N° d'ordre : 2307

THESE

présentée pour obtenir le titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Ecole doctorale : Transferts, Dynamique des Fluides, Energétique et Procédés

Spécialité : Génie des Procédés et de l'Environnement

par

Antonin PONSICH

STRATEGIES D'OPTIMISATION MIXTE EN GENIE DES PROCEDES

APPLICATION A LA CONCEPTION D'ATELIERS DISCONTINUS

Thèse soutenue le 14 Décembre 2005 devant le jury composé de :

M.	Antonio ESPUÑA	Rapporteur
M.	Arsène ISAMBERT	Rapporteur
M.	Maurice COMTAT	Président
M.	Joseph NOAILLES	Membre
M.	Luc PIBOULEAU	Membre
Mme	Catherine AZZARO-PANTEL	Directrice de thèse
M.	Serge DOMENECH	Membre invité

Et pis voilà…

(A.A. Aguilar Lasserre, entre oct. 2002 et janv. 2006)

Je souhaite tout d'abord témoigner ma reconnaissance à Antonio Espuña, Professeur à l'Universitat Politècnica de Catalunya (venu, un peu dans la précipitation, depuis Barcelone) et Arsène Isambert, Professeur à l'Ecole Centrale de Paris, qui m'ont fait l'honneur d'être les rapporteurs de ce mémoire. Leurs commentaires avisés m'ont proposé une vision plus ample, des perspectives plus larges sur mes travaux.
D'autre part, je voudrais exprimer toute ma sympathie à Maurice Comtat, Professeur à l'UPS (Toulouse) qui, très en verve ce jour-là, a su détendre l'ambiance de la soutenance. Je remercie aussi sincèrement Joseph Noailles, Professeur à l'ENSEEIHT (Toulouse), pour l'intérêt visible qu'il a porté à la lecture de ce mémoire et pour la finesse de ses remarques pendant la soutenance.

Par ailleurs, je veux exprimer vivement ma gratitude aux personnes qui m'ont suivi et guidé durant ces années de thèse, en commençant par Catherine Azzaro-Pantel, que je remercie pour ses idées, pour sa disponibilité, pour l'attention qu'elle a porté non seulement à mon travail, mais aussi à mes préoccupations autour du travail.
Puis je tiens à adresser de chaleureux remerciements à Luc Pibouleau, qui, même s'il n'était pas mon directeur de thèse officiel, a toujours pris le temps de s'intéresser à mes travaux, de participer à l'effort de réflexion, de s'asseoir à côté de moi pour m'aider à surmonter les difficultés… merci pour tout ce que j'ai appris avec vous, en optimisation ou simplement dans le domaine du numérique.
Je souhaite également remercier Serge Domenech et André Davin pour leur attention, d'autant plus profitable au moment de la rédaction du manuscrit. Je remercie Jorge Montagna, au cas où il lit un jour ces lignes, qui m'a soulagé, au moment critique de la fin de thèse, en répondant à mes doutes.
Outre mes encadrants, mes remerciements vont aussi à l'ensemble du laboratoire, et plus particulièrement aux thésards de l'équipe, avec lesquels j'ai eu la chance de partager, pendant quelques années, un bureau très international : Sofiane, Beto, Oscarín, Firooz, Ludo et, même s'il nous a fait voir rouge de temps en temps, el pibe Adrian. Je n'oublie pas non plus les stagiaires, de passage ou qui finalement sont restés au laboratoire : Azze-Idine, Maria, Nelson ; ni les terroristes du service info, Stéf et Denis ; ni non plus Christine, Dany, Jocelyne et Claudine pour leur bienveillance et leurs sourires quotidiens. J'aimerais enfin saluer les personnes avec lesquelles j'ai passé de nombreuses heures, en enseignement à l'ENSIACET : merci à Joël, Michel, Luc, Marion et Catherine.

D'autre part, j'ai de la peine à exprimer toute l'amitié que je ressens pour ceux qui m'ont entouré ces trois années, du début à la fin : Lolo, Alberto, Jeanne, Jojo, auxquels je dois inévitablement rajouter Alex et son épaule, et Ricket et son canapé : vous avez pris soin de moi mieux que je n'aurais pu le faire, tant matériellement que par ces moments partagés, ces discussions, ces repas… à renouveler.
Je veux aussi dire un mot à tous ceux qui, de près ou de loin, m'ont soutenu et qui ont fait que ces trois années n'ont pas été seulement trois années de thèse, mais aussi trois années pleines de moments agréables… alors en vrac, merci à Isa, à Sophie, à Gus, à Tos et Hind, à Malika, à Laurent et son overkitch ; un spécial kif pour Seb et Karis ; merci aux All Stars rugby pour leur bon esprit, à dj Chloé, merci à Boule et les autres et leur VélOvalie…

Ces trois années se sont déroulées dans une ambiance hispanophone, pour ne pas dire complètement mexicaine, alors je dois nécessairement ajouter quelques mots en espagnol… pues con toda la amistad y todo el cariño que ustedes me enseñaron, hicieron que me sintiera como adoptado por la familia mexicana de Toulouse, y esto, lo considero como una suerte muy grande. Entonces, este abrazo es para ustedes, Memito, Susi y los cachorros (MariFer y Dani), Leito salut-ça-va y Sofia, don tío Jano, Oscarín, Tatiana y la bebé (los bebés !), Maria Elena, Lomi el Cubano, Lili y François, Nacho, Claudia y David, Luisita, …
Y más especialmente, Beto, por todos los momentos que pasamos juntos… Sólo por haberte conocido, me alegro de haber hecho el doctorado. Nos veremos pronto chamaco, tú, yo, y el casco de gel !

Por fin, gracias a ti, Ana, por apoyarme, por animarme los días que nada quería funcionar, por jalarme de la mano cuando no veía la salida. Hubiera sido tan diferente sin ti… que bien que fue contigo.

Et finalement, je voudrais remercier toute ma famille pour leur soutien, mes parents et Aurélia, mes grands-parents et Pierre. Merci pour votre appui inconditionnel non seulement concernant le travail, mais aussi dans tout le reste… merci.

TITRE : Stratégies d'optimisation mixte en Génie des Procédés – Application à la conception d'ateliers discontinus

RESUME :

La conception de procédés discontinus implique la formulation de problèmes d'optimisation complexes. Les exigences industrielles, conjuguant rentabilité, flexibilité, respect de normes environnementales… conduisent à représenter les ateliers discontinus de manière toujours plus fidèle. Le degré élevé de sophistication des modèles résultants explique alors la difficulté rencontrée au moment de les traiter. Jusqu'à présent, l'essentiel des travaux était consacré au développement de méthodes déterministes mais une attention accrue est portée actuellement à la classe des métaheuristiques.

Ce travail a ainsi pour objectif de proposer une méthodologie pour le traitement d'un problème d'optimisation non-linéaire en variables mixtes, dans le cadre du Génie des Procédés. Il s'agit de fournir des lignes directrices aidant à guider, selon la formulation du problème, vers le choix pertinent d'une méthode d'optimisation.

Le support de l'étude, permettant d'évaluer les performances des méthodes, est un problème typique de conception d'ateliers discontinus multiproduit. Une formulation « orientée équation » est retenue pour permettre l'utilisation tant des méthodes de Programmation Mathématique que des métaheuristiques. Un jeu de onze exemples de complexité croissante est construit pour déterminer, selon la taille de l'instance étudiée, l'efficacité de chaque technique de résolution.

Le choix de représentants de chaque classe de méthodes est ainsi justifié. D'une part, deux solveurs disponibles sous l'environnement de modélisation *GAMS* sont retenus : il s'agit de *SBB* (Branch & Bound) et *DICOPT++* (méthode des Approximations Externes). D'autre part, un Algorithme Génétique (AG) représente la classe des métaheuristiques. Un axe complémentaire d'étude consiste à ajuster ses procédures de fonctionnement. Cette phase constitue un enjeu primordial pour l'application générique de toute méthode stochastique à des problèmes industriels, prenant en compte des variables mixtes et la gestion de contraintes.

Les résultats obtenus prouvent la performance de la méthode de Branch & Bound, en temps de calcul et qualité de la solution, alors que la méthode des Approximations Externes est mise en échec assez rapidement. L'AG fait également preuve d'une efficacité satisfaisante. La comparaison avec les solutions optimales fournies par *SBB* permet de fixer des procédures appropriées de codage et de gestion des contraintes. Les problèmes sont ensuite traités avec des variables d'optimisation purement discrètes : l'AG reste aussi efficace alors que les méthodes déterministes sont rapidement dépassées par la combinatoire des problèmes. La stratégie est finalement validée sur un exemple de bioprocédés formulé de manière similaire, impliquant différents types de variables relatives non seulement à la structure de l'atelier mais aussi aux conditions opératoires.

MOTS-CLES : Optimisation mixte non-linéaire (MINLP), Programmation Mathématique, Métaheuristiques, Algorithme Génétique, Conception d'ateliers discontinus.

TITLE : Mixed integer non-linear strategies for Process Engineering – Application to batch plant design

ABSTRACT :

Batch plant design area implies the formulation of complex optimisation problems. Industrial requirements combine profitability, flexibility, respect of environment-based rules,… Thus, the resulting models are more and more sophisticated in order to enable trully realistic representation of batch plants. That means an increasing solution difficulty. Until recently, research studies were mainly dedicated to deterministic methods development, but an increased attention is now given to the metaheuristics class.

So the aim of this work is to propose a methodology for mixed integer non-linear programming problems treatment, in the framework of Process Engineering. Guidelines will be provided in order to help to choose an appropriate optimisation method, according to the problem formulation.

A typical batch plant design problem will enable to evaluate the tackled methods performances. An analytical « equation-oriented » formulation is adopted to use Mathematical Programming methods as well as metaheuristics. A set composed of eleven increasing complexity examples is built to determine each method efficiency, according to the size of the studied instance.

Then, the operating mode consists in testing at least one method from both classes. On the one hand, two solvers available within the *GAMS* modelling environment were chosen, namely *SBB* (Branch & Bound technique) and *DICOPT++* (Outer Approximation technique). On the other hand, a Genetic Algorithm (GA) represented the stochastic class. A second theme of this study will deal with the tuning of GAs internal processes. These ones are an essential issue for a generic application of stochastic methods to engineering problems, which involve mixed variables and constraint handling.

The Branch & Bound proves to be really performing in terms of computational time and of result quality, while the Outer Approximation method fails quite quickly. GAs efficiency is also satisfactory. Comparison with optimal solutions provided by *SBB* enables to set appropriate encoding and constraint handling techniques. Besides, the problems are treated with pure integer optimisation variables : the GA keeps being performing while the deterministic methods fail because of the examples combinatorial effect. The strategy is finally validated with an example drawn from bioprocess engineering literature. The formulation is quite similar, and involves both (discrete) plant configuration variables and (continuous) process decision variables.

KEYWORDS : Mixed integer non-linear programming optimisation (MINLP), Mathematical Programmation, Metaheuristics, Genetic Algorithm, Batch plant design.

TÍTULO : Estrategias de optimización con variables mixtas en Ingeniería de Procesos – Aplicación al diseño de plantas con sistemas de producción en lotes

RESUMEN :

El diseño de procesos con sistemas de producción en lotes implica la formulación de problemas complejos de optimización. Las exigencias industriales, que engloban objetivos de rentabilidad, flexibilidad y respeto a normas del medio ambiente, conducen a representar plantas discontinuas de forma más realista. El grado de dificultad de los modelos tiene como consecuencia encontrar dificultades al momento de resolverlos. Hasta ahora, la mayoría de los trabajos fue dedicada al desarrollo de los métodos determinísticos, pero actualmente se han desarrollado métodos metaheurísticos.

Este trabajo tiene el objetivo de proponer una metodología para el tratamiento de un problema de optimización no lineal con variables mixtas. Se trata de proporcionar herramientas que ayuden a guiar la elección de un método de optimización adecuado, de acuerdo a la formulación del problema.

El ejemplo en estudio, con él que se lleva a cabo la evaluación de los métodos, es un problema típico de diseño de plantas discontinuas multiproducto. Una formulación « orientada-ecuación » es escogida para permitir el uso de los métodos de Programación Matemática y al mismo tiempo la aplicación de los métodos metaheurísticos. Un conjunto de once ejemplos con diferentes grados de dificultad es construido para determinar, según el tamaño de la instancia estudiada, la eficiencia de cada técnica de resolución.

Para elegir el método adecuado proponemos, por una parte, dos módulos disponibles del ambiente de modelización *GAMS* : *SBB* (Branch & Bound) y *DICOPT++* (Aproximaciones Externas). Por otra parte, un Algoritmo Genético (AG) representa la clase de métodos metaheurísticos. Un propósito adicional del estudio consiste en ajustar los procesos de funcionamiento. Esta fase constituye una condición fundamental para la aplicación genérica de los métodos estocásticos a problemas industriales, considerando variables mixtas y el manejo de restricciones.

Los resultados obtenidos prueban el buen funcionamiento del método de Branch & Bound, en términos de cálculo y calidad de la solución, mientras que el método de Aproximaciones Externas falla rápidamente. El AG muestra una eficiencia satisfactoria. La comparación de las soluciones óptimas proporcionadas por *SBB* permite fijar los procesos apropiados de codificación y de tratamiento de restricciones. En un segundo tiempo, los problemas son tratados con variables de optimización completamente discretas : el AG continua siendo eficiente mientras que los métodos determinísticos fallan debido al efecto combinatorio de los problemas. Finalmente, la estrategia es validada con un ejemplo de bioprocesos, formulado de manera similar, pero implicando, además de las variables discretas relacionadas a la estructura de la planta, variables continuas de condiciones de operación.

PALABRAS CLAVE : Optimización mixta no lineal (MINLP), Programación Matemática, Metaheurísticas, Algoritmo Genético, Diseño de plantas discontinuas.

SOMMAIRE

Page

INTRODUCTION

PRESENTATION DE LA PROBLEMATIQUE

Dans le domaine du Génie des Procédés, les problèmes de conception représentent un axe de recherche fondamental, qui suscite des travaux récurrents. L'examen de différents scénarii, générés par des études numériques, permet ainsi de prévenir un mauvais mode de fonctionnement ou d'optimiser un critère d'ordre technico-économique, environnemental, tenant à des aspects de sécurité… qu'il serait difficile, voire impossible, de corriger *a posteriori*. L'analyse préalable de ces problèmes, applicable tant à la conception d'ateliers qu'à celle de réseaux de réacteurs, ou d'échangeurs de chaleur et de matière, revêt donc un aspect primordial.

L'étape d'optimisation permet en général d'aboutir à une ou plusieurs solutions de bonne qualité, qui seront présentées aux décideurs, lesquels effectueront ensuite le choix le plus pertinent. Elle est cependant nécessairement précédée d'un travail de modélisation de l'atelier étudié. Le degré de précision de cette phase est bien entendu à double tranchant. Plus le modèle est proche de la réalité, plus les résultats obtenus sont fiables et applicables sans nuance dans le monde réel. En revanche, la contrepartie est évidemment la complexité du problème résultant et la nécessité de disposer d'outils suffisamment puissants pour le résoudre.

Actuellement, les exigences industrielles, conjuguant rentabilité, flexibilité, respect de normes de sécurité ou environnementales, ont conduit à représenter les ateliers de manière toujours plus fidèle, impliquant l'écriture de modèles de plus en plus complexes. Beaucoup de travaux ont ainsi été consacrés aux modèles d'ateliers fonctionnant en continu, qui impliquaient initialement des variables purement réelles, et qui intègrent peu à peu des variables entières de dimensionnement ou de représentation des installations.

Cette remarque vaut également pour la conception d'ateliers discontinus. Leur modélisation fait intervenir différents degrés de sophistication, tels que la prise en compte de facteurs économiques globaux, la modélisation de l'incertitude sur certains paramètres, la possibilité d'existence de bacs de stockage intermédiaire, la considération de critères environnementaux,… Il en résulte l'emploi fréquent de variables décisionnelles discrètes (parfois entières, le plus souvent binaires) au sein de la formulation. Elles sont régulièrement mises en jeu, conjointement avec des variables continues, dans des fonctions généralement non-linéaires et non-continues. Les problèmes d'optimisation résultants sont généralement difficiles à résoudre : problèmes non-linéaires en variables réelles, problèmes linéaires en variables mixtes voire, combinant les deux difficultés, problèmes de type mixte non-linéaire.

Cette complexité s'exprime également par un aspect fortement combinatoire induit, par les variables discrètes. Une grande partie des problèmes de conception d'ateliers discontinus appartient en effet à la classe des problèmes NP-difficiles (pour Non-Polynomial), c'est-à-dire pour lequel aucun algorithme de résolution en un temps polynomial n'est connu [AZZ05] : les temps de résolution augmentent de façon exponentielle avec le nombre de variables discrètes. Or, les exemples de taille industrielle impliquent des centaines de variables et des milliers de contraintes. Les temps de calcul deviennent donc rapidement rédhibitoires, en dépit de l'amélioration constante de la puissance des calculateurs et du développement de stratégies de calcul parallèle.

Une intensification de la recherche autour du développement de méthodes d'optimisation efficaces constitue un axe naturel pour répondre à ces exigences. En effet, le temps de calcul imposé par la complexité des modèles n'est pas le seul écueil pour la détermination de solutions faisables de bonne qualité. Les non-linéarités, l'existence d'optima locaux dus à la non-convexité des fonctions mises en jeu, la discrétisation de l'espace de recherche induite par la présence de variables entières, pénalisent fortement les performances des techniques d'optimisation. Ces dernières intéressent plus particulièrement les travaux de doctorat présentés dans ce mémoire, ciblés sur le développement d'une méthodologie pour aborder des problèmes d'optimisation mixte non-linéaire en Génie des Procédés. Le support de validation de l'étude est basé sur le problème de conception optimale d'ateliers discontinus. Ce chapitre d'introduction du problème s'organise comme suit :

- Un point de vue global sur la production scientifique orientée vers les méthodes d'optimisation et sur leur utilisation conduit à poser les objectifs de l'étude dans une première étape. Les grandes lignes de la méthodologie retenue introduisent alors les parties suivantes.

- Une revue rapide et synthétique, présentant l'ampleur du domaine de l'optimisation mixte non-linéaire, amène au choix des méthodes d'optimisation abordées au cours des travaux.

- Puis, une brève analyse des formulations existantes de conception d'ateliers discontinus permet de cerner un modèle particulier, qui sert de support tout au long de l'étude.

- Enfin, la démarche suivie dans la suite de ces travaux est détaillée et justifiée de manière à aboutir à la description du plan de ce mémoire.

Compte tenu de la diversité des sujets abordés, les parties bibliographiques spécifiques à chaque thème seront développées au sein des chapitres concernés.

1 – Position du problème

1.1 – Deux classes de méthodes

La complexification croissante des problèmes d'optimisation, mue par le souci de réalisme évoqué dans l'introduction, a entraîné le développement d'une grande quantité de méthodes de résolution. La globalité de ces techniques d'optimisation se divise typiquement en deux grandes classes, dont la première se compose des méthodes déterministes. Jusqu'à récemment, l'essentiel des travaux d'optimisation en Génie des Procédés était dédié à la mise en œuvre de ces dernières.

Elles se caractérisent par la garantie d'obtenir un optimum, dans la mesure où celui-ci existe. Dans le cas particulier de fonctions convexes, elles peuvent même assurer l'atteinte d'un optimum global du problème considéré. Cette performance explique grandement le succès de cette classe de méthodes. Son mode de fonctionnement repose sur une étude mathématique rigoureuse (continuité, dérivabilité…) des fonctions mises en jeu dans le problème d'optimisation. Lorsque les variables sont de nature mixte (à la fois entières et continues), les techniques consistent généralement à énumérer, si possible implicitement, l'ensemble des solutions de l'espace de recherche. Des heuristiques sont régulièrement utilisées pour stopper la recherche dans des directions selon lesquelles l'amélioration de la fonction objectif est impossible.

Cette classe présente néanmoins certains inconvénients. Tout d'abord, la mise en œuvre d'un algorithme robuste nécessite une étude approfondie des propriétés mathématiques de la classe du problème abordé. Puis, la phase méthodique d'implémentation peut s'avérer très longue à réaliser. Par ailleurs, la gamme d'applications de ces méthodes est relativement restreinte par la complexité mathématique et par la taille des problèmes (souvent NP-difficiles) étudiés.

Une grande partie de l'effort de recherche, plus spécifiquement dans les domaines de la Recherche Opérationnelle et de l'Intelligence Artificielle, est consacré depuis une vingtaine d'années à la deuxième classe de méthodes d'optimisation : les métaheuristiques. Une métaheuristique est constituée d'un ensemble de concepts fondamentaux qui permettent la conception de méthodes heuristiques pour un problème d'optimisation [HAO99]. Grâce à une inventivité remarquable, ces concepts s'appuient souvent sur une analogie avec des

phénomènes issus de domaines scientifiques radicalement différents : physiques (recuit simulé, [KIR82]), biologiques (algorithmes évolutifs, [HOL75]), sociologiques (méthode des colonies de fourmis, [WAN02]).

L'intérêt croissant porté à la classe des métaheuristiques est illustré dans [PIB99]. Les auteurs démontrent que, entre les années 1992 et 1998, les communications dédiées à la résolution de problèmes d'optimisation en Génie des Procédés se sont orientées résolument vers des méthodes telles que le recuit simulé ou les algorithmes génétiques. Le mode de fonctionnement des métaheuristiques est en effet attractif : elles consistent simplement à évaluer la fonction objectif en différents points de l'espace de recherche. La sélection de ces points repose sur la génération aléatoire de nombres réels (conférant à ces techniques leur caractère stochastique) et sur les heuristiques propres à la méthode. En exceptant le cas de problèmes contraints, les métaheuristiques ne requièrent pas d'informations complémentaires. Elles sont ainsi particulièrement séduisantes pour des problèmes impliquant des fonctions compliquées, pour lesquels un traitement des propriétés mathématiques s'avèrerait difficile, voire impossible. Par ailleurs, le fait qu'elles autorisent des détériorations (contrôlées) de la fonction objectif leur permet de s'extraire des pièges constitués par les optima locaux. Cependant, l'inconvénient principal de cette classe de méthodes tient à leur mode opératoire, qui ne leur permet pas de conclure sur la précision du résultat trouvé : l'obtention d'un optimum ne peut être garantie.

L'implémentation des algorithmes correspondants s'effectue de manière relativement simple, et leur application est facilement généralisable à n'importe quelle classe de problèmes. Plus particulièrement, les méthodes stochastiques ont permis le traitement de problèmes pour lesquels le critère est calculé au moyen d'un simulateur, sans qu'aucune expression analytique des fonctions mises en jeu ne soit explicitée.

1.2 – Problématique

Compte tenu de l'enjeu industriel qu'elles représentent, des avancées importantes ont été effectuées au sein des deux classes mentionnées ci-dessus. Les performances des méthodes ont été améliorées et la panoplie de problèmes susceptibles d'être résolus élargie. Cependant, la contrepartie de ces avancées réside sûrement dans le manque de lisibilité et de clarté dans l'utilisation qui est faite des méthodes d'optimisation. En effet, les publications mettent systématiquement en valeur la performance de la technique employée sur le problème abordé, le plus souvent sans référence à d'autres exemples ou options de résolution : l'amélioration de la meilleure solution connue au problème considéré semble être une justification suffisante,

sans qu'une quelconque explication relative au choix de la technique d'optimisation retenue soit donnée.

