *Eleventh international joint conference on AI. Volume 2. Août 1989. Page 875.*

Publications Personelles:

[1]	Mohamed Hakim Mabed, Malek Rahoual, El-Ghazali Talbi and Clarisse Dhaenens "**A Genetic Algorithm for Multicriteria Flow Shop Scheduling**", MCDM 2000: XV-th International Conference on Multiple criteria Decision Making, Ankara, Turky, July 10-14, 2000.
[2]	Mohamed Hakim Mabed, Malek Rahoual, El-Ghazali Talbi and Clarisse Dhaenens "**A Genetic Algorithm for Multicriteria Flow Shop Scheduling**" , Version étendue soumise au journal EJOR.
[3]	Mohamed Hakim Mabed, Malek Rahoual, El-Ghazali Talbi and Clarisse Dhaenens "**Using Genetic Algorithms to schedule Multicriteria Flow Shop**" MSPN 2000, Workshop on Multiobjective Problems Solving from Nature, Paris, France, 16 September 2000.
[4]	Mabed, Malek Mohamed Hakim Mabed "**Hybridization of Genetic Algorithms and Tabu search for Timetabling problems**", Soumis à JIM Journal International of metaheuristics (Canada).

Sites Internet:

www.lania.mx/~ccoello/EMOO/EMOObib.html	Site contenant une multitude d'articles sur l'optimisation multicritère.
http://mat.gsia.cmu.edu/	Site proposons des instances de problèmes combinatoires, parmi les ceux de *Taillard.*