En fait, une méthode ne peut avoir une efficacité constante quel que soit le problème étudié. La prédiction des performances d'une méthode sur un problème particulier s'avère être une tâche particulièrement ardue et la seule certitude existante est exprimée par le théorème *No Free Lunch* [WOL97], à savoir : il n'existe pas de méthode surpassant les autres sur tous les problèmes. Cela signifie qu'une méthode spécifique et adaptée à un problème donné proposera généralement de meilleurs résultats qu'un schéma de résolution rigide et systématique. De cette incapacité à prévoir l'efficacité d'une technique d'optimisation découle le manque de lisibilité évoqué plus haut : l'utilisation d'une méthode reste le plus souvent non justifiée *a priori* et apparaît parfois même inappropriée.

Un cas simple permet d'illustrer cette tendance : un problème de conception optimale d'ateliers discontinus multiproduit, formulé d'une manière classique (qui sera d'ailleurs reprise dans cette étude), a été résolu dans plusieurs publications. Dans chacune d'entre elles, une méthode d'optimisation stochastique particulière est implémentée et un nouveau résultat est proposé. Une synthèse de ces travaux est disponible dans le tableau 1.

Référence	Méthode employée	Solution proposée [$]
Patel et al. [PAT91]	Recuit simulé	368883
Wang et al. [WAN96]	Algorithme génétique	362130
Wang et al. [WAN99]	Recherche tabou	362817
Wang et al. [WAN02]	Méthode de colonie de fourmis	368858

Tableau 1 – Résolutions d'un exemple de conception d'ateliers discontinus

Or, d'autres résultats, présentés par la suite dans ces travaux et publiés dans [PON05], ont montré qu'un optimum de meilleure qualité (356610 [$]) pouvait être obtenu en appliquant une technique de Programmation Mathématique. Ainsi, même si les solutions proposées par les méthodes stochastiques sont d'une qualité relativement bonne, leur emploi sur cet exemple ne semble pas justifié. Evidemment, sachant qu'une des caractéristiques des métaheuristiques est leur adaptabilité à de larges classes de problèmes, ces résultats peuvent être considérés comme la validation de leur bon comportement dans l'optique de les réutiliser sur des exemples plus complexes. Mais on sait bien que chaque application à un exemple particulier demande une connaissance du problème et un effort d'adaptation qui ne garantissent pas forcément l'obtention immédiate d'un bon résultat.

Cette confusion et ce manque de clarté justifient ainsi l'un des axes d'orientation de ces travaux de doctorat. L'étude se situe dans le cadre général du traitement de problèmes d'optimisation non-linéaire en variables mixtes (ou MINLP, pour Mixed Integer Non-Linear Programming). L'objectif n'est, bien entendu, ni de proposer ni de chercher à caractériser une méthode meilleure que toutes les autres dans un cas général : le théorème *No Free Lunch* cité plus haut rappelle à quel point cette entreprise serait irréaliste. Au contraire, l'objectif est de développer une méthodologie de résolution de problèmes d'optimisation de type MINLP rencontrés en Génie des Procédés. L'idée globale est donc d'évaluer les performances de différentes méthodes d'optimisation, de manière à proposer des lignes directrices permettant de guider vers le choix pertinent de l'une d'entre elles.

Il n'est évidemment pas possible de traiter cette problématique sous une forme générale, c'est-à-dire d'aborder tous les types de méthodes et tous les problèmes existants. Ainsi, l'étude se restreindra (i) à un nombre limité de méthodes d'optimisation et (ii) à une certaine classe de problèmes, i.e. une formulation relative à la conception d'ateliers discontinus. Ces deux points feront chacun l'objet d'une partie spécifique, pour expliquer les choix effectués. Cependant, des idées générales peuvent tout de même être émises dès maintenant.

Tout d'abord, concernant la formulation, deux options sont envisageables. La première est de s'orienter vers une formulation classique de Programmation Mathématique, dite analytique ou « orientée équation », et impliquant la prise en compte d'un nombre plus ou moins important de contraintes. La seconde est l'approche privilégiée dans l'équipe depuis quelques années : il s'agit de simulateurs à évènements discrets (SED). Cette approche est justifiée car elle permet de décrire plus précisément les ateliers, en prenant en compte facilement des variables décisionnelles discrètes [BER99]. Des aspects d'ordonnancement peuvent également être abordés [BAU97]. Par ailleurs, l'intégration de la connaissance du procédé peut être appréhendée en considérant des variables opératoires propres aux opérations unitaires [DIE04]. Ce mode de représentation dirige nécessairement le choix de la méthode d'optimisation vers les métaheuristiques. Ces dernières doivent donc être adaptées au traitement de problèmes de Génie des Procédés, c'est-à-dire de problèmes contraints mettant en jeu des variables mixtes. Ainsi, les problèmes de gestion des contraintes au sein de la métaheuristique et de prise en compte de la nature des variables par un codage approprié se sont posés de manière récurrente.

Relativement au choix des méthodes, il semble essentiel de considérer au moins une technique de chaque classe pour pouvoir énoncer des conclusions aussi générales que possible. Cette remarque restreint immédiatement le choix de la formulation évoqué ci-dessus puisque les méthodes déterministes ne sont pas adaptées à une fonction objectif évaluée par

un simulateur. La formulation retenue sera alors nécessairement de type analytique. La démarche adoptée permettra en outre de déterminer les procédures internes de la métaheuristique employée, qui doivent répondre pertinemment aux problèmes de gestion des contraintes et de codage des variables évoqués plus haut.

Finalement, une donnée supplémentaire entre en ligne de compte dans l'évaluation de l'efficacité des méthodes, à savoir la taille des problèmes. Le caractère fortement combinatoire de ces derniers fait en effet de la taille des instances abordées un critère fondamental duquel dépend grandement la performance des méthodes testées.

Ainsi, ces considérations préalables permettent d'établir la méthodologie permettant de répondre aux objectifs fixés. Une analyse de l'efficacité de différentes méthodes d'optimisation, issues des deux grandes classes citées auparavant, sera menée sur des problèmes à formulation identique mais caractérisés par des tailles et complexités différentes. Par ailleurs, les solutions fournies par les méthodes déterministes, *a priori* optimales, serviront de référence pour fixer les procédures de codage et de gestion des contraintes au sein de la métaheuristique. Les parties suivantes s'attachent donc à détailler d'abord le choix des méthodes d'optimisation, puis celui du modèle de conception d'ateliers discontinus, et enfin les points clés de la méthodologie.

2 – Méthodes d'optimisation

Cette partie fournit une vue d'ensemble des méthodes d'optimisation dédiées au traitement de problèmes non-linéaires en variables mixtes. Comme l'indique la figure 1 [COL02], ce type de problèmes se situe dans la zone critique de l'optimisation « difficile », à la croisée de l'optimisation combinatoire et des programmes continus non-linéaires. Par ailleurs, en considérant les méthodes de résolution, la figure illustre également sa position intermédiaire entre méthodes mathématiques et métaheuristiques.

2.1 – Méthodes déterministes

Il ne s'agit pas ici de fournir une revue détaillée de l'ensemble des méthodes et algorithmes existants, mais de donner une idée générale des tendances en terme d'optimisation mixte non-linéaire. Il est conseillé de se reporter à [GRO02], [BUS03], [BIE04] ou [FLO05] pour une présentation de l'état de l'art, des verrous scientifiques et des enjeux de recherche actuels.

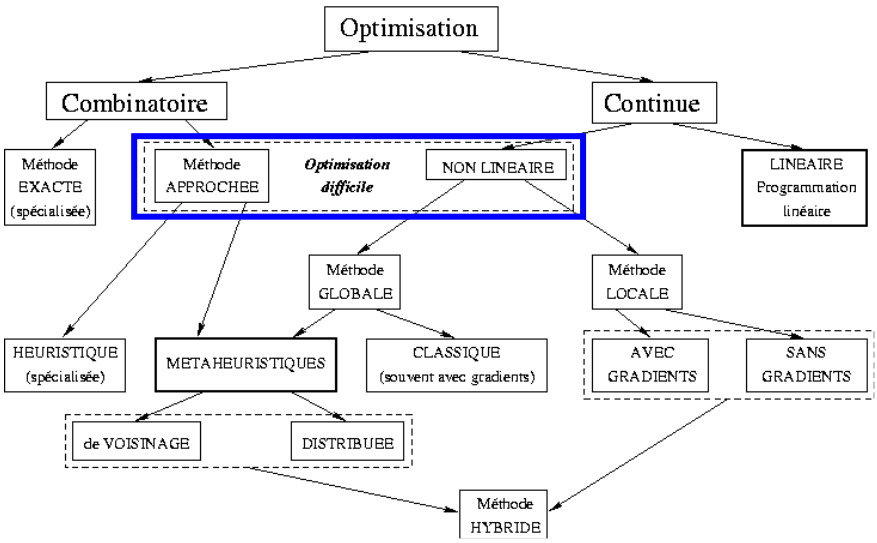


Figure 1 – Vue d'ensemble des problèmes d'optimisation d'après [COL02]

Le traitement de problèmes de type MINLP est motivé par la complexité croissante des modèles et accessible grâce à l'évolution des performances des moyens de calculs disponibles. Il passe généralement par la décomposition du problème global pour se ramener à des sous-problèmes plus simples. Les méthodes associées sont donc typiquement à chemin infaisable. La manière de décomposer le problème est propre à chaque classe de méthodes. Ainsi, soit le problème *(P)*, écrit sous sa forme classique :

$$
\begin{aligned}
& Min\ Z = f(x,y) && (P)\\
\text{soumis à}\quad & g_j(x,y) \le 0,\ \forall j = 1,\ldots, J\\
& h_k(x,y) = 0,\ \forall k = 1,\ldots, K\\
& x \in X \subseteq \mathbb{R}^n\\
& y \in Y \subseteq \mathbb{N}^m
\end{aligned}
$$

Il existe trois sous-problèmes continus non-linéaires (NLP) dérivant de cette écriture. D'après [GRO02], deux sont caractéristiques de classes de méthodes particulières, tandis que le dernier est simplement un problème de minimisation de l'infaisabilité. Le premier type de programmes NLP consiste en une relaxation continue des variables entières *y*, qui sont alors considérées comme réelles. De ce programme découle la méthode de Branch & Bound [GUP85]. Cette technique est basée sur la résolution d'une relaxation continue à chaque sommet d'une arborescence construite par séparation des variables entières et l'établissement de bornes d'autant plus étroites sur ces dernières que la recherche est avancée. Le deuxième type de sous-problèmes NLP s'écrit en fixant les variables entières *y* à chaque itération majeure de l'algorithme. Puis un programme maître linéaire mixte (MILP) est construit, généralement par la linéarisation des contraintes à la solution du programme NLP précédent :

c'est le cas de la Décomposition Généralisée de Benders [GEO72] et des méthodes d'Approximations Externes [DUR86] [YUA88].

Plus récemment, d'autres techniques, ne faisant pas appel à l'utilisation de sous-problèmes continus, ont été proposées. L'une des plus notables est celle de la théorie étendue des plans de coupe (*Extended Cutting Plane*, [WES95]), qui repose sur la résolution itérative de problèmes mixtes linéaires construits en ajoutant à chaque fois la linéarisation de la contrainte la plus sévèrement violées par la solution courante. Enfin, une dernière technique, basée sur des considérations logiques, a été proposée dans [RAM94] : l'ensemble des contraintes du problème est ainsi représenté par des sous-ensembles selon des disjonctions dictées par des variables booléennes. Des conditions logiques établissent des relations entre les ensembles disjoints définis par les variables booléennes. Dans [LEE03], ce formalisme est intégré dans une méthode de Branch & Bound.

Les méthodes citées ici ont permis d'apporter des solutions fiables à une partie des problèmes MINLP classiques. Cependant, leur gamme d'application est restreinte par l'obstacle majeur des non-convexités, très répandus dans les problèmes d'ingénierie. Ainsi, une attention particulière est accordée au traitement de problèmes MINLP non-convexes. Une grande majorité d'études s'est orientée vers les techniques de Branch & Bound spatial (i.e. les variables discrètes *et* continues peuvent être séparées dans la procédure arborescente). L'écueil principal est la possibilité que la résolution de la relaxation continue (non-convexe) du problème *(P)* évoquée ci-dessus soit piégée sur un optimum local. Les bornes générées peuvent alors être non valides et l'optimum global du problème risque d'être perdu.

Si une technique de resserrement des bornes est proposée dans [RYO95], ce sont généralement des techniques de sous-estimation convexe des fonctions initiales du problème qui sont employées pour contourner cet écueil. Cette relaxation doit être la plus fine possible de manière à accélérer la convergence des algorithmes. Des sous-estimateurs convexes ont été établis pour des termes bilinéaires, linéaires fractionnaires, ou mono-variables concaves. Cette approche est étendue dans [ADJ00] au cas de problèmes mettant en jeu des fonctions de classe C^2. Des termes quadratiques, paramétrés de manière à ce que les fonctions ainsi modifiées vérifient des conditions de convexité, sont ajoutés : c'est le αBB. Enfin dans [SMI99], une reformulation des termes non-convexes est proposée pour se ramener à des fonctions pour lesquelles des sous-estimateurs convexes sont connus.

Mais une grande partie de ces méthodes conserve un statut académique. Les algorithmes libres ou commercialisés sont bien moins nombreux. On citera surtout les modules MINLP *DICOPT++* (Approximations Externes, [VIS90]), *SBB* et *BARON* (Branch & Bound)

disponibles dans l'environnement de modélisation *GAMS* [BRO98]. Mais *αECP* (méthode *Extended Cutting Planes* [WES03]) ou *MINLP_BB* [LEY99] sont des codes qui ont également fait la preuve de leur efficacité. Un modèle de programmation disjonctive est implanté dans le module *LOGMIP* [VEC99], lui aussi disponible dans l'environnement *GAMS*. Il est conseillé de se reporter à [GRO02] ou [BUS03] pour plus de détails.

2.2 – Métaheuristiques

Une revue exhaustive ne sera pas non plus donnée ici, il est conseillé de se reporter à [HAO99] pour plus de détails. Les métaheuristiques s'orientent globalement selon deux tendances, i.e. les méthodes de voisinage et les algorithmes évolutifs.

La première classe repose, logiquement, sur la notion de voisinage, définie comme suit :

Définition : soient un ensemble X fermé et un vecteur $x = [x_1, x_1,\ldots, x_n] \in X$. Soit également une fonction d'exploration f telle que $f(x) = y$, avec $y = [y_1, y_1,\ldots, y_n] \in X$. Alors le voisinage $Y_x \subset X$ de x selon f est l'ensemble de toutes les images possibles y de x par l'application f.

Ainsi, une méthode de voisinage fonctionne typiquement en commençant par une configuration initiale, puis remplace de manière itérative la solution courante par l'un de ses voisins, de manière à faire évoluer de manière correcte la fonction objectif. Les méthodes de voisinage diffèrent d'une part par la fonction définissant le voisinage d'une solution, et d'autre part par la stratégie utilisée pour actualiser la solution courante.

Une grande variété de méthodes de voisinage a été proposée, telles que le recuit simulé, la recherche tabou, les méthodes GRASP, les algorithmes de seuil, la méthode de colonies de fourmis, etc… [HAO99]. Seuls deux exemples, les plus connus et représentatifs, seront rapidement décrits : le recuit simulé et la recherche tabou. Tout d'abord, le recuit simulé reproduit l'évolution physique d'un solide vers l'équilibre thermodynamique, par un lent refroidissement le conduisant à son état d'énergie minimal. L'analogie entre ce processus et une procédure d'optimisation est étudiée dans [KIR82] et [MET53]. Un nouvel état (ou solution) est accepté si la fonction objectif diminue ou, dans le cas contraire, selon une probabilité dépendant de l'augmentation de la fonction et de la température courante. Un paramètre analogue à la constante de Boltzmann entre également en jeu pour le calcul de cette probabilité. Ainsi, une grande variété de schémas de refroidissement est (encore actuellement) proposée [TRI05], certains d'entre eux garantissant même la convergence (sous des hypothèses cependant restrictives).

Par ailleurs, la recherche tabou considère un ensemble de voisins de la configuration courante *s* et conserve le meilleur d'entre eux, *s'*, même si ce dernier occasionne une détérioration de la fonction objectif. Une liste « tabou » des configurations déjà visitées est alors créée et actualisée de manière à éviter les cycles du type $s \longrightarrow s' \longrightarrow s$. En outre, des procédures spécifiques d'intensification ou de diversification permettent respectivement de concentrer la recherche sur les zones prometteuses ou bien au contraire de la rediriger vers des zones encore inexplorées de l'espace de recherche. Malgré des adaptations réussies au traitement d'exemples continus [TEH03] [HED06], la recherche tabou a reçu jusqu'à présent bien plus d'attention de la part de la communauté de l'optimisation combinatoire.

La seconde classe de métaheuristiques est composée des algorithmes évolutifs. Ils sont basés sur le principe de l'évolution naturelle énoncée par Darwin et impliquent trois facteurs essentiels : (i) une population de solutions au problème abordé ; (ii) une technique d'évaluation de l'adaptation des individus ; (iii) un processus d'évolution comportant des opérateurs reproduisant l'élimination de certains individus et la création de nouveaux à partir des survivants [BÄC97]. Ce dernier point amène à une amélioration de la qualité moyenne des solutions.

Les techniques les plus usitées sont les algorithmes génétiques (AG), les stratégies d'évolution (ES) et la programmation évolutive. Introduits par Holland [HOL75], les AG s'appuient fortement sur un codage universel binaire, de longueur fixe, et un ensemble d'opérateurs génétiques : croisement et mutation. Ces derniers sont définis de manière à opérer aléatoirement sur un ou deux individus. Cette méthode sera explicitée plus loin dans le détail, mais il est conseillé de se reporter à [BEA93] et [WHI93] pour des explications plus approfondies.

Concernant la seconde technique, appelée communément (n+m)-ES, l'algorithme crée m individus enfants, au moyen d'un opérateur de croisement, à partir de n parents ($n < m$). Une phase de sélection réduit ensuite la taille de population à n pour l'itération suivante. Enfin, la programmation évolutive est basée sur un codage approprié du problème étudié et sur un opérateur de mutation. Le cycle d'évolution typique de la programmation évolutive est le suivant : chaque configuration de la population courante est copiée dans la population suivante. L'opérateur de mutation est alors appliqué, créant de nouveaux individus. Puis, la population est réduite à sa taille initiale par une procédure de sélection.

2.3 – Méthodes retenues

La recherche bibliographique réalisée a mis en lumière toute la diversité des méthodes d'optimisation existantes. Les techniques adaptées au problème de conception d'ateliers discontinus puisent dans le large éventail de ces méthodes. L'objectif étant de comparer leur efficacité par rapport aux problèmes étudiés, le choix s'est orienté vers l'étude des plus significatives et habituellement employées pour leur résolution.

L'ensemble des techniques de résolution consacrées à la conception optimale d'ateliers discontinus recouvre les deux grandes classes de méthodes d'optimisation évoquées précédemment. Or, une des caractéristiques des méthodes déterministes est la garantie théorique d'obtenir un optimum. Cette assurance est d'autant plus intéressante dans l'optique de ce travail, qui intègre la comparaison des performances de différentes méthodes. La méthode déterministe fournira ainsi, en cas de convergence, un repère qui permettra d'évaluer les résultats des autres techniques utilisées.

Par contre, il est couramment admis que les méthodes déterministes se caractérisent par une certaine difficulté de convergence pour des problèmes de grande taille. La démarche adoptée ici, visant à tester des méthodes sur une panoplie d'exemples, doit permettre de déterminer le point de rupture à partir duquel la méthode déterministe devient incapable de donner un résultat en un temps de calcul raisonnable (si toutefois elle en trouve un).

A ce stade, il reste à choisir le(s) représentant(s) des méthodes déterministes utilisé(s). L'objectif des travaux n'est pas de développer une nouvelle méthode – ce qui représente un travail de recherche conséquent, ni d'implémenter une version particulière d'un algorithme déjà défini – dont la mise au point nécessite des phases longues et méthodiques. Il s'agit d'appliquer une méthode existante dont les performances ont été largement éprouvées et rapportées dans la littérature dédiée. Le choix s'est donc dirigé vers des solveurs commerciaux. C'est l'environnement de modélisation *GAMS* qui a été retenu dans la mesure où il se prêtait facilement à ces objectifs. Dans ce cadre, il a été décidé de tester les deux modules d'optimisation MINLP les plus cités dans la littérature de Génie des Procédés, à savoir *DICOPT++* et *SBB*.

Par ailleurs, un des objectifs majeurs de l'étude est également d'adapter au mieux une méthode stochastique au traitement de problèmes impliquant à la fois des variables entières et continues. Pour ce faire, le choix s'est orienté vers un Algorithme Génétique (AG), essentiellement pour deux raisons. D'une part, les AG ont montré leur efficacité dans de multiples exemples de problèmes de conception d'ateliers, notamment dans les travaux déjà

effectués dans l'équipe ([DED01] et [BER99]). D'autre part, ils représentent une option intéressante en vue d'une aide à la décision dans la mesure où ils fournissent un ensemble de bonnes solutions au praticien, libre de choisir parmi celles-ci la configuration la plus pertinente, selon des critères additionnels qui sont difficiles à formuler mathématiquement. Cette remarque est d'autant plus vraie dans le cas de l'optimisation multicritère, pour laquelle l'AG est particulièrement adapté [DIE04].

3 – Modèles de conception d'ateliers discontinus

Le but de la conception d'ateliers discontinus consiste à déterminer le nombre et les dimensions des équipements nécessaires pour chacune des opérations unitaires intervenant dans le procédé d'élaboration de différents produits, afin d'assurer un niveau de production donné, tout en minimisant un critère qui peut être d'ordre technico-économique mais aussi prendre en compte des aspects concernant la sécurité, l'environnement [AZZ05].

Une première restriction est faite, dans la mesure où l'étude se limitera aux ateliers multiproduit, c'est-à-dire à un niveau intermédiaire de flexibilité : tous les produits suivent un même cheminement dans l'atelier (ou recette opératoire). Cette distinction est faite par opposition aux ateliers multiobjectif, pour lesquels chaque produit est caractérisé par une recette de production qui lui est propre. Par ailleurs, seuls les niveaux structurels et « dimensionnement » sont impliqués ici, l'aspect ordonnancement (politiques de production, séquencement des produits) étant laissé de côté.

Une formulation générique classique a été retenue, présentant néanmoins des difficultés de résolution pour éprouver les performances des méthodes choisies. A notre connaissance, la référence la plus ancienne à ce type de modèles est présentée dans [ROB72], mais sert toujours d'exemple de validation dans des publications récentes. Le critère à minimiser est alors classiquement le coût d'investissement pour tous les équipements de l'atelier.

Le modèle initial, repris dans [GRO79], prévoyait simplement la prise en compte d'équipements discontinus et le respect d'un horizon de temps fixé pour satisfaire une demande faisant partie des données du problème. Les exemples de validation, comportant jusqu'à trois produits et quatre équipements, étaient résolus par une méthode spécifique, dite posynomiale.

Depuis, ce modèle a été repris et modifié successivement : des équipements semi-continus sont ajoutés au formalisme, dont la partie continue (tailles des équipements) est résolue par des méthodes de gradient réduit généralisé [KNO82] et [YEH87], le nombre

d'équipements étant géré par des heuristiques ; le même problème est traité dans [ESP89], avec une représentation quelque peu modifiée des coûts d'équipements, et résolu par des heuristiques basés sur une méthode du gradient. Le cas du remodelage, c'est-à-dire la modification d'un atelier déjà existant, y est également envisagé. Des exemples allant jusqu'à huit étapes opératoires sont considérés.

Les variables entières (nombres d'équipements en parallèle) sont alors réellement abordées à partir de [KOC88] avec la méthode déterministe des Approximations Externes. Le modèle est modifié pour prendre en compte des bacs de stockage intermédiaire, dont la position est fixée, dans [MOD89] qui le résolvent par heuristiques et [PAT91] avec une méthode de recuit simulé. Les tailles d'exemples atteignent alors dix étapes et quinze produits. Ce même modèle est traité dans [RAV98] grâce au code *DICOPT++* [VIS90], puis dans [WAN96], [WAN99], [WAN02] au moyen des métaheuristiques présentées dans la première partie de ce chapitre : algorithme génétique, recherche tabou et méthode de colonies de fourmis. C'est ce dernier modèle qui a été retenu ici.

Enfin, des raffinements sont apportés par la prise en considération de variables opératoires de procédé dans [MON00] et [PIN01] (résolution avec *DICOPT++*). Le problème de remodelage est envisagé dans [MON03] (traité par un programme disjonctif et le module *LOGMIP* [VEC99]). La possibilité d'incertitudes sur la demande est traitée par une représentation probabiliste (distribution normale) dans [EPP97] ou par le biais de l'arithmétique floue au moyen d'un AG multicritère dans [AGU05]. Ces degrés supplémentaires de sophistication n'ont cependant pas été envisagés dans cette étude.

4 – Démarche adoptée

L'analyse bibliographique a permis de délimiter le champ d'investigation de ce travail. Deux méthodes de Programmation Mathématique implémentées dans des solveurs de *GAMS*, ainsi qu'un algorithme génétique, ont été sélectionnés comme méthodes d'optimisation MINLP. Celles-ci seront évaluées par la suite sur un formalisme classique de Génie des Procédés.

Selon la méthodologie retenue, les performances des trois méthodes seront tout d'abord testées sur un jeu d'exemples de complexité croissantes. Cette approche permettra de déterminer, en fonction de la taille de l'instance traitée, l'adaptation de chacune des techniques considérées au traitement de problèmes mixtes formulés selon le modèle choisi.

Cette étude sera par ailleurs complétée par une modification tendant vers un réalisme accru du modèle, qui amènera à considérer les mêmes problèmes mais en variables purement entières cette fois-ci : les tailles des équipements sont en effet discrétisées. L'application des trois méthodes et l'examen de leurs résultats respectifs permettra d'élargir les conclusions tirées précédemment à un cadre plus concret.

Il est clair que les conclusions globales ne pourront être extrapolables à n'importe quelle classe de problèmes. Néanmoins, elles seront finalement testées sur un exemple de validation, caractérisé par un formalisme proche de celui employé jusqu'alors, de manière à observer si une certaine répétabilité peut être observée lorsque la modification de la formulation reste minime.

Compte tenu des ces considérations, le mémoire de thèse s'articule selon le plan suivant :

- Dans un premier chapitre, les méthodes d'optimisation retenues pour mener ces travaux sont présentées et explicitées dans le détail. Les points clés de chacun des algorithmes, aidant à la compréhension des résultats, sont précisés.

- Le deuxième chapitre est consacré à la description du modèle retenu. Le jeu d'exemples construit pour fournir le support des calculs effectués est également explicité.

- Les résultats obtenus sur les problèmes en variables mixtes, par les trois méthodes d'optimisation, sont présentés dans le troisième chapitre. Les modes de codage et de gestion des contraintes, inhérents au fonctionnement de l'AG, n'ont pas été dissociés pour respecter la logique des calculs réalisés.

- Le quatrième chapitre présente les calculs effectués avec des tailles d'équipements prenant des valeurs purement entières. Le jeu d'exemples est de nouveau traité au moyen des trois méthodes, et les résultats correspondants présentés.

- Un problème issu du domaine des bioprocédés, formulé d'une manière similaire, est traité dans le cinquième chapitre. Il sert d'exemple de validation des conclusions tirées des calculs précédents.

Finalement, les conclusions de ces travaux sont présentées et quelques perspectives de recherche proposées.

CHAPITRE 1

METHODES D'OPTIMISATION EN VARIABLES MIXTES

Ce chapitre est dédié à la présentation détaillée des méthodes d'optimisation considérées au cours de l'étude. La sélection de ces méthodes, orientée suivant la problématique qui sous-tend ces travaux, a été justifiée précédemment. Ce chapitre s'articule donc autour de leur description. Deux d'entre elles sont des techniques de Programmation Mathématique implémentées dans l'environnement de modélisation *GAMS* :

- Le solveur *DICOPT++* est une application de l'algorithme des Approximations Externes initié par Duran et Grossmann [DUR86], puis modifié plusieurs fois. L'idée directrice de la méthode sera exposée de manière à rendre compte du fonctionnement de l'algorithme. Les améliorations apportées pour surmonter certaines limitations sont également mentionnées.

- Le solveur *SBB* implique un algorithme Branch & Bound de forme classique mais adapté à la résolution de problèmes mixtes non-linéaires (MINLP par la suite). De même que pour *DICOPT++*, l'algorithme et ses points-clé seront explicités ainsi que certaines options de fonctionnement.

Le mode opératoire de *DICOPT++* repose sur des modules de résolution de problèmes continus non-linéaires (NLP) et linéaires en variables mixtes (MILP), respectivement *CONOPT3* et *CPLEX*. *SBB* utilise uniquement le module *CONOPT3* pour la résolution des problèmes NLP. Ces solveurs étant insérés dans la boucle d'optimisation MINLP, les grandes lignes de leur fonctionnement seront décrites brièvement.

La troisième technique choisie, de type stochastique, est un algorithme génétique. Les grands principes de la méthode seront exposés et ses paramètres généraux seront présentés. Enfin, les opérateurs régissant ses performances seront évoqués. Essentiels à l'efficacité de la résolution, ceux-ci feront l'objet d'études particulières dans les chapitres suivants.

1 – Techniques de Programmation Mathématique

Les techniques de Programmation Mathématique utilisées ici sont implémentées dans des solveurs intégrés au sein de l'environnement de modélisation commercial *GAMS*. Par conséquent, l'accès au code des algorithmes et leur modification sont impossibles. Il est important de préciser que de nombreux points de détails de leur implémentation pratique restent dans l'ombre. Mais étant donné que les algorithmes mis en œuvre ont été développés par des organismes indépendants (Université de Carnegie Mellon, pour *DICOPT++* et ARKI Consulting & Development pour *SBB*), ils ont fait l'objet de publications scientifiques

ouvertes à l'ensemble de la communauté. Leurs principes généraux sont donc facilement disponibles et peuvent donc être décrits dans cette partie.

1.1 – Tendances communes

Les deux méthodes de Programmation Mathématique (MP) choisies sont basées sur des hypothèses, des modes d'opération et des philosophies radicalement différents. Mais, les deux algorithmes de résolution de problèmes de type MINLP ont des caractéristiques communes, dont les principales sont la formulation du modèle, la décomposition du problème et, enfin, des tendances en termes de résolution.

Tout d'abord, concernant la formulation du problème, l'environnement *GAMS* offre un langage de modélisation convivial et relativement facile à comprendre. Il est basé sur la création d'ensembles sur lesquels sont définies les variables. Si les équations s'écrivent pratiquement de façon naturelle, une attention toute particulière doit être accordée à la formulation du modèle, car les équations sont prises en compte par le solveur comme des contraintes. Il en résulte qu'une formulation bien ou peu adaptée aidera ou empêchera la convergence. Par exemple, les contraintes de type égalité sont explicitement plus sévères que les contraintes inégalité et il est judicieux de tenter de remplacer les premières par les secondes. Ainsi, un effort doit être porté à la formulation du problème de manière à faciliter ensuite la mise en oeuvre du module de résolution. De nombreuses techniques, telles que celle du *Grand M*, l'introduction de variables d'écart, la formulation préférentielle par inégalités,… sont utilisées dans cette optique.

Par ailleurs, quel que soit le mode d'opération de la méthode MP, il est basé sur la décomposition du problème. En effet, s'il est vrai que les deux obstacles que sont le traitement de variables mixtes et celui de fonctions non-linéaires ont été résolus séparément, leur prise en compte simultanée est très complexe. Qui plus est, elle n'a pas pu être traitée de manière générale. Le fonctionnement des algorithmes MINLP passe ainsi toujours par la décomposition en un sous-problème continu NLP, à part du traitement des variables entières. Cette dernière partie est alors traitée au moyen d'algorithmes de résolution MILP. Cette approche peut être perçue comme un découplage des difficultés.

Enfin, des tendances propres à la résolution sont communes à toutes les méthodes mathématiques. Tout d'abord, la vérification de propriétés mathématiques telles que différentiabilité et continuité sont nécessaires pour leur mise en oeuvre. Par ailleurs, une convergence efficace est conditionnée par la convexité des fonctions. En programmation non-linéaire (mixte ou non), cette dernière propriété est très importante dans la mesure où, si elle

n'est pas vérifiée, elle fausse la garantie de globalité de l'optimum trouvé. En revanche, si toutes les fonctions mises en jeu dans le modèle sont convexes, l'optimum global est garanti.

Les concepts énoncés ci-dessus étant valables pour toutes les méthodes mathématiques, celles qui ont été utilisées dans le cadre de ces travaux peuvent être maintenant présentées individuellement.

1.2 – DICOPT++

La première technique de Programmation Mathématique utilisée dans cette étude est le solveur *DICOPT++*, implémenté au sein de l'environnement de modélisation *GAMS*. La version utilisée date de 1990, et repose sur l'algorithme dit OA/ER/AP (Outer Approximation / Equality Relaxation / Augmented Penalty) développé par Viswanathan et Grossmann [VIS90]. Mais elle n'est que la forme finale de l'algorithme dit des Approximations Externes, dont les premières bases furent posées par Duran et Grossmann [DUR86]. Une variante du même algorithme initial a été également proposée dans [YUA88].

On s'attachera, donc, dans cette partie, à expliciter tout d'abord les bases théoriques de la méthode des Approximations Externes, dont découle le premier algorithme OA [DUR86]. Puis, les améliorations successives ayant conduit à la version finale OA/ER/AP seront présentées. Une fois le critère d'arrêt détaillé, il sera possible de décrire l'algorithme implémenté dans *DICOPT++*.

1.2.1 – Principe des Approximations Externes

La méthode des Approximations Externes est une méthode dite à chemin irréalisable, dans la mesure où l'optimum est approché par l'extérieur de l'espace faisable. D'un point de vue conceptuel, elle repose sur l'hypothèse que l'espace faisable des solutions d'un problème d'optimisation peut être représenté comme une suite infinie d'ensembles. Dans le cas où cet ensemble des solutions faisables est convexe, il peut être défini comme l'intersection de demi-espaces. Ces demi-espaces reposent sur les hyperplans constitués par les linéarisations des contraintes en des points particuliers. Si ce nombre de points tend vers l'infini, l'espace faisable est décrit complètement. Ceci est illustré par la figure 1 pour un problème à deux variables, où les contraintes g_j délimitent l'espace faisable F. Leur linéarisation aux points x^i sont les hyperplans H_i (réduits ici à des droites) et représentent bien une approximation externe de F.

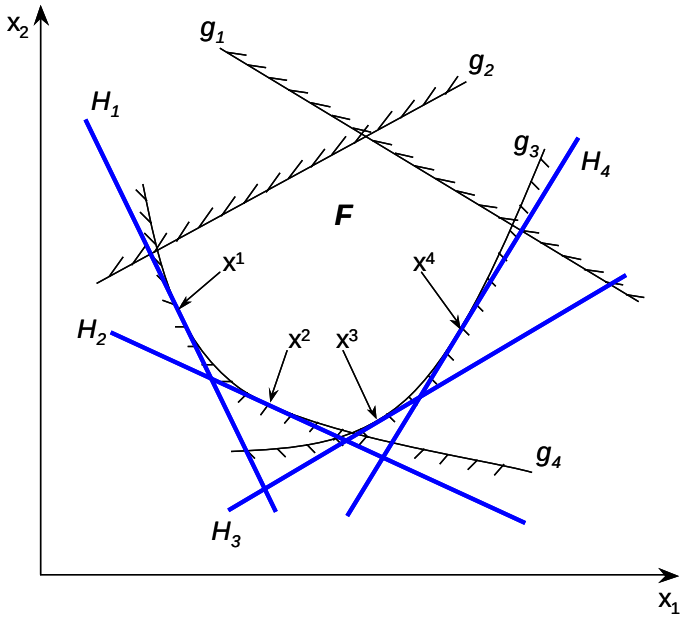


Figure 1 - Approximations externes d'un ensemble convexe

L'idée de base de la méthode des Approximations Externes est ainsi restreinte par l'hypothèse que la fonction objectif et les contraintes impliquant des variables continues sont des fonctions convexes. Par ailleurs, le problème étudié doit respecter le fait que les variables entières ne peuvent être impliquées que dans des fonction linéaires. Cependant, si une variable entière y est impliquée dans un terme non-linéaire, il est parfois possible de la remplacer par une variable continue x et d'ajouter la contrainte, bien linéaire cette fois, $y = x$. La forme générale du problème d'optimisation *(P0)*, de type MINLP, s'écrit alors :

$$\begin{aligned} & Min\ Z = c^T y + f(x) && (P0) \\ soumis\ à\quad & g(x) + By \leq 0 \\ & x \in X \subseteq \mathfrak{R}^n \\ & y \in Y \subseteq \mathbb{N}^m \end{aligned}$$

où f et g sont des fonctions continûment différentiables et convexes sur X, c est un vecteur de dimension m et B est une matrice de dimension $m \times p$ où p est le nombre de contraintes inégalité. La formulation ci-dessus met également en évidence que *(P0)* est, sous sa forme classique, un problème à variables x et y séparables, même si en pratique cette condition n'est pas nécessaire.

Remarque : par la suite et dans le reste de ce mémoire, on se restreindra au cas d'une minimisation.

La méthode des approximations externes est appliquée au problème *(P0)*, uniquement à l'espace faisable défini par les variables continues. Cette transformation concerne les contraintes et la fonction objectif, et équivaut respectivement, vu l'hypothèse de convexité, à une *sur-estimation* de la région faisable et une *sous-estimation* linéaire de *f* [FLO95]. On obtient donc le problème *(P1)* suivant, équivalent à *(P0)* :

$$
\begin{aligned}
& Min\ Z = c^T y + \mu && (P1)\\
soumis\ à\quad & f(x^i) + \nabla f(x^i).(x - x^i) - \mu \le 0,\ \forall x^i \in X\\
& g(x^i) + \nabla g(x^i).(x - x^i) + By \le 0,\ \forall x^i \in X\\
& x \in X \subseteq \Re^n\\
& y \in Y \subseteq \aleph^m\\
& f_L \le \mu \le f_U
\end{aligned}
$$

f_L et f_U sont les solutions respectives des problèmes continus non contraints *{Min f(x), x∈X}* et *{Max f(x), x∈X}*. Il est important de noter que *(P1)* est un problème de type MILP, ce qui prouve que la difficulté du problème a été réduite.

Par contre, *(P1)* représente une infinité de sous-problèmes, puisqu'il est équivalent à *(P0)* si et seulement si x^i décrit tout son espace de définition *X*. Mais Duran et Grossmann [DUR86] prouvent que la projection de *(P0)* sur l'espace des variables discrètes est équivalente à *(P0)*, ce qui entraîne que le nombre de sous-problèmes mis en jeu dans *(P1)* est fini et égal à la combinatoire rapportée aux variables discrètes.

Ils montrent également que pour chaque y^i considéré, le point x^i auquel la linéarisation doit être effectuée est la solution optimale du sous-problème continu non-linéaire suivant, paramétré par y^i :

$$
\begin{aligned}
& Min\ z(y^i) = c^T y^i + f(x) && (S(y^i))\\
soumis\ à\quad & g(x) + B\, y^i \le 0\\
& x \in X \subseteq \Re^n
\end{aligned}
$$

Le problème maître *(P1)* ne doit donc être résolu qu'au moyen d'un nombre fini de linéarisations, qui peut cependant être très important. Pour contourner la difficulté due au fait qu'il faudrait déterminer par anticipation les approximations externes correspondant à chaque valeur possible de *y*, ce problème est relaxé en ne prenant en compte que les linéarisations déjà calculées lors des itérations précédentes.

Finalement, le problème maître de type MILP à l'itération *k* est alors :

$$
\begin{aligned}
& Min\ Z = c^T y + \mu && (M^k)\\
soumis\ à\quad & f(x^i) + \nabla f(x^i).(x - x^i) - \mu \le 0,\ \forall i = 1,\ldots,k\\
& g(x^i) + \nabla g(x^i).(x - x^i) + By \le 0,\ \forall i = 1,\ldots,k\\
& x \in X \subseteq \mathbb{R}^n\\
& y \in Y \subseteq \mathbb{N}^m\\
& f_L \le \mu \le f_U
\end{aligned}
$$

Duran et Grossmann [DUR86] déduisent finalement le théorème suivant :

« *(x*, y*)* est la solution optimale de *(P0)* si et seulement si *(x*, y*)* est optimale pour le problème maître (M^k), avec $\mu^* = f(x^*)$ ».

Ceci signifie que la solution optimale de *(P0)* est (x^k, y^k), où y^k est la solution de (M^k) à l'itération *k* et x^k la solution du sous-problème $(S(y^k))$ correspondant. Il a été montré que, sous les hypothèses de convexité énoncées plus haut, ce nombre d'itérations *k* est fini.

Les grandes lignes du fonctionnement de l'algorithme découlent logiquement des propriétés mentionnées ci-dessus : il consiste en une suite de séquences composées d'un problème de type NLP et d'un problème de type MILP, résolus à tour de rôle. Cependant, de par la sévérité des hypothèses formulées, l'algorithme OA se heurte à des difficultés de résolution l'empêchant de garantir l'obtention de l'optimum global dans de nombreux cas. Les améliorations dont il a fait l'objet par la suite sont exposées dans la partie suivante.

1.2.2 – Evolution de l'algorithme initial

Grossièrement, trois types d'inconvénients majeurs faussent la résolution d'un problème MINLP par l'algorithme des Approximations Externes tel qu'il a été défini précédemment. Le premier provient de la non-linéarité des sous-problèmes $(S(y^i))$, qui en conséquence peuvent se révéler infaisables.

Les deux autres, étant dus aux hypothèses émises dans la première version de l'algorithme de Duran et Grossmann [DUR86], sont corrigés dans les versions suivantes : ils concernent, d'une part, la gestion de contraintes égalité impliquant des termes non-linéaires, et d'autre part, le traitement de problèmes non-convexes.

1.2.2.1 – Sous-problèmes infaisables

Soit le programme (M^k) la relaxation du problème maître initial à l'itération k, il est possible que le sous-problème continu qui lui soit associé $(S(y^k))$ soit infaisable. Mais une pseudo-solution x^k au sous-problème $(S(y^k))$ peut néanmoins être obtenue en résolvant le problème continu non-linéaire suivant :

$$\begin{aligned} & Min \sum_j a_j && \text{(NLP-FEAS)} \\ \text{soumis à} \quad & g(x) + By^k \leq a \\ & a \geq 0 \end{aligned}$$

On peut donc calculer les approximations externes associées à cette solution :

$$f(x^k) + \nabla f(x^k).(x - x^k) - \mu \leq 0 \qquad g(x^k) + \nabla g(x^k).(x - x^k) + By \leq 0$$

Ces approximations sont tout de même valides pour éliminer une région de l'espace infaisable et il est possible de les ajouter à l'ensemble des linéarisations définissant le problème maître. Par contre, il est également nécessaire de faire une *integer cut* pour s'assurer que la variable entière y^k soit supprimée de l'ensemble faisable et ne pourra plus constituer une solution acceptable.

1.2.2.2 – Contraintes égalité non-linéaires

Compte tenu de la formulation de la méthode des approximations externes, il est clair qu'il est impossible de traiter le cas de contraintes égalité non-linéaires. La correction de ce point a été l'objet des travaux de Kocis et Grossmann [KOC87]. Le nouveau problème s'écrit sous la forme :

$$\begin{aligned} & Min\ Z = c^T y + f(x) && \text{(P-ER)} \\ \text{soumis à} \quad & h(x) + Ay = 0 \\ & g(x) + By \leq 0 \\ & x \in X \subseteq \Re^n \\ & y \in Y \subseteq \aleph^m \end{aligned}$$

Les hypothèses de convexité de f et g sont toujours valides, et en plus, h est supposée quasi-convexe, i.e. $\forall (x,x') \in \Re^n x \Re^n$, $\forall \lambda \in [0,1]$, $h[\lambda x+(1-\lambda)x'] \leq max[h(x),h(x')]$. En appliquant les conditions de Karush-Kuhn-Tucker au sous-problème NLP associé à une

valeur y^k, on détermine les multiplicateurs de Lagrange λ_i [FLO95] et on peut construire une matrice diagonale T dont les éléments diagonaux t_{ii} sont tels que :

$$t_{ii} = \left\{ \begin{array}{lll} -1 & \text{si} & \lambda_i < 0 \\ +1 & \text{si} & \lambda_i > 0 \\ 0 & \text{si} & \lambda_i = 0 \end{array} \right\}$$

Sous les hypothèses évoquées plus haut, Kocis et Grossmann [KOC88] montrent alors qu'il est possible de relaxer la contrainte égalité $h(x) + Ay = 0$ en $T[h(x) + Ay] \leq 0$. Cette expression n'est rien d'autre qu'un nouvel ensemble d'inégalités non-linéaires, traitées, on l'a vu, de manière efficace par l'algorithme des Approximations Externes initial. Le nouvel algorithme s'appelle alors l'algorithme des Approximations Externes / Relaxation des Egalités (OA/ER). Mais cette méthode reste fortement assujettie à des hypothèses de convexité ou quasi-convexité, qui sont peu fréquentes dans les formulations habituelles de problèmes d'optimisation réalistes. Le traitement de ce dernier écueil est explicité dans la partie suivante.

1.2.2.3 – Traitement de fonctions non-convexes

En effet, si les hypothèses de convexité sur lesquelles repose le mode d'opération de l'algorithme OA/ER ne sont pas vérifiées, ce dernier risque de se heurter à deux types de problèmes :

- le solveur NLP résolvant le sous-problème continu peut se trouver piégé dans un optimum local ;

- les linéarisations des contraintes pour la construction du problème maître MILP peuvent ne pas être valides. En effet, si l'espace faisable n'est pas convexe, l'approximation *a priori* externe peut en couper certaines parties et faire perdre l'optimum global. La figure 2 illustre ce comportement : la contrainte g_3 est non-convexe sur X et la linéarisation H_2 calculée en x^2 empiète sur l'espace faisable F.

Kocis et Grossmann ([KOC88] et [KOC89]) ont développé une stratégie en deux phases pour pallier à ces difficultés. Elle consiste à appliquer initialement l'algorithme OA/ER, puis, dans un deuxième temps, à effectuer des tests locaux et globaux pour tenter de détecter des non-convexités pouvant expliquer pourquoi certaines approximations externes ne sont pas valides. Les tests du premier type vérifient tout d'abord la convexité au voisinage des points x^k ($k=1,\ldots,K$ itérations) auxquels ont été effectuées les linéarisations des contraintes et de la

fonction objectif : cela revient à s'assurer localement que les fonctions considérées sont bien au-dessus de leur linéarisation.

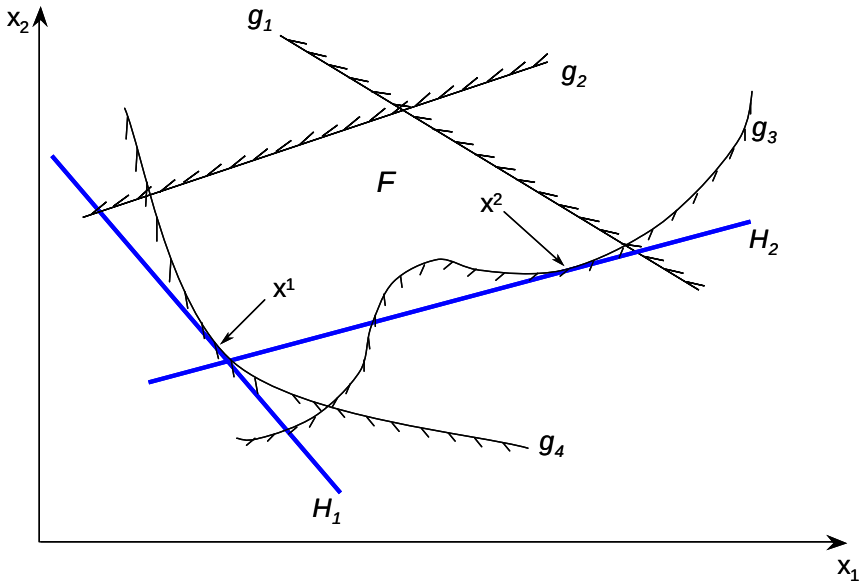


Figure 2 - Approximations externes non valides

Les tests globaux cherchent à déterminer, parmi les linéarisations ajoutées successivement au problème maître, celles qui coupent manifestement l'espace faisable. Les points x^k sur lesquels sont calculées les linéarisations sont des bornes supérieures du MINLP initial *(P0)*. Cela signifie qu'ils appartiennent à l'espace faisable initial et respectent logiquement la relaxation linéaire des contraintes et de la fonction objectif si ces dernières sont bien externes. Par conséquent, il suffit de vérifier, pour chaque itération $k \in \{1,\ldots,K\}$, que les approximations externes f^{l}_{LIN} et g^{l}_{LIN} calculées aux itérations $l \neq k$ sont effectivement valides en x^k, ce qui s'écrit :

$$\forall (k,l) \in \{1,\ldots,K\}^2,\ l \neq k,\ g(x^k) \geq g^{l}_{LIN}(x^k) \text{ et } f(x^k) \geq f^{l}_{LIN}(x^k) \qquad (1)$$

Si l'un des tests est positif, un nouveau problème maître est formulé en modifiant les linéarisations ayant été identifiées comme non valides, puis l'algorithme est exécuté de nouveau. Cette méthode n'est pas explicitée plus en détails ici car elle n'est pas implémentée dans *DICOPT++*. En effet, c'est la technique dite de la Pénalité Augmentée qui gère, avec plus de succès, les problèmes de non-convexité fréquemment rencontrés.

L'idée de Viswanathan et Grossmann [VIS90] est de permettre une certaine violation des contraintes lors de la recherche d'une nouvelle valeur de y par la résolution du MILP. La violation est permise par l'introduction de variables d'écart (*slack variables*) s^0, p et q. Par ailleurs, un terme de pénalité est ajouté dans la fonction objectif du problème maître,

représentant la violation de chaque contrainte. Cette violation est pondérée par des poids assez lourds de manière à faire tendre prioritairement les variables d'écart s^0, p et q vers zéro. Ainsi, ce mode de fonctionnement permet de retrouver les résultats du MILP initial lorsque la convexité est vérifiée. Le nouveau problème maître à l'itération k s'écrit alors :

$$Min\ Z = c^T y + \mu + w^0 s^0 + \sum_{jp} w_{jp} p_{jp} + \sum_{jq} w_{jq} q_{jq} \qquad (M^k\text{-}PA)$$

$$\text{soumis à} \quad f(x^i) + \nabla f(x^i).(x - x^i) - \mu \le s^0, \ \forall i = 1,\ldots,k$$

$$T[h(x^i) + \nabla h(x^i).(x - x^i) + Ay] \le p, \ \forall i = 1,\ldots,k$$

$$g(x^i) + \nabla g(x^i).(x - x^i) + By \le q, \ \forall i = 1,\ldots,k$$

$$x \in X \subseteq \mathbb{R}^n$$

$$y \in Y \subseteq \mathbb{N}^m$$

$$s^0 \ge 0,\ p \ge 0,\ q \ge 0$$

Cette méthode revient globalement à une extension de l'approximation de la région faisable, et réduit par conséquent les risques de couper une partie de la région faisable par des linéarisations non valides. La figure 3, reprenant l'exemple de la figure 2, met en évidence ce mode de fonctionnement : la linéarisation non valide H_2 est repoussée pour devenir H_2' qui elle, par contre, réalise bien une approximation externe de F. La valeur α représente la valeur de la variable d'écart q.

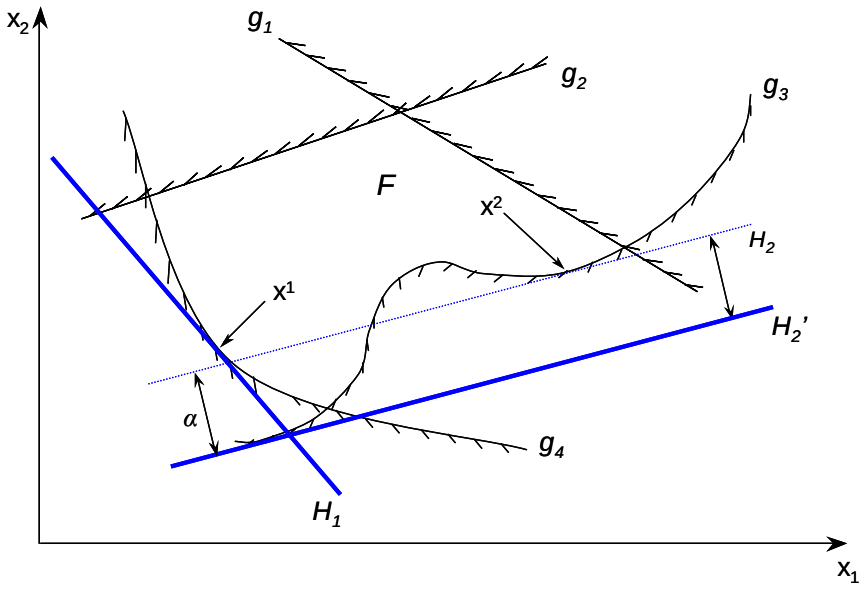


Figure 3 - Approximations externes avec méthode de Pénalité Augmentée

L'efficacité globale de la méthode proposée pour dépasser les difficultés posées par les non-convexités a été prouvée sur différents exemples-tests [VIS90]. Malgré cela, il est cependant important de noter que l'algorithme ne peut pas toujours assurer l'obtention de l'optimum global. En effet, l'évidente faiblesse de la méthode de Pénalité Augmentée est le

choix adéquat des valeurs initiales des variables d'écart s^0, p et q. Si celui-ci est mal effectué, les relaxations linéaires des contraintes, même modifiées par la Pénalité Augmentée, peuvent amener à couper l'espace faisable et ainsi éventuellement l'optimum global. Par ailleurs, si les sous-problèmes pour des variables entières fixées présentent plusieurs optima locaux, l'algorithme peut être piégé sur une solution locale. Les bornes calculées ne sont alors plus valides.

La méthode ayant été définie et ses améliorations présentées dans ce qui précède, l'algorithme OA/ER/AP final est décrit dans la partie suivante.

1.2.3 – Algorithme OA/ER/AP

Comme cela a été démontré plus haut, l'algorithme consiste à découpler le problème en un sous-problème de type NLP pour lequel les variables discrètes sont fixées et un problème maître MILP. Dans le cas du sous-problème continu, les valeurs des variables discrètes sont égales à celles de la solution optimale du problème maître précédemment traité. Sa résolution fournit une solution faisable, constituant une borne supérieure Z_U. Cette borne supérieure évolue de manière monotone décroissante (au sens large). Si le critère d'arrêt n'est pas vérifié, les contraintes sont alors linéarisées à la solution optimale du problème NLP. Le problème maître à l'itération k est alors défini par les linéarisations des contraintes construites à toutes les itérations i précédentes, i.e. $i = 1,\ldots, k$. Puis le problème MILP est résolu. Dans cette phase, les valeurs d'initialisation des variables continues sont celles de la solution du NLP précédent. La solution du MILP est une borne inférieure Z_L et de nouvelles valeurs discrètes sont fournies pour paramétrer le sous-problème continu suivant. Puisqu'à chaque itération un nouvel ensemble de contraintes linéarisées est ajouté au problème maître, l'approximation linéaire de la région faisable s'améliore. La borne inférieure calculée croît ainsi de manière monotone avec le nombre d'itérations. Par ailleurs, des *integer cuts* sont imposées sur le jeu de variables discrètes de manière à éviter de retomber plusieurs fois sur le même au cours de la recherche.

Remarque : la résolution des problèmes NLP et MILP nécessite l'utilisation de solveurs, i.e. *CONOPT3* et *CPLEX* [BRO98] respectivement, qui seront présentés plus tard. L'objectif ici est seulement de présenter en détail la méthode MINLP.

Concernant la première itération, l'algorithme débute en résolvant le problème MINLP relaxé (RMINLP). Celui-ci, défini comme étant la relaxation continue du problème initial, est donc de type NLP. Si une solution mixte (x^0, y^0) est trouvée (tous les y^0 ont des valeurs entières) l'algorithme s'arrête, sinon il continue par la linéarisation des contraintes en x^0 et la

résolution du premier problème maître. Cette démarche présente l'avantage de ne pas nécessiter de valeurs d'initialisation pour les variables discrètes. En outre, l'expérience a montré que généralement, la solution du premier problème MILP fournit une bonne estimation des variables continues, et donc une première approximation externe du problème MINLP de bonne qualité.

La représentation finale de l'algorithme OA/ER/AP est donnée dans la figure 4.

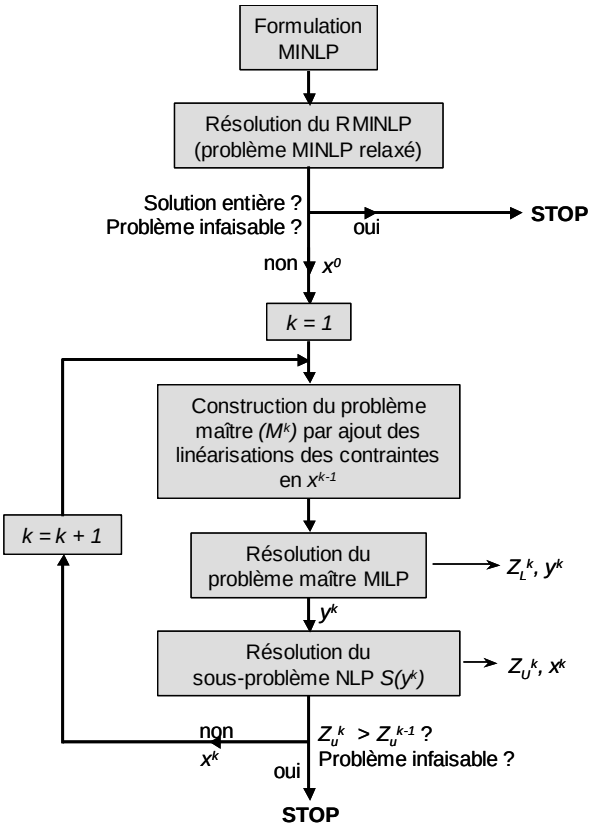


Figure 4 – Organigramme de l'algorithme OA/ER/AP

1.2.4 – Critère d'arrêt

A priori, le critère d'arrêt est censé s'inscrire dans la logique du déroulement de l'algorithme. Duran et Grossmann [DUR86] tirent parti des propriétés des bornes calculées par le sous-problème continu et le problème maître discret, i.e. décroissance monotone de la borne supérieure Z_U et croissance monotone de la borne inférieure Z_L respectivement. Ils en déduisent que la convergence est atteinte lorsque ces deux bornes se rencontrent en-dessous d'une certaine précision, c'est à dire dès que $|Z_L - Z_U| \leq \varepsilon$.

Mais il a été montré qu'à cause des non-convexités, le problème MILP ne produira généralement pas des bornes inférieures valides. Le critère d'arrêt proposé dans [VIS90]

repose alors sur une heuristique fondée sur l'observation du comportement de l'algorithme sur des problèmes non-convexes et convexes, c'est-à-dire pour lesquels le critère de convergence théorique énoncé précédemment est applicable. Viswanathan et Grossmann [VIS90] remarquent en effet que la détérioration de la fonction objectif, i.e. la borne supérieure, entre deux problèmes NLP faisables consécutifs indique l'atteinte de l'optimum global. Ainsi, le critère d'arrêt utilisé dans *DICOPT++* est : $Z_U(y^k) > Z_U(y^{k-1})$.

Remarque : le critère d'arrêt proposé dans [YUA88] repose sur le théorème affirmant que si une solution en variables entières y^k du problème maître discret est identique à une solution y^p rencontrée précédemment ($1 \leq p < k$), alors la solution optimale du problème *(P0)* est atteinte.

1.3 – SBB

Le module de résolution *SBB* (pour Simple Branch & Bound) est plus récent que *DICOPT++*, puisque sa première implémentation date d'octobre 2000. Son principe repose sur la combinaison d'un algorithme Branch & Bound classiquement utilisé pour la résolution de problèmes MILP et d'un solveur de type NLP.

Les principes généraux de l'algorithme seront explicités dans un premier temps. Les options offertes à l'utilisateur, lui permettant de jouer sur des paramètres guidant la recherche, seront ensuite présentées. Enfin, des éléments théoriques de comparaison entre *DICOPT++* et *SBB* seront rapidement développés.

1.3.1 – Description de l'algorithme

L'idée directrice de l'algorithme de *SBB* est une reproduction du fonctionnement d'un algorithme Branch & Bound, habituellement dédié au traitement de problèmes de type MILP. La différence repose sur la résolution, à chaque sommet du Branch & Bound, d'un sous-problème continu non-linéaire au moyen d'un solveur NLP de *GAMS*.

1.3.1.1 – Branch & Bound

L'algorithme de Branch & Bound est une technique dite constructive dans la mesure où la solution est élaborée variable par variable, au fur et à mesure de l'avancée de la recherche. Son principe repose sur trois notions distinctes : séparation du problème principal, relaxation des sous-problèmes et troncature de l'arbre de recherche [FLO95].

- *Séparation* :
  Considérons le problème *(P)* de type MILP dont l'espace faisable est noté *FS(P)*. L'ensemble de sous-problèmes $(P_1), (P_2),\ldots, (P_n)$ est une séparation de *(P)* si :

$$\bigcup_i FS(P_i) = FS(P) \text{ et } \forall i,j \text{ avec } i \neq j,\ FS(P_i) \cap FS(P_j) = \varnothing .$$

  Ceci est le plus souvent réalisé en rajoutant des contraintes contradictoires sur des variables entières. Par exemple, pour une variable binaire *y* impliquée dans *(P)*, les sous-problèmes (P_1) et (P_2) sont respectivement définis par les contraintes supplémentaires $y = 0$ et $y = 1$.

- *Relaxation*
  La relaxation *(RP)* du problème *(P)* a un espace faisable contenant celui de *(P)* ce qui implique que (i) si *(RP)* est infaisable, alors *(P)* l'est aussi ; (ii) la solution optimale de *(RP)* est une borne inférieure de la solution optimale de *(P)* ; (iii) une solution optimale de *(RP)* et faisable pour *(P)* est une solution optimale de *(P)*.
  Le plus souvent la forme de relaxation considérée est la relaxation de la contrainte d'intégralité sur les variables entières. La relaxation de *(P)* est alors un problème de type Linear Programming.

- *Troncature*
  Soit (P_k) un sous-problème de *(P)* susceptible de mener à la solution optimale de *(P)*. Trois critères peuvent autoriser à tronquer (P_k) :
  - la relaxation LP de (P_k) est infaisable ;
  - la solution optimale de la relaxation de (P_k) est supérieure à la meilleure solution faisable trouvée antérieurement (borne supérieure) ;
  - la solution optimale de la relaxation de (P_k) est faisable, donc optimale pour (P_k). Si en plus elle est faisable pour *(P)* et inférieure à la borne supérieure, alors cette dernière peut être actualisée.

L'idée générale de l'algorithme de Branch & Bound est que chaque sommet représente un sous-problème MILP ayant des bornes propres sur les variables entières. Sa relaxation continue, si elle n'est pas infaisable, est résolue. Ceci fournit une borne inférieure du sous-problème, qui peut éventuellement être une borne inférieure du problème initial. Si cette borne inférieure est plus grande que la meilleure solution faisable trouvée jusque là, le sommet est tronqué. Si la solution est faisable, i.e. respecte la contrainte d'intégralité pour le vecteur *y*, la borne supérieure de *(P)* est éventuellement actualisée. On obtient ainsi, au cours de la recherche, une borne inférieure croissante et une borne supérieure décroissante (au sens large dans les deux cas). Puis, une variable discrète pour laquelle la solution du sous-

problème relaxé est non entière est choisie ou « séparée ». Les deux valeurs entières les plus proches de la solution réelle pour cette variable représentent alors de nouvelles bornes pour deux nouveaux sommets « enfants ». Ce fonctionnement est représenté sur la figure 5 où une variable entière y_k est séparée à partir du sommet racine pour former les sommets 1-1 et 1-2.

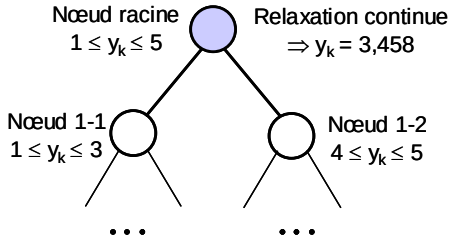


Figure 5 – Exemple de séparation

Les deux points-clé de l'implémentation de la méthode de Branch & Bound sont la sélection de la variable à séparer pour créer deux sommets enfants et du prochain sommet à résoudre. Il n'y a pas vraiment de méthode robuste et universelle pour répondre à la première de ces questions, c'est donc le plus souvent l'utilisateur qui établit une liste de priorités. Quelques alternatives existent cependant, comme la méthode des « pseudo-coûts » (disponible dans les options de *SBB*). En revanche, diverses techniques classiques sont utilisées pour le choix du prochain sommet à traiter : en profondeur d'abord, en largeur d'abord, etc… Certaines d'entre elles sont décrites dans la partie 1.3.2, traitant des options de *SBB*. Il n'existe cependant pas, parmi ces différentes stratégies de choix du sommet suivant, de meilleure méthode. Un compromis est à faire pour profiter des avantages respectifs de chacune d'entre elles.

1.3.1.2 – Algorithme de *SBB*

SBB fonctionne comme un Branch & Bound classique, à la différence que les problèmes relaxés à chaque sommet sont de type NLP. Leur résolution s'effectue au moyen d'un solveur NLP, *CONOPT3* dans notre cas. Le point négatif est, comme toujours, le traitement de fonctions non-convexes. En effet, celles-ci peuvent, au niveau de la résolution du problème continu NLP, amener le module à rester bloqué sur un optimum local. Par conséquent, la borne inférieure fournie n'est pas valide et une branche de l'arbre de recherche peut ainsi être tronquée par erreur, faisant perdre l'optimum global.

De la même manière que *DICOPT++*, *SBB* commence par résoudre la relaxation continue du problème initial (ou sommet racine). Si elle est infaisable, *SBB* s'arrête. Si toutes les variables discrètes prennent des valeurs entières, *SBB* retourne le résultat trouvé comme

étant la solution optimale. Dans les autres cas, la mise en œuvre de la procédure arborescente est amorcée.

Pour chaque traitement de la relaxation continue d'un sommet, les sommets enfants sont placés dans une liste de sommets. Cette liste est actualisée à chaque itération. C'est le cas, par exemple, lorsqu'une nouvelle borne supérieure est générée : les sommets de la liste présentant une borne inférieure plus grande que cette nouvelle valeur sont éliminés. L'algorithme s'arrête alors lorsque la liste de sommets est vide. La représentation schématique de l'algorithme mis en œuvre dans *SBB* apparaît sur la figure 6.

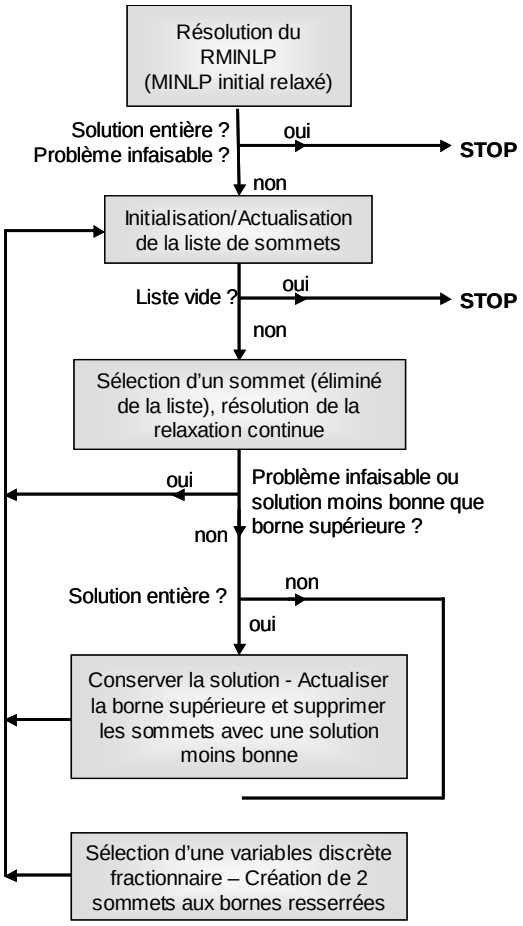


Figure 6 – Organigramme de l'algorithme *SBB*

1.3.2 – Options

Les deux points clés évoqués pour l'implémentation d'un Branch & Bound classique, i.e. le choix de la variable à séparer et du prochain sommet à résoudre, valent aussi pour *SBB*. Concernant le choix de la prochaine variable à séparer, *SBB* propose une stratégie sophistiquée, la méthode des « pseudo-coûts ». Elle consiste globalement à calculer, à un sommet donné, l'effet sur la fonction objectif si l'on fait bouger une variable discrète ayant

une valeur réelle vers l'entier le plus proche. Cette démarche revient à affecter un pseudo-coût à l'infaisabilité (par rapport à la contrainte d'intégralité) due à chaque variable discrète. La variable à séparer est alors celle qui maximise ce coût, i.e. celle qui signifie une augmentation plus importante de la borne inférieure : la convergence sera alors d'autant plus rapide.

Cette technique peut également être appliquée pour le choix du sommet suivant à calculer. En effet, il est possible de calculer le pseudo-coût d'un sommet comme la somme des pseudo-coûts de toutes les variables discrètes à valeur réelle qui ne sont pas encore séparées. La valeur calculée équivaut à un « prix » de l'infaisabilité de la solution du problème relaxé, permettant de diriger le choix vers la branche la plus prometteuse. Cependant, cette stratégie est très coûteuse à cause du temps nécessaire au calcul du pseudo-coût [BRO98].

D'autres techniques plus classiques sont utilisées pour le choix du prochain sommet à traiter, et tout d'abord la recherche en profondeur, qui considère prioritairement les sommets enfants du sommet qui vient d'être traité. Si le sommet courant est tronqué, on remonte dans l'arborescence jusqu'à trouver un sommet qui n'a pas encore été traité (*backtracking*). L'avantage de cette méthode est la similarité des problèmes NLP successivement calculés, qui diffèrent seulement par les bornes sur les variables. En outre, il est notable d'ajouter que les solutions faisables sont souvent situées profondément dans l'arborescence. Par contre, cette technique implique généralement le calcul de beaucoup de sous-problèmes pour pouvoir prouver l'optimalité d'un résultat. Une deuxième méthode classique de choix du sommet consiste à traiter préférentiellement le sommet ayant fourni la borne inférieure la plus petite (il apparaît que c'est souvent celle qui conduit à une convergence plus efficace). Il en résulte habituellement le traitement d'un nombre moins élevé de sous-problèmes.

Il n'existe pas, parmi ces différentes stratégies proposées par *SBB*, de meilleure méthode. Un compromis est à faire pour profiter des avantages respectifs de chacune d'entre elles. *SBB* propose donc également des méthodes intermédiaires, mixant les différentes techniques citées.

1.3.3 – Eléments de comparaison théoriques entre DICOPT++ *et* SBB

Au vu du développement précédent, il est clair que *DICOPT++* et *SBB* ont des modes d'opération radicalement différents. Leur philosophie respective laisse en conséquence prévoir des comportements et performances différents sur un même problème [BRO98].

DICOPT++ est basé sur l'hypothèse que, dans la décomposition NLP/MILP, le goulet d'étranglement est constitué par les sous-problèmes NLP, un nombre trop important de problèmes infaisables pouvant même empêcher la convergence. L'idéal serait alors la résolution d'un nombre réduit de problèmes continus. A l'opposé, les MILP sont généralement résolus plus facilement au moyen de stratégies linéaires simples. Mais, il peut arriver que la linéarisation des contraintes génère des problèmes maîtres mal conçus, dont la solution constitue une mauvaise initialisation pour des NLP difficiles à résoudre, voire infaisables.

Au contraire, *SBB* utilise la plus grande partie de son temps de calcul à résoudre des NLP. Mais, faisant l'hypothèse que les NLP ne diffèrent les uns des autres que par quelques bornes, ceux-ci peuvent être traités grâce à une procédure de redémarrage efficace, en utilisant les valeurs optimales du NLP précédent comme initialisation.

Finalement, il semblerait logique, au vu de ces considérations, que *DICOPT++* soit plus performant sur des problèmes présentant un aspect combinatoire important, tandis que *SBB* sera plus efficace pour des problèmes impliquant moins de variables discrètes, mais des termes plus sévèrement non-linéaires.

1.4 – Solveurs complémentaires

Comme cela a été mis en évidence par le mode de fonctionnement des deux algorithmes explicités ci-dessus, les solveurs *DICOPT++* et *SBB* se situent à un niveau supérieur de la résolution : leur tâche est essentiellement d'analyser le modèle formulé pour créer des problèmes simplifiés, relaxés, aménagés, donc de moindre difficulté et qui peuvent être résolus numériquement. Puis, la gestion des résultats leur permet de construire sur un schéma itératif, de nouveaux problèmes jusqu'à ce que les conditions assurant la convergence soient remplies.

Ainsi, à un niveau inférieur se situent les solveurs effectuant l'essentiel du travail numérique. Ils sont appelés une fois par le module MINLP à chaque itération « majeure » de l'algorithme. Vu leur fonctionnement respectif, *DICOPT++* nécessite un module de résolution de type NLP et un module de type MILP, tandis que *SBB* fait simplement appel à un module NLP.

La figure 7 illustre l'architecture du jeu de solveurs utilisés dans chaque cas. Ce sont les solveurs *CONOPT3* et *CPLEX* qui, faisant partie de la bibliothèque de routines de *GAMS*, ont été choisis. Ils sont décrits dans ce qui suit.

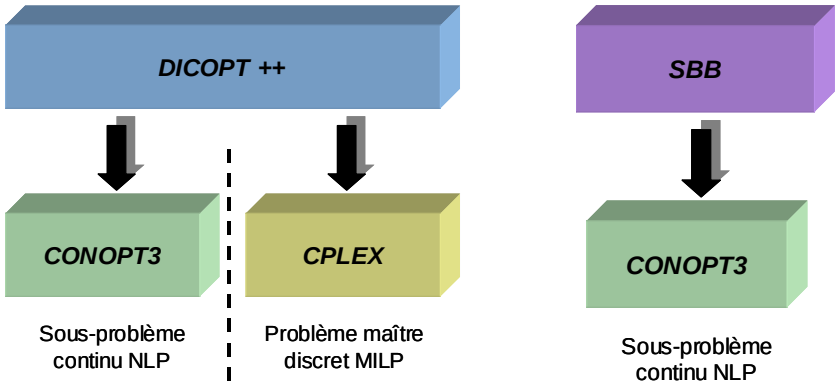


Figure 7 – Hiérarchie des solveurs utilisés

1.4.1 – CONOPT3

Il existe une variété assez large de méthodes numériques pour la résolution de problèmes d'optimisation continue non-linéaire et sous contraintes. D'après [FLO95], les principales sont les méthodes de pénalité (extérieure ou intérieure), la méthode du gradient projetée proposée par Rosen [ROS60], la technique du gradient réduit généralisé (GRG) due à [ABA69] et enfin, les techniques de programmation séquentielle linéaire (SLP) ou quadratique (SQP). La technique du GRG est implémentée dans le solveur *CONOPT3*, développé (comme *SBB*) par la société ARKI Consulting & Development. Elle a été jusqu'à récemment considérée comme l'une des plus performantes pour traiter le cas général où la fonction objectif comme les contraintes sont non-linéaires, bien que les techniques SQP soient actuellement très réputées pour leur efficacité. Il est nécessaire de rappeler que seul un optimum local peut être déterminé par ces méthodes, l'optimum global n'étant assuré que dans des conditions bien particulières, et exceptionnelles en pratique.

La méthode du GRG repose sur une extrapolation de la méthode du gradient réduit, développée initialement pour des problèmes à contraintes linéaires, et adaptée pour la résolution de problèmes non-linéaires. La forme classique du problème résolu est :

$$\begin{aligned} & Min\ f(x) \\ soumis\ à\quad & Ax = b \\ & x \in X \subseteq \mathfrak{R}^n \end{aligned} \qquad (P\text{-}LP)$$

Pour des problèmes à contraintes linéaires inégalité, il est possible de se ramener à la formulation précédente en introduisant une variable d'écart positive pour chaque inégalité. L'idée principale est alors de réduire la taille du problème en déterminant un sous-ensemble de variables indépendantes (dites hors base) à partir desquelles les autres variables du problème (basiques) peuvent être exprimées. Puis, le gradient réduit de la fonction objectif *f*

est calculé seulement par rapport aux variables hors base. L'opposé du gradient réduit fournit une direction de recherche ; une technique de pas de descente optimal est en général adoptée pour déterminer le nouveau vecteur solution à l'itération suivante. Les gradients réduits suivants sont calculés relativement à la même base tant qu'aucune des variables n'atteint ses bornes. Lorsque cela arrive, si c'est une variable hors base, la phase précédente peut continuer. Si c'est une variable basique, alors un changement de base valide est nécessaire pour continuer la recherche. Les opérations mentionnées précédemment sont répétées tant que le gradient réduit est non nul ; il est démontré que ce critère est bien équivalent aux conditions d'optimalité de Kuhn-Tucker.

La méthode explicitée ci-dessus est étendue à des problèmes à contraintes non-linéaires en résolvant une séquence de problèmes, chacun d'entre eux constituant une itération majeure. A chaque itération majeure, un nouveau problème est construit en linéarisant les contraintes à la solution de l'itération majeure précédente et en pénalisant la fonction objectif. Ce problème est résolu au moyen de la méthode du gradient réduit, en un certain nombre d'itérations mineures. La séquence s'arrête lorsque les conditions de Kuhn-Tucker sont remplies, et les contraintes non-linéaires vérifiées.

Il est alors intéressant de noter que si la méthode du gradient réduit est à chemin réalisable, la méthode du GRG classique est, elle, à chemin irréalisable (ce qui est néanmoins trivial vu son principe). Cependant, l'algorithme implémenté dans *CONOPT3* est particulier dans le sens où la faisabilité est maintenue au cours de la recherche. Les détails de l'algorithme implémenté dans *CONOPT3* sont disponibles dans [DRU85], bien que de nombreuses modifications aient été apportées pour améliorer ses performances.

En effet, *CONOPT3* se compose tout d'abord d'une partie de pré-traitement du modèle, visant à une réduction préliminaire de la dimension du problème. Ce mécanisme est réalisé en identifiant les contraintes où intervient une seule variable. La variable et la contrainte sont alors supprimées par des techniques de substitution, ce qui a d'ailleurs un effet de propagation en créant de nouvelles contraintes monovariées (effet triangulaire). Ce cas est rencontré typiquement lorsque *CONOPT3* est imbriqué dans une procédure de résolution MINLP : une contrainte peut impliquer des variables entières et ces dernières être fixées par le module MINLP (après une étape de séparation du Branch & Bound par exemple). Si une variable continue intervenait aussi dans la contrainte, alors elle devient la seule variable de la contrainte. De la même manière, des contraintes et des variables intermédiaires peuvent être supprimées en intégrant les premières dans la fonction objectif, où elles sont généralement plus faciles à traiter.

Après une procédure de mise à l'échelle des ordres de grandeur de chaque contrainte (*scaling*), la recherche d'une première solution faisable débute. Celle-ci s'effectue en deux temps. Dans le premier, un nombre de variables égal au degré de liberté du problème est fixé et le système d'équations non-linéaires résultant des contraintes est résolu par une méthode de Newton. Cette technique fonctionne assez bien quand les non-linéarités ne sont pas trop fortes, mais l'intégralité du problème ne peut souvent pas être résolue aussi simplement.

Une deuxième phase consiste alors en une méthode plus systématique mais aussi plus coûteuse en temps de calcul, passant par la minimisation de la somme des variables résiduelles d'infaisabilité introduites dans toutes les contraintes non vérifiées. Ce problème de faisabilité est lui-même résolu par la méthode du GRG et trouve sa solution quand les variables résiduelles tendent vers zéro.

Finalement, une fois la première solution faisable trouvée, l'algorithme du GRG est appliqué. Le choix des variables basiques et les directions de descente sont orientés par des informations du second-ordre de manière à maintenir la faisabilité au cours de la recherche. Le critère d'arrêt est donc atteint quand le gradient réduit est nul (sous une certaine tolérance) ou quand il n'y a plus de variables hors base (la globalité de l'optimum peut alors parfois être prouvée). Il est proposé de se référer à [BRO98] pour plus de détails sur la méthode et les options du module de résolution dans *GAMS*.

1.4.2 – CPLEX

Le module *CPLEX* est développé par la société ILOG pour un usage comprenant d'une part la résolution de problèmes continus : programmation linéaire (LP) et quadratique (QP) ; d'autre part, il est capable de résoudre des problèmes linéaires en variables mixtes (MILP) de grande taille. Etant donné que *DICOPT++* fait appel à un module de résolution MILP, c'est le second aspect qui intéresse le plus cette étude, même si la résolution de problèmes LP est bien entendu un passage obligé du traitement des problèmes MILP.

Les algorithmes dédiés aux problèmes MILP peuvent être rangés dans les classes suivantes [FLO95] : (i) Branch & Bound ; (ii) Méthodes de plans de coupe ; (iii) Méthodes de décomposition ; (iv) Méthodes disjonctives. L'algorithme implémenté dans *CPLEX* est en fait la combinaison d'une technique classique de Branch & Bound et d'une méthode de plans de coupes. Comme ils ont déjà été exposés lors de la présentation de *SBB*, les principes de la méthode de Branch & Bound ne seront pas rappelés ici. La seule différence est en effet que la relaxation continue à chaque sommet de l'arbre de recherche est un problème de programmation linéaire.

La technique des plans de coupe, initiée dans [GOM58], consiste à générer de nouvelles contraintes (*coupes*) qui réduisent l'espace faisable des problèmes linéaires relaxés, mais pas celui du problème initial en variables mixtes. Ainsi, sa combinaison avec la méthode de Branch & Bound s'appelle le Branch & Cut : à chaque sommet où une solution *(x*,y*)* non mixte au problème relaxé est trouvée, le problème de séparation suivant (à ne pas confondre avec la séparation de deux variables dans l'arbre de Branch & Bound) est posé : trouver une contrainte inégalité qui est valide pour le problème MILP mais violée par *(x*,y*)*.

Ce problème de séparation est généralement compliqué à résoudre, mais des coupes particulières ont été proposées dans divers algorithmes, se basant sur la récurrence de certaines formulations en programmation mixte. *CPLEX* utilise plusieurs des méthodes de coupe existantes, mais elles ne seront pas présentées ici dans la mesure où elles nécessiteraient une explication longue et méthodique des concepts mis en jeu.

Il est suggéré de se reporter à [BIX00] pour les techniques employées au sein de *CPLEX* et à [PAD91], [LET03] et [JOH00] pour plus de détails sur la méthode de Branch & Cut et ses variantes.

Enfin, concernant la résolution des relaxations continues linéaires, *CPLEX* propose différents algorithmes : simplex primal et dual, ainsi que les algorithmes de *barrier*, *sifting* et *network optimizer*. Ils ne seront pas non plus détaillés ici. Le mode de fonctionnement consiste généralement à utiliser toutes ces méthodes parallèlement, la plus rapide fournissant la solution.

Les principes des méthodes de programmation mathématique retenues ont été explicités et leur mode de fonctionnement respectif détaillé. La méthode stochastique est alors présentée dans la partie suivante.

2 – Méthode stochastique : Algorithme Génétique

Dans la classe des métaheuristiques, les algorithmes génétiques font partie de l'ensemble des Algorithmes Evolutifs. Ces derniers reposent sur des principes communs, c'est-à-dire le traitement d'une population de solutions à laquelle sont appliquées des heuristiques qui la font évoluer vers une meilleure qualité vis-à-vis d'un problème d'optimisation donné. Les algorithmes génétiques ont fait leurs preuves sur de larges classes d'exemples, et l'amélioration de leur fonctionnement fait toujours l'objet de travaux de recherche poussés.

2.1 – Principe généraux

Le principe d'un AG repose sur l'analogie conceptuelle établie entre une population d'individus évoluant dans leur milieu naturel et un ensemble de solutions plus ou moins bonnes d'un problème d'optimisation quelconque. Suivant les règles de l'évolution énoncées par Darwin, la population d'individus va évoluer de manière à s'adapter au milieu qui l'entoure : les plus faibles vont disparaître tandis que les mieux adaptés survivront et se reproduiront. Au sein de leur patrimoine génétique se conserveront donc, au fil des générations, les caractéristiques qui rendent ces individus plus "forts", mieux adaptés à leur environnement, et assureront la pérennité de l'espèce [BAU97].

Les premiers algorithmes développés sur ces bases par des biologistes, dans les années 1950, cherchaient à simuler l'évolution d'êtres vivants. Mais ce n'est qu'à partir des années 1970 que des numériciens se sont intéressés à ce domaine pour en tirer parti dans le cadre de l'optimisation. Holland [HOL75] puis Goldberg [GOL89] furent de fait les pionniers de la transition entre théorie de l'évolution et algorithmes évolutifs d'optimisation.

Dans la réalité, les découvertes de Watson et Crick et les principes de la génétique moderne ont défini la manière dont sont stockées les données caractérisant un individu : l'enchaînement de quatre bases forme l'ADN qui, lui-même, constitue les gènes, lesquels s'expriment sur des chromosomes. Cet ensemble représente le phénotype d'un individu.

L'évolution du phénotype est basée sur le concept d'hérédité : le phénotype d'un individu est construit à partir de celui des individus des générations antérieures. Le plus souvent, il résulte de la recombinaison des phénotypes de deux individus parents. Le brassage des gènes assure tout d'abord la conservation des caractéristiques qui font la force de ces individus. Cette tendance s'exprime par le fait que deux individus forts ont toutes les chances de donner naissance à un individu qui sera bien armé pour survivre à la sélection naturelle imposée par l'environnement et pour se reproduire lui aussi. Mais il permet parfois aussi à certains gènes de s'exprimer, ce qui peut constituer un apport au regard des générations précédentes. De plus, le phénotype est sujet à des mutations, globalement aléatoires, qui peuvent altérer sa qualité mais tout aussi bien introduire de nouvelles caractéristiques favorisant l'adaptation de l'individu et la variété nécessaire au développement de l'espèce.

Reprenant l'analogie évoquée plus haut, un individu représente une solution à un problème d'optimisation. Son matériel génétique est alors la structure de données qui décrit cette solution, le plus souvent ses variables. Après avoir établi une forme de codage permettant de représenter de façon numérique le « génotype » de la solution, les mécanismes

d'évolution mentionnés ci-dessus peuvent être transposés à une population de solutions. En créant un mode d'évaluation traduisant l'adaptation ou non d'un individu-solution à l'environnement-instance du problème auquel il est soumis, il est possible de déterminer parmi la population les meilleures solutions. Celles-ci auront alors plus de chances de survivre à une procédure de sélection laissant cependant la place au hasard (représenté par des effets aléatoires). On applique aux survivants des opérateurs de croisement, mimant la reproduction, ou de mutation. Ils permettent de diversifier la recherche et d'éviter de la figer sur un optimum local : en effet, contrairement aux méthodes de Programmation Mathématique, la dégradation de la fonction objectif est autorisée. Ce procédé est appliqué itérativement pour obtenir, au bout d'un certain nombre de générations, une population d'individus bien adaptés à leur milieu, c'est-à-dire un ensemble de « bonnes » solutions au problème d'optimisation auquel on s'intéresse.

L'intervention de générations aléatoires dans la majorité des opérateurs mis en jeu dans le déroulement d'un algorithme génétique confère à cette méthode son caractère stochastique. C'est ce dernier qui permet une exploration plus large de l'espace de recherche. Cet aspect est cependant canalisé par les heuristiques qui définissent les mécanismes de l'algorithme génétique et qui affirment sa supériorité sur une simple recherche aléatoire.

Finalement, la puissance des algorithmes génétiques repose sur leur aspect générique. L'évolution des espèces n'est guidée préalablement dans aucune direction, par aucun facteur, si ce n'est la force de certains individus par rapport aux autres. De la même manière, l'algorithme génétique n'a pas besoin d'autre information que le critère des solutions. Il doit être vu comme un concept adaptable et adapté pour la résolution de problèmes d'optimisation complexes [BÄC97].

2.2 – Mise en oeuvre d'un Algorithme Génétique

L'algorithme génétique utilisé dans cette étude a été adapté d'une version développée dans l'équipe. L'organigramme détaillant l'enchaînement de ses étapes est présenté sur la figure 8. Il souligne les différentes procédures qui doivent être définies pour la mise en œuvre de l'algorithme.

Ces procédures sont au nombre de sept :

- Création de la population initiale : étant le point de départ de l'algorithme génétique, la population initiale doit proposer un échantillonnage représentatif de l'ensemble de l'espace de recherche.

- Méthode d'évaluation des individus : elle traduit leur adaptation vis-à-vis du problème étudié, et est donc calculée d'une manière générale à partir de la fonction objectif et/ou du respect des contraintes intrinsèques au problème.
- Procédure de sélection déterminant les individus survivants : nécessairement en adéquation avec la méthode d'évaluation évoquée ci-dessus, elle met en œuvre le processus de survie des meilleurs individus propres à la théorie de l'évolution.
- Opérateur de croisement : illustrant la reproduction de deux individus, celui-ci doit permettre le brassage efficace des gènes d'une population.
- Opérateur de mutation : il apporte un caractère stochastique important pour le fonctionnement de l'algorithme génétique en introduisant aléatoirement des modifications ponctuelles du génotype d'un individu.
- Facteur d'élitisme : il permet de conserver les meilleurs individus d'une génération à l'autre.
- Méthode de codage des variables : elle n'apparaît pas sur le schéma mais est indispensable au niveau de l'évaluation des individus. En effet, c'est le codage qui est le trait d'union entre le chromosome, vecteur sans signification explicite, et le jeu de variables représentant la physique du problème. Pour le cas de conception optimale d'ateliers discontinus, cette relation est exprimée dans la figure 9.

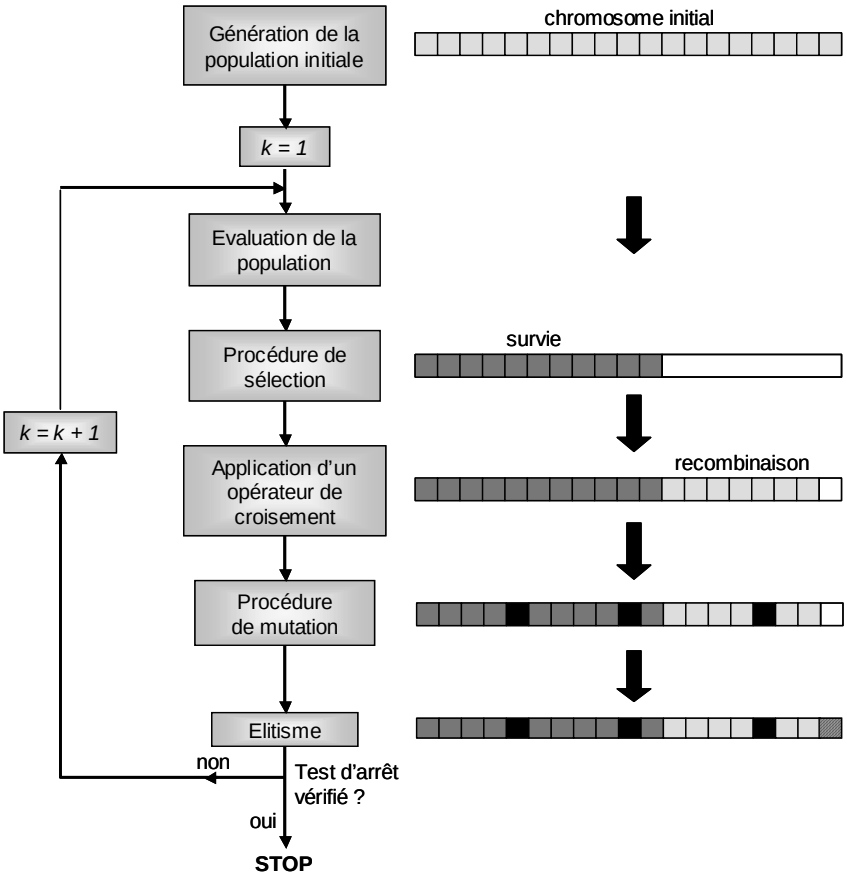


Figure 8 – Organigramme d'un algorithme génétique classique

Le codage dépend naturellement de la nature des variables, discrètes ou continues, et pour les secondes, de la précision que l'on souhaite avoir sur ces dernières. De la même manière, les opérateurs de croisement et de mutation doivent être en harmonie avec le codage employé pour pouvoir agir de manière adéquate sur les variables et atteindre les deux objectifs qui leur sont fixés : diversification de la population mais aussi conservation de la qualité du patrimoine génétique. Ces deux aspects, apparemment opposés, équivalent dans le langage des métaheuristiques aux notions d'exploration de l'espace de recherche et d'intensification sur les régions prometteuses.

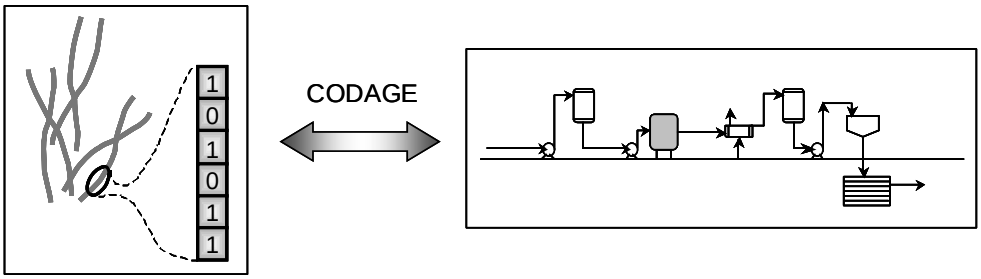


Figure 9 – Codage d'un atelier discontinu

Un deuxième point-clé dans le développement d'un algorithme génétique n'apparaît pas non plus sur la figure 8. Il concerne le cas où un problème d'optimisation est soumis à des contraintes et la gestion de ces dernières. Cette remarque, aussi importante que le codage, orientera le choix d'un mode de sélection approprié et la méthode de génération de la population initiale. Dans des exemples traditionnels de Génie des Procédés, généralement sévèrement contraints, la gestion des contraintes aura une influence déterminante sur les performances de l'algorithme.

Malgré le caractère générique des algorithmes génétiques, ces deux aspects nécessitent donc une adaptation particulière au problème considéré. Par contre, certains paramètres peuvent être définis et établis de manière plus définitive.

2.3 – Paramètres classiques

Parmi ces paramètres figurent tout d'abord des paramètres généraux qui fixent de manière numérique le caractère évolutif de l'algorithme. Dans un deuxième temps, deux procédures apparaissant sur la figure 8 et non affectées par le codage ou la gestion des contraintes sont explicitées : l'élitisme et le critère d'arrêt.

2.3.1 – Paramètres de fonctionnement

Le réglage des paramètres de fonctionnement constitue la principale difficulté de l'utilisation d'un algorithme génétique. Celui-ci repose essentiellement sur l'expérience de l'utilisateur et sa connaissance du problème. Les paramètres doivent néanmoins être ajustés de manière adéquate car ils influent largement sur le résultat final. Des études de sensibilité et des choix empiriques sont souvent nécessaires pour déterminer le meilleur jeu des paramètres suivants :

- taille de population ;
- nombre de générations maximal ;
- taux de survie ;
- taux de mutation.

Les deux premiers dépendront largement de la taille du problème. En effet, il est clair que plus ils sont élevés, plus le nombre de solutions visitées est grand, ce qui a une influence très positive sur la qualité du résultat final mais très pénalisante concernant le temps de calcul. Ce dernier devient un facteur limitant dans le cas de problèmes complexes, pour lesquels l'évaluation est longue. Par ailleurs, il est établi que la recherche est plus efficace si le nombre de générations calculées est supérieur ou égal à la taille de population. Ceci se comprend aisément si l'on prend en compte le fait que l'algorithme tire une bonne partie de son efficacité du mélange, du partage des différents génotypes. Un nombre de générations trop faible pourrait alors ressembler à une génération aléatoire de solutions. Au cours des études réalisées, les valeurs de ces deux paramètres seront ajustées de manière empirique selon la taille du problème abordé, de manière à trouver un compromis entre temps de calcul et qualité de la solution obtenue.

Les taux de croisement (complémentaire du taux de survie) et de mutation ont également un impact important sur le déroulement de la recherche. S'ils sont trop élevés, ils peuvent détruire rapidement des structures prometteuses et même gêner la convergence. Par contre, des valeurs trop faibles entraîneront un brassage des gènes peu efficace et une diversification de la population peu importante, pouvant causer la stagnation sur des optima locaux.

2.3.2 – Elitisme

Les procédures d'élitisme ont pour objectif de conserver le patrimoine génétique du ou des meilleurs individus au fil des générations. En effet, bien que les étapes de sélection permettent d'assurer à ces derniers de grandes chances de survie, les effets aléatoires peuvent

tout de même entraîner la perte de bons éléments. Dans l'algorithme utilisé, divers degrés d'élitisme, de sévérité croissante, sont envisageables :

- Conservation du meilleur individu. Il remplacera un individu choisi aléatoirement à la génération suivante.
- Conservation des *N* meilleurs individus. Ils remplaceront *N* individus choisis aléatoirement à la génération suivante.
- Conservation des *N* meilleurs individus. Ils remplaceront les *N* moins bons individus de la population suivante.

Dans l'approche envisagée, c'est le deuxième schéma qui a été choisi, *N* représentant 2 % de la population. Cette proportion doit rester assez faible de manière à ne pas favoriser excessivement l'exploration locale et conduire la recherche vers un optimum local.

2.3.3 – Critère d'arrêt

Malgré les efforts réalisés dans ce sens, aucun critère d'arrêt général et universel, en accord avec la convergence vers une solution satisfaisante, n'a pu être clairement établi. Le critère le plus répandu est lié au nombre d'itérations de l'algorithme : nombre de générations maximal, temps de calcul, nombre d'évaluations de la fonction objectif…

L'algorithme utilisé dans cette étude propose en outre un critère d'arrêt basé sur la non-évolution de la moyenne et de l'écart-type de la fonction objectif dans la population, sur un nombre fixé de générations. Divers tests ont cependant montré l'inefficacité de cette méthode, due au fait que l'aspect aléatoire introduit par le croisement et la mutation engendre, à chaque génération, des solutions de qualité moyenne inégale et un nombre différent de solutions infaisables. Par conséquent, le bruit provoqué sur la moyenne et l'écart type de la fonction objectif au cours des générations empêche l'algorithme de conclure sur une non-évolution. Ainsi, le critère d'arrêt retenu dans ici est, de manière classique, le nombre de générations. Son caractère intuitif ne peut fournir de garantie sur la qualité de la solution. Vu l'aspect stochastique de l'AG, il est donc nécessaire de traiter plusieurs fois le même problème pour s'assurer de la convergence.

2.4 – Enjeux

Les grandes lignes définissant le déroulement de l'algorithme génétique sont ainsi tracées et des paramètres de fonctionnement ont été choisis. Mais même si la définition de ces basiques est un passage obligé pour l'utilisation de l'AG et si leur réglage permet d'améliorer

les performances, les enjeux se situent ici sur un plan fondamental et sont basés sur les deux points suivants :

- Prise en compte de la nature mixte des variables.
- Définition d'un mode de gestion des contraintes efficace dans une structure qui ne les prévoit pas initialement.

Car, comme cela a été exposé dans la définition de la problématique, les algorithmes génétiques doivent être capables de gérer des problèmes industriels, mettant en œuvre des modèles complexes implémentés au sein de simulateurs impliquant des variables mixtes et soumis à des contraintes plus ou moins sévères. Une réponse pertinente à ces exigences nécessite un effort d'adaptation de la méthode aux problèmes abordés. Cette phase implique la considération et la comparaison de diverses techniques de mise en œuvre des procédures propres à l'algorithme génétique.

2.4.1 – Gestion des variables

Par leur fonctionnement, les algorithmes génétiques sont voués au traitement de problèmes discrets. La prise en compte de variables continues passe nécessairement par leur discrétisation. Un nombre important de méthodes de discrétisation, dont la plus célèbre est sans doute le codage binaire, a été développé. Les techniques de croisement ont alors consisté, de manière presque intuitive, en de simples manipulations de la structure des chromosomes, tandis que les opérateurs de mutation réalisaient des opérations simples, plus ou moins adaptées à la physique du problème, sur les valeurs des bits (inversion, soustraction ou ajout d'une unité… [DED01]). Plus récemment, les variables continues ont été codées sur des bits à valeur réelle, donnant naissance aux « real-valued GAs », largement utilisés depuis lors [JAIN05] [TSU01]. Des procédures numériques de croisement et de mutation plus complexes, tâchant de reproduire le comportement des opérateurs agissant sur des chromosomes à valeurs entières, ont été adaptées à ce nouveau fonctionnement.

Variant par la taille du chromosome, par la précision de la représentation des variables, par la pertinence des opérateurs génétiques qui leur sont associés, les différentes formes de codage ont une influence déterminante sur le résultat obtenu et sur la rapidité du calcul. Le codage est donc un facteur clé de l'efficacité de l'algorithme génétique.

2.4.2 – Gestion des contraintes

La gestion des contraintes dans un algorithme génétique représente le deuxième problème sur lequel s'est portée notre attention. En effet, les contraintes ne peuvent pas toujours être

simplement implémentées comme des équations supplémentaires s'ajoutant au modèle, comme c'est le cas en Programmation Mathématique. Il semblerait naturel de fixer de manière rigide les limites de l'espace faisable en éliminant les solutions violant une des contraintes, mais ce mode de gestion est très pénalisant en termes de temps de calcul. En outre, il est probable que ce fonctionnement fasse également perdre des solutions qui, par le biais d'une partie de leur patrimoine génétique, pourraient se révéler intéressantes. D'autres techniques, plus ou moins détournées, ont été développées pour traiter efficacement ce problème.

La procédure de sélection de l'algorithme génétique est, prioritairement, l'étape au cours de laquelle s'exprime le mode de gestion des contraintes employé : roulette de Golberg, tournois, tri de Pareto… Mais celui-ci apparaît aussi implicitement au moment de l'évaluation de la population puisque la sélection se fait à partir du critère relatif à chaque individu. Il est ainsi nécessaire de calculer des « forces », de pénaliser la fonction objectif ou de juger l'importance de la violation de la contrainte. Le choix du mode de gestion des contraintes a également des répercussions sur la façon dont la population initiale va être générée, puisqu'il peut impliquer ou non la création d'une population entièrement ou partiellement faisable.

Ainsi, la manière selon laquelle le codage et la gestion des contraintes vont être implémentés constitue un des axes de cette étude. La comparaison des résultats obtenus au moyen de diverses techniques sur plusieurs exemples tendra à mettre en valeur la plus efficace. Tels sont les enjeux des calculs effectués dans les parties suivantes : la détermination des techniques de codage et de gestion des contraintes, accompagnées des procédures et opérateurs génétiques qui en découlent, pour l'utilisation pertinente d'un algorithme génétique sur des problèmes complexes d'optimisation en variables mixtes.

Conclusion

Dans ce chapitre, les méthodes d'optimisation représentant le fil rouge de ces travaux ont été présentées. Le choix s'est porté d'une part sur deux méthodes déterministes, les solveurs commerciaux *DICOPT++* et *SBB*, et d'autre part sur une méthode stochastique, un algorithme génétique. Ce choix implique l'utilisation d'un modèle général, analytique, auquel peuvent s'appliquer les deux types de méthodes. La formulation de ce modèle sera présentée dans le chapitre suivant.

Les principes algorithmiques des deux méthodes de Programmation Mathématique ont été explicités, dans l'objectif non seulement de clarifier leur utilisation mais surtout dans l'optique d'une compréhension accrue des résultats obtenus et présentés par la suite.

Finalement, les principes généraux définissant les heuristiques de l'algorithme génétique ont été exposés. Les paramètres classiques de son fonctionnement ont été définis et déterminés. Mais surtout, les enjeux pour la performance de l'AG, ainsi que les procédures particulières qui lui sont associées, ont été ciblés : il s'agit de déterminer un codage et un mode de gestion des contraintes adaptés au problème abordé.

CHAPITRE 2

MODELE DE CONCEPTION D'ATELIERS DISCONTINUS

Ce chapitre présente le cadre de l'étude, dans lequel s'inscrit la comparaison des performances des méthodes d'optimisation décrites dans le chapitre précédent. La conception d'ateliers discontinus, reconnue comme un problème scientifique majeur en Génie des Procédés, constitue le support de l'étude. La richesse de la production scientifique sur cette thématique en atteste.

Ce chapitre se divise en quatre parties. Dans un premier temps, les fondements théoriques et les hypothèses du formalisme utilisé sont présentés, puis le modèle est développé en détail. Ce modèle ne peut cependant pas être soumis aux méthodes de résolution sans un conditionnement ou, en d'autres termes, une adaptation préalable. Les modifications qui lui sont apportées, particulières à chaque classe de méthodes d'optimisation, sont donc décrites par la suite. Enfin, la création et la mise en forme du jeu d'exemples sur lequel l'étude a été menée sont explicitées.

1 – Formalisme retenu de conception d'ateliers discontinus

Il convient d'abord de présenter le formalisme retenu pour modéliser le problème de conception optimale d'ateliers discontinus. Il est soumis à certaines hypothèses simplificatrices qui amènent à traiter des exemples, certes didactiques, mais néanmoins industriellement réalistes. Les lignes directrices du modèle sont exposées ci-après.

1.1 – Généralités

D'une manière générale, les équipements fonctionnant selon un mode discontinu nécessitent la charge par un opérateur d'un lot de produits dans chaque appareil. Le lot subit un certain nombre de traitements dans différents équipements suivant une séquence opératoire, ou recette, pour synthétiser un produit donné. Les appareils doivent être nettoyés après chaque passage d'un lot pour éviter d'éventuelles contaminations croisées. La flexibilité de l'atelier est une des qualités essentielles requises pour répondre aux exigences d'un marché variable : il doit pouvoir être adapté à des changements tant en termes de nature que de demande ou de caractéristiques des produits fabriqués.

Dans le cadre de l'étude, seuls des ateliers multiproduit sont envisagés, correspondant à un degré de complexité intermédiaire, entre monoproduit et multiobjectif. Cela signifie que plusieurs produits peuvent être synthétisés, mais ceux-ci auront un cheminement identique dans l'atelier, selon une recette similaire, ce qui restreint les difficultés liées à la modélisation.

Classiquement, l'objectif des problèmes de conception de procédés discontinus est de minimiser le coût d'investissement pour la construction d'un atelier en optimisant le nombre et la taille des équipements. D'autres critères ont cependant été formulés, comme la maximisation du bénéfice actualisé (*NPV* pour *Net Present Value*), d'un critère de flexibilité [AGU05] ou d'un critère mesurant l'impact sur l'environnement du rejet d'effluents [DIE04]… Dans le cadre présent d'une étude comparative de méthodes d'optimisation, seul le critère classique de coût d'investissement est pris en compte. Les données à disposition sont la quantité souhaitée de chaque produit, les paramètres relatifs aux appareils (coefficients de coût, temps opératoires) et l'horizon de temps attribué à la production totale.

Le formalisme retenu pour modéliser le problème étudié est largement abordé dans la littérature dédiée à la conception optimale d'ateliers discontinus multiproduit. Bien entendu, sa formulation s'est complexifiée pour reproduire au mieux les contraintes industrielles, au fur et à mesure des améliorations tant des performances des méthodes d'optimisation que de la puissance des calculateurs. Pour mémoire, dans la version initiale de cette formulation, datant de 1972, le problème consistait à optimiser les volumes continus des équipements de trois ou quatre étapes opératoires [ROB72] (un équipement par étape). Les versions les plus récentes considèrent, outre des étapes pouvant comporter plusieurs appareils en parallèle, des équipement discontinus et semi-continus, des bacs de stockage intermédiaire… Elles aboutissent à des tailles d'ateliers synthétisant une quinzaine de produits en une quinzaine d'étapes. Le choix de cette formulation a permis en outre de comparer, du moins sur quelques instances du problème, les résultats avec ceux de la littérature.

1.2 – Description et hypothèses

Le formalisme utilisé est celui employé par Modi et Karimi [MOD89]. Il a été repris de nombreuses fois (cf. partie 3 de l'introduction de ce mémoire). Différents modes de résolution lui ont été successivement appliqués : heuristiques, recuit simulé, algorithme génétique, recherche tabou, recherche par colonie de fourmis… (cf. tableau 1 de l'introduction).

Ce modèle prend en compte non seulement les équipements discontinus apparaissant habituellement dans toutes les formulations, mais présente en outre l'avantage de considérer les appareils semi-continus qui font partie intégrante du procédé global (pompes, échangeurs de chaleur, de matière, etc...). Un appareil semi-continu est défini comme un appareil continu fonctionnant suivant une alternance de périodes de faible activité et de périodes d'activité uniforme normale [MAH90]. De la même manière que la variable caractéristique associée à un élément discontinu est son volume, à chaque équipement semi-continu correspond une capacité de traitement par unité de temps, de dimension [$L.h^{-1}$]. Un groupe de plusieurs étapes

semi-continues successives est appelé sous-train semi-continu, dont l'équipement limitant est celui qui a le temps de traitement le plus élevé.

Chaque étape opératoire, discontinue ou semi-continue, comporte plusieurs équipements en parallèle. Une formulation habituelle consiste à prendre en compte à la fois des équipements fonctionnant en-phase et hors-phase. Le fonctionnement en-phase correspond à l'activité synchronisée de plusieurs équipements en parallèle, c'est-à-dire que les dates de début et de fin de la tâche coïncident ; par opposition, le fonctionnement hors-phase correspond à un type de fonctionnement asynchrone [AZZ05]. Ici, la formulation de Modi et Karimi [MOD89] exclut tout type de séparation ou regroupement des tâches, tous les équipements en parallèle fonctionnant hors-phase.

Par ailleurs, la formulation prend également en compte des bacs de stockage intermédiaire, pour une moyenne ou faible durée. Ils séparent l'atelier en différents sous-procédés et servent simplement à découpler les tailles de lots et les temps de cycle (temps passé par un lot de produit dans l'équipement limitant) entre les sous-procédés en amont et en aval du stockage. Ceci revient à conserver momentanément une éventuelle quantité de matière correspondant à la différence entre les productivités de chaque sous-procédé. Ce découplage confère ainsi au procédé global une plus grande souplesse lors de la résolution, évitant qu'un équipement limitant ne paralyse la production de tout l'atelier. Au final, un atelier discontinu est représenté comme une suite d'équipements discontinus (B), d'équipements semi-continus (SC) et de bacs de stockage intermédiaire (T). L'équivalence entre le schéma d'un atelier discontinu typique et sa forme modélisée, i.e. un enchaînement de « boîtes » auxquelles sont affectés des facteurs de coût et de temps opératoire, est illustrée par la figure 1.

Le modèle obéit aux hypothèses suivantes :

(i) Le procédé fonctionne « par lots », c'est-à-dire en injectant dans l'atelier des quantités prédéfinies de produits représentant une fraction de la production totale souhaitée.
(ii) Les appareils employés sur une même ligne de production ne peuvent être communs à plusieurs étapes.
(iii) L'atelier fonctionne suivant une succession de campagnes mono-produit.
(iv) Les équipements d'une même étape, discontinue ou semi-continue, sont de même type et ont le même volume.
(v) Tous les bacs de stockage intermédiaire sont de taille finie.
(vi) Le fonctionnement du procédé entre deux étapes est de type "zéro attente", sauf s'il existe un bac de stockage entre ces deux étapes.
(vii) Aucune limitation sur les stockages en entrée et sortie n'est considérée.

(viii) Le temps de nettoyage des équipements est compris dans le temps de traitement.

(ix) Les tailles des appareils varient de manière continue entre leurs bornes.

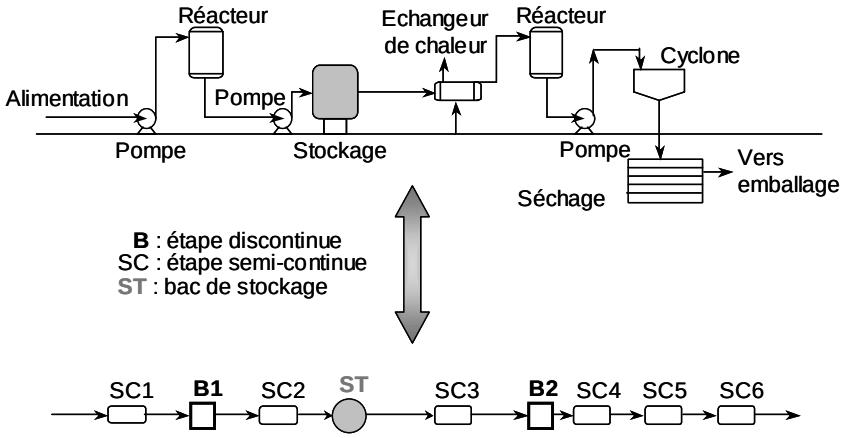


Figure 1 – Atelier discontinu et sa représentation

Le modèle mathématique correspondant au mode de représentation de l'atelier discontinu et aux hypothèses définies ci-dessus est décrit dans la partie suivante.

2 – Formulation du modèle

On considère J étapes discontinues, K étapes semi-continues et I produits à synthétiser. Chaque étape discontinue d'indice j est composée de m_j équipements de taille identique V_j et fonctionnant hors-phase. Chaque étape semi-continue d'indice k est composée de n_k équipements de même taux opératoire R_k (correspondant à une capacité de traitement par unité de temps) et fonctionnant hors-phase. Les enchaînements non-interrompus d'étapes semi-continues, ou trains semi-continus, sont indicés sur t. Les S-1 bacs de stockage intermédiaire, de volume V_s, divisent le procédé global en S sous-procédés.

2.1 – Fonction objectif et variables

Suivant les hypothèses et notations détaillées plus haut, il est possible de formuler un problème d'optimisation de type MINLP, dont le critère à minimiser est le coût d'investissement pour l'ensemble des équipements de l'atelier. Ce coût d'investissement s'écrit classiquement sous la forme d'une fonction exponentielle de la taille des différents appareils :

$$Min\ Coût = \sum_{j=1}^{J} m_j a_j V_j^{\alpha_j} + \sum_{k=1}^{K} n_k b_k R_k^{\beta_k} + \sum_{s=1}^{S-1} c_s V_s^{\gamma_s} \qquad (1)$$

L'équation (1) met en évidence que seul un terme exponentiel est considéré dans le prix des différents équipements, qui s'écrit donc : aX^{α} (où X est la variable d'optimisation continue caractérisant l'équipement, qu'il soit discontinu ou semi-continu). Cette expression devrait prendre en compte un coût fixe (représentant les capteurs, indicateurs, vannes...) et s'écrire : $a_1 + a_2X^{\alpha}$ mais en fonctionnement continu, les équipements sont régulièrement de très grande taille et la constante a_1 est négligée. Cette approximation a été conservée dans ces travaux.

Cependant, il est important de noter qu'elle est ici moins justifiée, car l'ordre de grandeur des tailles d'équipements est largement inférieur à celui des procédés continus. Par ailleurs, ce terme fixe aiderait certainement à diriger l'optimisation vers la diminution du nombre d'équipements par étape. La formulation présentée a néanmoins été adoptée pour deux raisons : d'une part, aucune valeur relative à ces coûts fixes n'a été trouvée dans la littérature ; d'autre part, cette démarche a permis de comparer les résultats obtenus avec ceux de travaux antérieurs. Les valeurs des coefficients de coût sont données dans le tableau 1.

Etapes discontinues	a_j	250
	α_j	0,60
Etapes semi-continues	b_k	370
	β_k	0,22
Bacs de stockage intermédiaire	c_s	278
	γ_s	0,49

Tableau 1 – Coefficients pour le calcul du coût des équipements

Il est à noter que, comme la place des bacs de stockage est fixée par les données du problème et leur volume calculé dans le modèle, la structure de l'atelier discontinu est décrite complètement par V_j et m_j, et R_k et n_k. Ces dernières sont donc les variables d'optimisation et sont au nombre de deux par étape (discontinue ou semi-continue) : une variable continue (volume ou taux opératoire) et une variable entière (nombre d'équipements par étape). Ces variables sont bornées :

$$\forall j \in \{1,..,J\} \qquad V_{min} \leq V_j \leq V_{max} \tag{2}$$

$$\forall k \in \{1,..,K\} \qquad R_{min} \leq R_k \leq R_{max} \tag{3}$$

$$\forall j \in \{1,..,J\} \qquad N_{min} \leq m_j \leq N_{max} \tag{4}$$

$$\forall k \in \{1,..,K\} \qquad N_{min} \leq n_k \leq N_{max} \tag{5}$$

Dans les chapitres suivants, on prendra :

- V_{min} = 250 [L] ;
- R_{min} = 300 [L.h^{-1}] ;
- V_{max} = 10000 [L] ;
- R_{max} = 10000 [L.h^{-1}] ;
- N_{min} = 1 et N_{max} = 3.

2.2 – Contrainte et modèle

Outre les bornes sur les variables, le problème est soumis à une forme de contrainte, dite temporelle. Elle impose à la somme des temps nécessaires à la production de tous les produits d'être inférieure à un horizon de temps H fixé dans les données du problème :

$$\sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{Prod_i} \leq H \tag{6}$$

où Q_i [kg] désigne la demande en produit d'indice i et $Prod_i$ [kg.h^{-1}] la productivité globale du procédé en produit i. Dans tous les exemples étudiés, on aura H = 6000 [h], ce qui correspond à une année à laquelle sont retranchées les périodes de congé. Q_i fait partie des données de chaque problème. Enfin, la productivité globale en produit i est nécessairement le minimum des productivités locales $Prodloc_{is}$ [kg.h^{-1}], définies, pour le même produit i, sur chaque sous-procédé s par l'équation (7) :

$$\forall i \in \{1,..,I\}\ Prod_i = \min_{s \in S}\left[Prodloc_{is}\right] \tag{7}$$

Le calcul des productivités locales dans chaque sous-procédé s, $Prodloc_{is}$, découle de la vérification des équations (8) à (14), exprimées comme suit.

- ✓ La productivité locale en produit i dans le sous-procédé s est le rapport entre la taille des lots B_{is} [kg] et le temps de cycle limitant T_{is}^{L} [h] :

$$\forall i \in \{1,...,I\}; \forall s \in \{1,...,S\}\ Prodloc_{is} = \frac{B_{is}}{T_{is}^{L}} \tag{8}$$

- ✓ Le temps de cycle limitant est le plus grand des temps de cycle en produit i dans le sous-procédé s (J_s étant l'ensemble des étapes discontinues dans le sous-procédé) :

$$\forall i \in \{1,...,I\}; \forall s \in \{1,...,S\}\ T_{is}^{L} = \max_{j \in J_s}\left[T_{ij}\right] \tag{9}$$

- ✓ Le temps de cycle du produit *i* dans l'étape discontinue *j* comprend le temps opératoire dans l'étape p_{ij} [h] et les temps de remplissage et de vidange, qui correspondent aux temps de traitement dans des sous-trains semi-continus Θ_{it} [h] respectivement en aval et en amont de l'étape discontinue considérée. Ce temps est à diviser par le nombre d'équipements en parallèle, travaillant hors-phase :

$$\forall i \in \{1,...,I\}; \forall j \in \{1,...,J\}\ T_{ij} = \frac{\Theta_{it} + \Theta_{i(t+1)} + p_{ij}}{m_j} \tag{10}$$

Dans (10), les indices *t* et *t+1* représentent ici les sous-trains semi-continus entourant l'étape discontinue *j* dans le sous-procédé *s*.

- ✓ Le temps opératoire du produit *i* dans l'étape discontinue *j* est calculé selon l'expression suivante :

$$\forall i \in \{1,...,I\}; \forall j \in \{1,...,J_s\}; \forall s \in \{1,...,S\}\ p_{ij} = p_{ij}^0 + g_{ij} B_{is}^{d_{ij}} \tag{11}$$

où p_{ij}^0, g_{ij} et d_{ij} sont des données propres aux équipements de l'atelier.

- ✓ Le temps de traitement du produit *i* dans le sous-train semi-continu *t* est limité par le plus grand des temps de traitement d'un lot de *i* dans les étapes semi-continues qui composent ce sous train :

$$\forall i \in \{1,...,I\}; \forall t \in \{1,...,T\}\ \Theta_{it} = \max_{k \in K_t} [\theta_{ik}] \tag{12}$$

où *T* est le nombre de sous-trains semi-continus, K_t étant l'ensemble des étapes semi-continues dans le train *t*.

- ✓ Le temps de traitement du produit *i* dans l'étape semi-continue *k* dépend de son taux opératoire R_k, du nombre d'équipements en parallèle n_k, de la taille des lots B_{is} le traversant, et du facteur D_{ik}. D_{ik} [L.kg^{-1}]est assimilable à un facteur de taille tel qu'il est défini dans le paragraphe suivant.

$$\forall i \in \{1,...,I\}; \forall k \in \{1,...,K_s\}; \forall s \in \{1,...,S\}\ \theta_{ik} = \frac{B_{is} D_{ik}}{R_k n_k} \tag{13}$$

où K_s est l'ensemble des étapes semi-continues dans le sous-procédé *s*.

✓ La taille de lot B_{is} est définie comme la plus petite quantité de produit i contenue dans un équipement de volume V_j appartenant au sous-procédé s :

$$\forall i \in \{1,...,I\}; \forall s \in \{1,...,S\}\ B_{is} = \min_{j \in J_s} \left[\frac{V_j}{S_{ij}} \right] \tag{14}$$

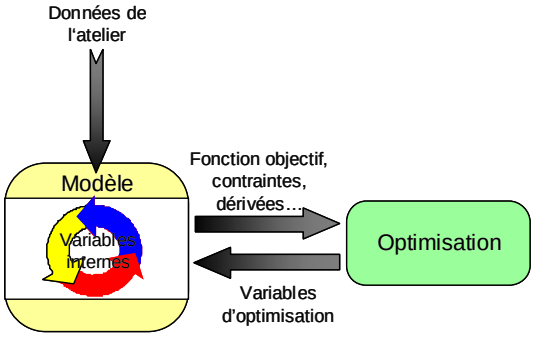
La quantité de matière est calculée grâce au facteur de taille S_{ij} [L.kg^{-1}]. Ce paramètre, propre à chaque équipement, représente le volume nécessaire de l'appareil pour traiter une unité de masse de produit i.

Il ne reste plus, pour compléter le modèle, qu'à calculer le volume des bacs de stockage intermédiaire. Celui-ci est estimé par la différence maximale entre la taille des lots traités par les deux sous-procédés que le bac de stockage sépare, soit selon l'expression suivante :

$$\forall s \in \{1,...,S-1\}\ V_s = \max_{i \in I} \left[Prod_i S_{is} \left(T^L_{is} + T^L_{i(s+1)} - \Theta_{it} - \Theta_{i(t+1)} \right) \right] \tag{15}$$

Les indices t et $t+1$ représentent ici les trains semi-continus situés en amont et en aval du bac de stockage.

L'interaction entre le modèle et un optimiseur quelconque est illustrée par la figure 2, qui récapitule également tous les paramètres du problème. Le problème résultant de la formulation décrite dans la partie précédente est bien de type MINLP. Comme cela a été signalé par Wang et al. [WAN96], il est de type NP-difficile, ce qui implique des temps de calcul croissant de manière exponentielle au fur et à mesure de l'augmentation de la taille des instances traitées.



Variables d'optimisation	Variables internes	Données de l'atelier
✓ Volume des équipements V_j } **continues** ✓ Capacité de traitement R_k } **continues** ✓ Nombre d'équipements m_j } **discrètes** ✓ Nombre d'équipements n_k } **discrètes**	✓ Productivités globales et locales ✓ Taille des lots de produit ✓ Temps opératoires pour les étapes discontinues et semi-continues ✓ Temps de cycle limitant	✓ Recette, horizon de temps et demandes ✓ Coefficients de coût ✓ Facteurs de taille ✓ Coefficients de calcul des temps opératoires

Figure 2 – Récapitulatif du fonctionnement et des paramètres

3 – Traitement du problème résultant

L'intégration du modèle défini précédemment dans une boucle d'optimisation est une tâche délicate. Cette remarque est d'autant plus vérifiable dans le cas de méthodes mathématiques. En revanche, il est vrai que l'adaptation à un algorithme génétique se fait de manière plus souple, comme cela est montré par la suite.

3.1 – Méthodes de Programmation Mathématique

Comme cela a déjà été dit dans le chapitre décrivant les méthodes d'optimisation, un effort doit être fait pour formuler le problème de manière pertinente pour faciliter ensuite la mise en oeuvre du module de résolution. Toutes les subtilités de la formulation employée au sein de l'environnement *GAMS* ne seront évidemment pas évoquées dans ce qui suit. Par contre, il est intéressant d'étudier les propriétés mathématiques vérifiées par le modèle, notamment en termes de convexité : cet aspect est fondamental pour l'utilisation de méthodes de Programmation Mathématique. En effet, il permettra de juger de la pertinence des résultats fournis puisque la convexité conditionne l'obtention de l'optimum global.

3.1.1 – Analyse de la convexité

Etonnamment, la preuve de la non-convexité du modèle utilisé n'a pas fait l'objet de travaux reportés dans la littérature. Un aparté est ainsi consacré à sa démonstration, qui se restreindra à l'étude de la fonction objectif puisqu'il est possible de considérer que le modèle est non-convexe à partir du moment où c'est le cas pour la fonction objectif.

Soit $F(X)=\sum_{j=1}^{J} m_j a_j V_j^{\alpha_j} + \sum_{k=1}^{K} n_k b_k R_k^{\beta_k} + \sum_{s=1}^{S-1} c_s V_s^{\gamma_s}$, $X = [Vj, Rk, mj, nk]^T$, $j \in \{1,\ldots,J\}$, $k \in \{1,\ldots,K\}$.

Soient également les deux propriétés suivantes :

Propriété 1 : la fonction $F : E \subset \mathcal{R}^n \rightarrow \mathcal{R}$, de classe C^2, est convexe sur son ensemble de définition E si et seulement si sa matrice hessienne H_F est semi-définie positive.

Propriété 2 : la matrice carrée symétrique M d'ordre m et de rang p est semi-définie positive si et seulement si toutes ses p valeurs propres sont positives ou nulles.

Ainsi, il suffit de trouver au moins une valeur propre négative de la matrice hessienne H_F pour prouver que F est non-convexe. Le calcul de H_F est disponible en annexe 1. Vue la forme de la matrice, la détermination analytique de ses valeurs propres est impossible. La

routine *EVLSF* de la librairie *IMSL®* a été utilisée pour calculer les valeurs numériques des valeurs propres λ de H_F. Quelques tests ont été effectués en faisant varier X, puisque H_F est exprimée en fonction des variables du problème. Un exemple est donné en annexe 1. Dès le premier essai, des valeurs propres négatives ont été calculées. La non-convexité de la fonction objectif F et, donc, du modèle utilisé est démontrée.

3.1.2 – Formulation modifiée dans GAMS

Etant donné qu'il est démontré que le modèle employé est non-convexe, un effort de convexification est nécessaire pour aider au bon fonctionnement des méthodes de Programmation Mathématique mises en oeuvre au sein de *GAMS*. La formulation modifiée proposée par Kocis et Grossmann [KOC88] a été reprise dans ce travail. Elle consiste en un simple changement de variable logarithmique, i.e. $var_i = ln(VAR_i)$. Cette formule est appliquée aux variables suivantes dans le modèle décrit précédemment :

- $v_j' = ln(V_j)$;
- $m_j' = ln(m_j)$;
- $r_k' = ln(R_k)$;
- $m_k' = ln(m_k)$;
- $b_{is}' = ln(B_{is})$;
- $r_k' = ln(R_k)$;
- $\theta_{ik}' = ln(\theta_{ik})$;
- $\Theta_{it}' = ln(\Theta_{it})$;
- $t_{ij}' = ln(T_{ij})$;
- $t_{is}^L' = ln(T_{is}^L)$;
- $v_s' = ln(V_s)$;
- $prodloc_{is}' = ln(Prodloc_{is})$;
- $prod_i' = ln(Prod_i)$;
- $h_i' = ln(H_i)$.

Ce changement de variable a un effet sur les équations (1) et (6) à (15), qui deviennent respectivement les équations (16) à (26) :

$$Min\ Coût = \sum_{j=1}^{J} a_j exp(m_j' + \alpha_j v_j') + \sum_{k=1}^{K} b_k exp(n_k' + \beta_k n_k') + \sum_{s=1}^{S-1} c_s exp(\gamma_s v_s') \qquad (16)$$

$$\sum_{i=1}^{I} H_i = \sum_{i=1}^{I} [Q_i - prod_i'] \leq H \qquad (17)$$

$$\forall i \in \{1,..,I\}\ prod_i' \leq prodloc_{is}' \qquad (18)$$

$$\forall i\in\{1,...,I\};\ \forall s\in\{1,...,S\}\ prodloc_{is}' = b_{is}' - t_{is}^{L'} \tag{19}$$

$$\forall i\in\{1,...,I\};\ \forall s\in\{1,...,S\}\ t_{is}^{L'} \geq t_{ij}' \tag{20}$$

$$\forall i\in\{1,...,I\};\ \forall j\in\{1,...,J\}\ exp(m_j + T_{ij}) = exp(\Theta_{it}') + exp(\Theta_{i(t+1)}') + exp(p_{ij}') \tag{21}$$

$$\forall i\in\{1,...,I\};\ \forall j\in\{1,...,J_s\};\ \forall s\in\{1,...,S\}\ exp(p_{ij}') = p_{ij}^0 + g_{ij}\, exp(d_{ij} b_{is}') \tag{22}$$

$$\forall i\in\{1,...,I\};\ \forall t\in\{1,...,T\}\ \Theta_{it}' \geq \theta_k' \tag{23}$$

$$\forall i\in\{1,...,I\};\ \forall k\in\{1,...,K_s\};\ \forall s\in\{1,...,S\}\ n_k' + n_k' = b_{is}' + ln(D_{ik}) - \theta_k' \tag{24}$$

$$\forall i\in\{1,...,I\};\ \forall s\in\{1,...,S\}\ v_j' \geq ln(S_{ij}) + b_{is}' \tag{25}$$

$$\forall s\in\{1,...,S-1\}\ exp(v_s') \geq exp(prod_i')S_{is}\left(exp(t_{is}^{L'}) + exp(t_{i(s+1)}^{L}{}') - exp(\Theta_{it}') - exp(\Theta_{i(t+1)}')\right) \tag{26}$$

Comme cela a été dit, la preuve de la non-convexité est facile à établir selon la méthode employée plus haut puisqu'un unique exemple de valeur propre négative est suffisant. Par contre, la démarche inverse est autrement plus difficile à réaliser.

Cependant, à l'instar de l'étude faite par Kocis et Grossmann [KOC88], l'examen de la reformulation présentée ci-dessus met en évidence que, à l'exception des termes exponentiels qui sont trivialement convexes, toutes les équations sont complètement linéaires. Ainsi, le modèle correspondant est indéniablement convexe lui-aussi, dans le cas où les coefficients précédant les termes exponentiels soient positifs(ce qui ne sera pas toujours vérifié dans les problèmes étudiés par la suite).

L'examen effectué montre que les solveurs mathématiques utilisés avec *GAMS* ne sont pas capables de trouver de solution sans cet effort sur la formulation. Par ailleurs, il est également nécessaire d'évaluer avec finesse les bornes de chacune des variables internes du modèle (tailles des lots ou temps de traitement, par exemple) pour éviter les problèmes d'infaisabilité du problème RMINLP initial (problème MINLP relaxé). Une estimation, faite parallèlement sur un tableur de calcul, de l'intervalle à l'intérieur duquel varie chaque variable permet de dépasser cette difficulté tout en s'assurant de ne pas couper des zones de l'espace de recherche, ce qui pourrait faire perdre l'optimum global.

3.2 – Algorithme Génétique

L'adaptation du modèle à l'algorithme génétique est bien plus simple. Par son appartenance à la classe des métaheuristiques, il ne nécessite que l'évaluation du critère. Le calcul de la fonction objectif peut donc être complètement découplé de la procédure d'optimisation. La seule information spécifique requise par l'algorithme génétique concerne la nature des variables d'optimisation impliquées dans le problème ; cette dernière aura une influence logique sur le codage. Ce point sera largement développé dans le chapitre suivant.

La seule remarque notable est que, dans tous les exemples abordés, tous les trains semi-continus sont composés d'une seule étape. En conséquence, la variable Θ_{it} a été supprimée de la formulation, et remplacée directement par θ_{ik} dans l'équation (10). Compte tenu de cette simplification, le modèle final a été implémenté de manière totalement générique dans l'algorithme : il suffit de donner le nombre d'étapes, la position des bacs de stockage et les données relatives à ces équipements (coefficients de coût et de calcul du temps opératoire, facteurs de taille) pour décrire complètement l'atelier.

Finalement, tous les aspects relatifs à la formulation et à son adaptation aux méthodes employées ont été détaillés de façon précise. Il reste à définir le jeu de problèmes sur lequel ont été testés *DICOPT++*, *SBB* et l'algorithme génétique.

4 – Exemples traités

Pour répondre aux exigences posées par la problématique de l'étude, il est nécessaire de construire presque entièrement le jeu d'exemples sur lequel ont été effectués les calculs présentés dans le chapitre suivant. Seulement presque, car les ordres de grandeur des données n'ont pas été choisis de manière totalement intuitive. En effet, la construction des exemples est basée sur deux problèmes existants et résolus dans la littérature.

Le premier (atelier 1), proposé dans Kocis et Grossmann [KOC88], est utilisé pour valider le bon comportement de l'algorithme OA/ER, auquel est ajoutée la stratégie de résolution en deux étapes pour le traitement des non-convexités repérées par des tests locaux et globaux (cf. chapitre 1). Dans le même article est également proposée la reformulation convexe qui est employée ici pour l'optimisation par des méthodes mathématiques. L'exemple traité est relativement simple : il est constitué de trois étapes discontinues (mélangeur, réacteur et cyclone) et ne compte ni équipement semi-continu, ni bac de stockage intermédiaire. La synthèse de deux produits est considérée. L'optimum déterminé par Kocis et Grossmann vaut 285506 [$]. Cette valeur ne sera cependant pas retrouvée dans cette étude

puisque le problème a été légèrement modifié pour introduire la dépendance des temps de traitement vis-à-vis de la taille de lots de produits, exprimée dans l'équation (11). En outre, des bornes plus larges ont été admises pour les tailles des équipements.

Le deuxième exemple (atelier 4) a été proposé par Patel et al. [PAT91] et résolu au moyen d'une méthode de recuit simulé, puis repris successivement dans trois études par Wang et al. ([WAN96], [WAN99], [WAN02]) avec trois métaheuristiques : un algorithme génétique, une méthode tabou et une méthode de colonies de fourmis. Ce problème comporte quatre étapes discontinues et six étapes semi-continues. L'atelier est divisé en deux sous-procédés par un bac de stockage intermédiaire et doit permettre la synthèse de trois produits. La meilleure solution trouvée, déterminée avec un algorithme génétique, vaut 362130 [$] mais viole légèrement la contrainte de temps [WAN96]. La figure 3 illustre les deux ateliers évoqués.

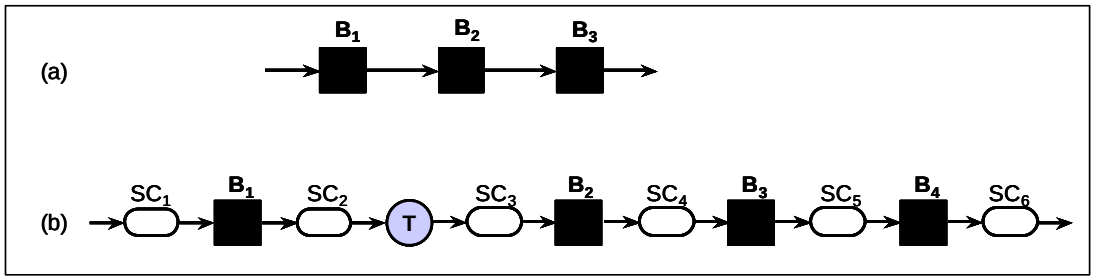


Figure 3 - Représentation des deux ateliers réutilisés dans cette étude
a) Kocis et Grossmann [KOC88]
b) Patel et al. [PAT91]

Vu leur similarité, il a été possible de se baser sur les données de ces deux problèmes pour construire un jeu d'exemples de complexité croissante. Celui-ci est résumé dans le tableau 2. Tout d'abord, deux ateliers de taille intermédiaire (par rapport à ceux existants) ont été créés. Ils ressemblent à l'exemple de Kocis et Grossmann [KOC88], en y ajoutant simplement des équipements semi-continus (atelier 2) puis un bac de stockage (atelier 3). Les nombres d'étapes discontinues et de produits à synthétiser sont conservés.

Par la suite, des procédés plus complexes ont été envisagés. La taille de ces derniers est supérieure à celle du problème de Patel et al. [PAT91] et va jusqu'à compter 45 étapes discontinues et 60 étapes semi-continues (atelier 11), ce qui dépasse les limites d'un atelier réaliste. Dans tous ces derniers cas, le nombre de produits à synthétiser est égal à trois. Les ateliers 5, 6 et 7 ont été créés en augmentant de manière intuitive le nombre d'étapes discontinues, d'étapes semi-continues et de sous-procédés. Finalement, les ateliers 8 à 11 ont été construits par des juxtapositions de l'atelier 7, mises bout à bout et séparées par un bac de stockage. Ces derniers exemples ont été mis en oeuvre à des fins uniquement numériques,

plus précisément pour tester les limites des capacités de résolution des méthodes d'optimisation. Ces ateliers se situent en fait au-delà de la limite supérieure d'une taille industriellement réaliste. Les données relatives à tous ces exemples sont disponibles dans l'annexe 2 de ce chapitre.

Atelier n°	Nombre total d'étapes	Stockage	Nombre de produits	Combinatoire
1	3	0	2	$2{,}70.10^{1}$
2	7	0	2	$2{,}19.10^{3}$
3	8	1	2	$6{,}56.10^{3}$
4	10	1	3	$5{,}90.10^{4}$
5	14	1	3	$4{,}78.10^{6}$
6	17	2	3	$1{,}29.10^{8}$
7	21	2	3	$1{,}05.10^{10}$
8	42	5	3	$1{,}09.10^{20}$
9	63	8	3	$1{,}14.10^{30}$
10	84	11	3	$1{,}20.10^{40}$
11	105	14	3	$1{,}25.10^{50}$

Tableau 2 – Exemples abordés et leurs caractéristiques

Le nombre de variables de chaque exemple peut être déduit de la deuxième colonne du tableau 2. En effet, d'après la formulation employée, chaque étape opératoire induit une variable continue (volume ou taux opératoire) et une variable discrète (nombre d'équipements en parallèle). Par ailleurs, la dernière colonne est un indicateur approximatif de la complexité de chacun des exemples, calculée en fonction du nombre de variables discrètes. La représentation de ces dernières diffère selon la méthode d'optimisation employée. Mais au final, la combinatoire reste la même, comme cela est démontré ici.

En effet, en anticipant sur la description du codage employé au sein l'AG, les variables entières sont codées telles quelles, sur un locus adoptant la valeur de la variable. Bien entendu, les variables continues ne sont pas prises en considération dans le calcul de la combinatoire, étant donné que la complexité associée dépendrait de la précision souhaitée sur ces dernières. Compte tenu du fait qu'il peut y avoir au maximum trois équipements en parallèle par étage, la combinatoire pour l'AG vaut $3^{NEtapes}$. En revanche, la combinatoire associée aux méthodes de Programmation Mathématique repose sur des variables binaires. Leur présence s'explique par le mode de représentation des variables entières dans *GAMS*. Ainsi, pour une étape discontinue par exemple, le nombre d'équipements m_j s'écrit :

$$m_j = \sum_{k=1}^{3} y_{jk}.coeff_k \qquad \text{avec} \begin{cases} y_{jk} \in \{0,1\},\ j=1,\ NEtapes \\ coeff_k = k,\ k=1,\ 3 \end{cases} \tag{27}$$

En conséquence, il y a trois variables binaires par étape, ce qui signifierait une combinatoire équivalente à $2^{Nb\ var.\ binaires} = 2^{3xNEtapes}$ pour les méthodes mathématiques. Mais pour éviter des solutions sans signification physique, il est indispensable d'ajouter à la formulation précédente une contrainte sur les variables binaires de chaque étape :

$$\sum_{k=1}^{3} y_{jk} = 1, \qquad \text{pour } j=1,\ldots, NEtapes \tag{28}$$

En effet, sans cette contrainte supplémentaire, il serait possible d'avoir, par exemple, à la fois $y_{j1} = y_{j2} = 1$, c'est-à-dire en même temps un et deux équipements en parallèle à l'étape j. Par conséquent, le choix représentatif déterminant la combinatoire porte sur laquelle des trois variables binaires est égale à un pour chaque étape, soit $3^{NEtapes}$. Les combinatoires sont donc identiques quelle que soit la méthode d'optimisation employée.

Cependant, il sera démontré au cours de l'étude suivante que la complexité des problèmes ne peut être seulement expliquée par le nombre de variables, puisque des exemples de relativement petite taille ont parfois posé plus de difficultés de résolution que d'autres, de plus grande taille.

Conclusion

Ce chapitre a tout d'abord fourni les bases pour établir le formalisme retenu dans cette étude. Il s'agit d'un modèle classique de conception d'atelier discontinu multiproduit. Il exprime la minimisation d'un critère d'investissement, soumise à une contrainte de production en un temps imparti. Tant les objectifs que le mode de représentation sont couramment retrouvés dans la littérature : les seules spécificités notables du modèle employé sont la présence d'équipements semi-continus et de bacs de stockage intermédiaire. Puis, les adaptations de la formulation, propres aux modes de fonctionnement respectifs des deux types de méthodes étudiées, ont également été définies. Enfin, le jeu d'exemples sur lequel reposent les calculs effectués par la suite a été présenté.

La formulation résultante est un problème de type MINLP, combinant de plus un aspect NP-difficile. Il a par ailleurs été prouvé qu'elle est non-convexe, ce qui a impliqué une reformulation pour les méthodes déterministes. En conséquence, le modèle adopté fournit effectivement un cadre intéressant pour la comparaison des performances de méthodes d'optimisation. Sur la panoplie d'exemples formulés de cette manière, les deux méthodes MP et l'algorithme génétique seront ainsi confrontés à un problème associant plusieurs types de difficultés.

Il faudra d'ailleurs garder en mémoire l'effort de formulation indispensable au bon fonctionnement des méthodes mathématiques, et non à celui de l'algorithme génétique. Ainsi, quelle que soit l'efficacité des deux premières techniques, l'étude des fonctions mises en jeu et de la pertinence de la formulation qu'elles impliquent nécessairement sera a prendre en considération.

Le cadre de l'étude étant dès lors complètement défini, il est possible d'aborder la résolution des onze problèmes par les deux méthodes déterministes et l'algorithme génétique.

CHAPITRE 2
ANNEXE 1

ANALYSE DE LA CONVEXITE

La vérification de la convexité ou de la non-convexité nécessite, comme cela a été vu, le calcul de la matrice hessienne H_F de la fonction objectif F. Cette dernière s'écrit sous la forme :

$$F(X) = \sum_{j=1}^{J} m_j a_j V_j^{\alpha_j} + \sum_{k=1}^{K} n_k b_k R_k^{\beta_k} + \sum_{s=1}^{S-1} c_s V_s^{\gamma_s} \text{ , où } X = [Vj, Rk, mj, nk]^T, \ \forall j, \ \forall k.$$

Sa forme générale est donc, quel que soit l'atelier considéré, la suivante :

$$H_F(X) = \begin{pmatrix} \alpha_j(\alpha_j-1)a_j m_j V_j^{\alpha_j-2} & \alpha_j a_j V_j^{\alpha_j-1} & 0 & 0 \\ \alpha_j a_j V_j^{\alpha_j-1} & 0 & 0 & 0 \\ 0 & 0 & \beta_k(\beta_k-1)b_k n_k R_k^{\beta_k-2} & \beta_k b_k R_k^{\beta_k-1} \\ 0 & 0 & \beta_k b_k R_k^{\beta_k-1} & 0 \end{pmatrix}$$

Cependant, pour simplifier cette formulation, la matrice H_F est exprimée pour le premier atelier, ne comportant aucun équipement semi-continu. Il semble en effet trivial que si la fonction objectif n'est pas convexe pour l'exemple le plus simple, l'ajout de termes a priori similaires ne changera pas cette caractéristique. Ainsi, pour le premier atelier, composé de trois étapes discontinues, la matrice hessienne de la fonction objectif s'écrit :

$$H_{F,Ex1}(X) = \begin{pmatrix} \alpha_j(\alpha_j-1)a_j m_j V_j^{\alpha_j-2} & \alpha_j a_j V_j^{\alpha_j-1} \\ \alpha_j a_j V_j^{\alpha_j-1} & 0 \end{pmatrix}$$

Remarque : la matrice hessienne est symétrique par définition, donc ses valeurs propres sont toutes réelles.

La recherche des valeurs propres λ de $H_{F\text{-}Ex1}$ nécessite la résolution de l'équation :

$$H_{F\text{-}Ex1}.Y = \lambda.Y \qquad (1)$$

où Y est le vecteur propre. Cette équation est résolue de manière numérique grâce à la routine *EVLSF* de la librairie *IMSL®*. Celle-ci nécessite seulement la dimension et les coefficients réels de la matrice. Par conséquent, il est nécessaire de fixer des valeurs numériques pour les vecteur des variables X. Plusieurs tests ont été effectués, notamment aux bornes inférieures et supérieures des intervalles de variation de chaque variable V_j et m_j. Un exemple est donné ci-dessous.

Soit $X_0 = [4875, 4875, 4875, 3, 3, 3]$, i.e. les variables continues sont fixées au milieu de leurs bornes et les nombres d'équipements fixés à leur maximum possible. Alors la matrice hessienne vaut :

$$H_{F\text{-}Ex1}(X_0) = \begin{pmatrix} -1{,}2.10^{-3} & 0 & 0 & 5{,}022 & 0 & 0 \\ 0 & -1{,}2.10^{-3} & 0 & 0 & 5{,}022 & 0 \\ 0 & 0 & -1{,}2.10^{-3} & 0 & 0 & 5{,}022 \\ 5{,}022 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5{,}022 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5{,}022 & 0 & 0 & 0 \end{pmatrix}$$

La résolution de l'équation (1) par *EVLSF* donne $Y_0 = [$-5,022, -5,022, -5,022 , 5,022, 5,022, 5,022$]$. Ainsi, sur six valeurs propres trois sont négatives : la non-convexité de la fonction objectif, donc du modèle de représentation de l'atelier 1, est démontrée. En première approximation, cette caractéristique sera étendue au modèle global, indépendamment de l'atelier considéré.

CHAPITRE 2
ANNEXE 2

DONNEES DU JEU D'EXEMPLES

Toutes les données relatives au jeu d'exemples proposé dans le chapitre 2 et utilisé pour les calculs par la suite sont détaillées dans cette annexe. Pour l'exemple 1, les données du problème traité dans [KOC88] ont été conservées, sauf en ce qui concerne les bornes des variables continues et les coefficients de temps de traitement. L'exemple 4, proposé dans [PAT91], a été conservé tel quel. Mis à part ces deux cas particuliers, toutes les valeurs ont été inventées. Cependant, certaines données sont communes à tous les exemples. C'est le cas des coefficients de coût disponibles dans le tableau 1 du chapitre 2. De la même manière, quelle que soit l'instance, restent inchangés :

- l'horizon de temps imparti pour la production des I produits : H = 6000 [h] ;
- les facteurs de taille pour les bacs stockage intermédiaire : S_{is} = 1 [L.kg^{-1}] ;
- les capacités de traitement dans les étapes semi-continues : D_{ik} = 1 [L.kg^{-1}.h^{-1}].

Il reste à donner pour chaque exemple des informations sur la structure des ateliers : nombres d'étapes discontinues J et semi-continues K, nombre de sous-procédés S, demande en chaque produit Q_i et recettes. Enfin, seuls manquent les données relatives aux étapes discontinues : facteurs de taille S_{ij} et coefficients de calcul des temps de traitement dans les équipements discontinus (cf. équation (11) du chapitre 2).

Exemple 1

Données générales		Recette par sous-procédé	
I	2	s = 1	{B1,B2,B3}
J	3		
K	0		
S	1		
Q [kg]	[150.10^3, 200.10^3]		

		B1	B2	B3
S_{ij}	i = 1	2	3	4
	i = 2	4	6	3
p_{ij}^0	i = 1	6	16	3
	i = 2	7,5	8	2,25
g_{ij}	i = 1	0,4	0,35	0,15
	i = 2	0,6	0,5	0,2
d_{ij}	i = 1	0,4	0,3	0,2
	i = 2	0,4	0,3	0,2

Exemple 2

Données générales		Recette par sous-procédé	
I	2	$s = 1$	{SC1, B1, SC2, B2, SC3, B3, SC4}
J	3		
K	3		
S	1		
Q [kg]	$[150.10^3, 200.10^3]$		

		B1	B2	B3
S_{ij}	$i = 1$	2	3	4
	$i = 2$	4	6	3
p_{ij}^0	$i = 1$	6	16	3
	$i = 2$	7,5	8	2,25
g_{ij}	$i = 1$	0,4	0,35	0,15
	$i = 2$	0,6	0,5	0,2
d_{ij}	i = 1	0,4	0,3	0,2
	i = 2	0,4	0,3	0,2

Exemple 3

Données générales		Recette par sous-procédé	
I	2	$s = 1$	{SC1, B1, SC2}
J	3	$s = 2$	{SC3, B2, SC4, B3, SC5}
K	3		
S	2		
Q [kg]	$[150.10^3, 200.10^3]$		

		B1	B2	B3
S_{ij}	$i = 1$	2	3	4
	$i = 2$	4	6	3
p_{ij}^0	$i = 1$	6	16	3
	$i = 2$	7,5	8	2,25
g_{ij}	$i = 1$	0,4	0,35	0,15
	$i = 2$	0,6	0,5	0,2
d_{ij}	i = 1	0,4	0,3	0,2
	i = 2	0,4	0,3	0,2

Exemple 4

Données générales		Recette par sous-procédé	
I	3	$s = 1$	{SC1, B1, SC2}
J	4	$s = 2$	{SC3, B2, SC4, B3, SC5, B4, SC6}
K	6		
S	2		
Q [kg]	[437.10^3, 324.10^3, 258.10^3]		

		B1	B2	B3	B4
S_{ij}	$i = 1$	8,28	9,70	2,95	6,57
	$i = 2$	5,58	8,09	3,27	6,17
	$i = 3$	2,34	10,30	5,70	5,98
p_{ij}^0	$i = 1$	1,15	9,86	5,28	1,20
	$i = 2$	5,95	7,01	7,00	1,08
	$i = 3$	3,96	6,01	5,13	0,66
g_{ij}	$i = 1$	0,20	0,24	0,40	0,50
	$i = 2$	0,15	0,35	0,70	0,42
	$i = 3$	0,34	0,50	0,85	0,30
d_{ij}	$i = 1$	0,40	0,33	0,30	0,20
	$i = 2$	0,40	0,33	0,30	0,20
	$i = 3$	0,40	0,33	0,30	0,20

Exemple 5

Données générales		Recette par sous-procédé	
I	3	$s = 1$	{SC1, B1, SC2, B2, SC3}
J	6		
K	8		
S	2	$s = 2$	{SC4, B3, SC5, B4, SC6, B5, SC7, B6, SC8}
Q [kg]	[437.10^3, 324.10^3, 258.10^3]		

		B1	B2	B3	B4	B5	B6
S_{ij}	$i = 1$	8,28	6,92	9,70	2,95	6,57	10,60
	$i = 2$	5,58	8,03	8,09	3,27	6,17	6,57
	$i = 3$	2,34	9,19	10,30	5,70	5,98	3,14
p_{ij}^0	$i = 1$	1,15	3,98	9,86	5,28	1,20	3,57
	$i = 2$	5,95	7,52	7,01	7,00	1,08	5,78
	$i = 3$	3,96	5,07	6,01	5,13	0,66	4,37
g_{ij}	$i = 1$	0,20	0,36	0,24	0,40	0,50	0,40
	$i = 2$	0,15	0,50	0,35	0,70	0,42	0,38
	$i = 3$	0,34	0,64	0,50	0,85	0,30	0,22
d_{ij}	$i = 1$	0,40	0,29	0,33	0,30	0,20	0,35
	$i = 2$	0,40	0,29	0,33	0,30	0,20	0,35
	$i = 3$	0,40	0,29	0,33	0,30	0,20	0,35

Exemple 6

Données générales		Recette par sous-procédé	
I	3	$s = 1$	{SC1, B1, SC2, B2, SC3}
J	7		
K	10	$s = 2$	{SC4, B3, SC5, B4, SC6, B5, SC7}
S	3		
Q [kg]	[437.10^3, 324.10^3, 258.10^3]	$s = 3$	{SC8, B6, SC9, B7, SC10}

		B1	B2	B3	B4	B5	B6	B7
S_{ij}	$i = 1$	8,28	6,92	9,70	2,95	10,60	7,59	6,74
	$i = 2$	5,58	8,03	8,09	3,27	6,57	5,23	4,21
	$i = 3$	2,34	9,19	10,30	5,70	3,14	7,41	3,92
p_{ij}^0	$i = 1$	1,15	3,98	9,86	5,28	3,57	6,25	2,22
	$i = 2$	5,95	7,52	7,01	7,00	5,78	4,38	4,57
	$i = 3$	3,96	5,07	6,01	5,13	4,37	3,86	1,39
g_{ij}	$i = 1$	0,20	0,36	0,24	0,40	0,40	0,47	0,16
	$i = 2$	0,15	0,50	0,35	0,70	0,38	0,29	0,25
	$i = 3$	0,34	0,64	0,50	0,85	0,22	0,32	0,27
d_{ij}	$i = 1$	0,40	0,29	0,33	0,30	0,35	0,39	0,18
	$i = 2$	0,40	0,29	0,33	0,30	0,35	0,39	0,18
	$i = 3$	0,40	0,29	0,33	0,30	0,35	0,39	0,18

Exemple 7

Données générales		Recette par sous-procédé	
I	3	$s = 1$	{SC1, B1, SC2, B2, SC3}
J	9		
K	12	$s = 2$	{SC4, B3, SC5, B4, SC6, B5, SC7, B6, SC8}
S	3		
Q [kg]	[437.10^3, 324.10^3, 258.10^3]	$s = 3$	{SC9, B7, SC10, B8, SC11, B9, SC12}

		B1	B2	B3	B4	B5	B6	B7	B8	B9
S_{ij}	$i = 1$	8,28	6,92	9,70	2,95	6,57	10,60	7,59	6,74	8,90
	$i = 2$	5,58	8,03	8,09	3,27	6,17	6,57	5,23	4,21	5,35
	$i = 3$	2,34	9,19	10,30	5,70	5,98	3,14	7,41	3,92	6,63
p_{ij}^0	$i = 1$	1,15	3,98	9,86	5,28	1,20	3,57	6,25	2,22	10,00
	$i = 2$	5,95	7,52	7,01	7,00	1,08	5,78	4,38	4,57	4,02
	$i = 3$	3,96	5,07	6,01	5,13	0,66	4,37	3,86	1,39	5,89
g_{ij}	$i = 1$	0,20	0,36	0,24	0,40	0,50	0,40	0,47	0,16	0,68
	$i = 2$	0,15	0,50	0,35	0,70	0,42	0,38	0,29	0,25	0,51
	$i = 3$	0,34	0,64	0,50	0,85	0,30	0,22	0,32	0,27	0,45
d_{ij}	$i = 1$	0,40	0,29	0,33	0,30	0,20	0,35	0,39	0,18	0,26
	$i = 2$	0,40	0,29	0,33	0,30	0,20	0,35	0,39	0,18	0,26
	$i = 3$	0,40	0,29	0,33	0,30	0,20	0,35	0,39	0,18	0,26

Exemples 8 à 11

Les exemples 8 à 11 sont construits comme de simples juxtapositions de l'exemple 7 : à chaque fois, trois sous-procédés dont la structure et les données sont identiques à celles de l'exemple 7 sont ajoutés en fin de procédé. Le tableau suivant illustre ces derniers cas.

	Données générales		Recette par sous-procédé	
Exemple 8	I	3	$s = 4$	{SC13, B10, SC14, B11, SC15}
	J	18	$s = 5$	{SC16, B12, SC17, B13, SC18, B14, SC19, B15, SC20}
	K	24		
	S	3	$s = 6$	{SC21, B16, SC22, B17, SC23, B18, SC24}
	Q [kg]	[437.10^3, 324.10^3, 258.10^3]		
Exemple 9	I	3	$s = 7$	{SC25, B19, SC26, B20, SC27}
	J	27	$s = 8$	{SC28, B21, SC29, B22, SC30, B23, SC31, B24, SC32}
	K	36		
	S	3	$s = 9$	{SC33, B25, SC34, B26, SC35, B27, SC36}
	Q [kg]	[437.10^3, 324.10^3, 258.10^3]		
Exemple 10	I	3	$s = 10$	{SC37, B28, SC38, B29, SC39}
	J	36	$s = 11$	{SC40, B30, SC41, B31, SC42, B32, SC43, B33, SC44}
	K	48		
	S	3	$s = 12$	{SC45, B34, SC46, B35, SC47, B36, SC48}
	Q [kg]	[437.10^3, 324.10^3, 258.10^3]		
Exemple 11	I	3	$s = 13$	{SC49, B37, SC50, B38, SC51}
	J	45	$s = 14$	{SC52, B39, SC53, B40, SC54, B41, SC55, B426, SC56}
	K	60		
	S	3	$s = 15$	{SC57, B43, SC58, B44, SC59, B45, SC60}
	Q [kg]	[437.10^3, 324.10^3, 258.10^3]		

CHAPITRE 3

CALCULS EN VARIABLES MIXTES ET INTERPRETATIONS

Comme cela a été mentionné dans l'introduction de ce mémoire, les deux axes de la problématique de ces travaux concernent : (i) d'une part, l'étude comparative des performances des trois méthodes d'optimisation évoquées ; (ii) d'autre part, l'adaptation des paramètres et procédures propres au fonctionnement de la méthode stochastique employée, i.e. l'algorithme génétique, de manière à optimiser son efficacité face à des problèmes contraints en variables mixtes.

Le premier objectif est traité en résolvant le jeu de onze exemples, construit et détaillé dans le chapitre précédent, au moyen des trois méthodes. Pour chaque exemple, les résultats obtenus sont comparés sur la base de critères naturels : qualité du résultat et temps de calcul nécessaire à la résolution. Ce mode opératoire permet alors d'atteindre conjointement le second objectif : les limitations de l'algorithme génétique sont mises en évidence, expliquées, et des alternatives permettant d'améliorer son fonctionnement sont proposées et testées. La résolution des exemples a été effectuée sur une station de travail Compaq W6000 avec processeur Xéon.

Ce chapitre est ainsi consacré à la présentation et l'analyse des résultats obtenus. Nous souhaitons préciser ici au lecteur que ce chapitre, malgré sa longueur, a été conservé tel quel dans le souci de respecter la logique de la méthodologie. Il a en effet semblé essentiel de ne pas dissocier, pour l'algorithme génétique, le codage de la gestion des contraintes. Pour faciliter la lecture, le chapitre sera jalonné de repères récapitulatifs fixant les conclusions partielles obtenues. Il s'organise de la manière suivante :

- Dans un premier temps, les résultats obtenus avec les méthodes de Programmation Mathématique mises en œuvre dans les solveurs de *GAMS* sont présentés.

- La deuxième partie expose le protocole utilisé pour l'évaluation de l'algorithme génétique et les premiers calculs effectués, avec un premier type de codage. Certaines limitations apparaissent avant d'arriver aux exemples les plus complexes, mettant en lumière la nécessité d'une étude sur le mode de gestion des contraintes.

- La troisième partie est donc dédiée à la gestion des contraintes dans un algorithme génétique. Plusieurs techniques sont testées et celle paraissant la plus efficace est conservée pour les exemples plus complexes.

- Cependant, les résultats de l'algorithme génétique ne sont toujours pas complètement convaincants sur divers exemples. En conséquence, deux nouvelles propositions de

codage sont émises et le jeu d'exemples est résolu de nouveau avec l'AG modifié. Les conclusions relatives aux deux objectifs fixés peuvent alors être données.

1 – Calculs avec les méthodes MP

La formulation adaptée au mode de fonctionnement des méthodes mathématiques est intégrée dans l'environnement *GAMS*. Les modules de résolution sont alors utilisés directement, sans aucune initialisation : ce point représente un avantage conséquent par rapport à d'autres solveurs tels que ceux de la bibliothèque *IMSL®*. Ainsi, pour la résolution du problème MINLP relaxé qui est propre aux deux méthodes, les variables continues sont initialisées automatiquement au milieu de leurs bornes et les variables binaires sont initialisées à 1. Les calculs réalisés successivement avec *DICOPT++*, puis *SBB* sont exposés ci-dessous. Les performances respectives des deux solveurs sont ensuite comparées.

1.1 – Résultats obtenus avec DICOPT++

Comme cela apparaît clairement sur le tableau 1, le module *DICOPT++* fait preuve d'une grande efficacité pour les quatre premiers exemples. Des solutions optimales sont trouvées en des temps pratiquement insignifiants. Seul l'exemple 5 implique un temps de calcul plus long, sans être excessif.

Exemple	1	2	3	4	5
Valeur de l'optimum [$]	166079	122470	125146	356610	620638
Temps CPU [s]	< 1	< 1	< 1	< 1	45

Tableau 1 – Résultats obtenus avec *DICOPT++*

En revanche, dès le traitement de l'exemple 6 (comportant au total 17 étapes discontinues et semi-continues), *DICOPT++* est incapable de fournir un résultat. L'exécution, de vingt-quatre heures CPU, a finalement été interrompue sans qu'aucune solution mixte ne soit trouvée. Un test a également été réalisé sur l'exemple 7 et a confirmé ce comportement : aucune solution mixte n'est trouvée en 24 heures d'exécution. Ceci justifie le fait que la résolution sur des exemples de plus grande taille n'est pas nécessaire.

Le fichier de résultats ne contient pas assez d'informations pour permettre d'interpréter et de commenter de manière approfondie l'échec de *DICOPT++* sur les deux derniers exemples traités. Cependant, il apparaît que dès l'itération suivant la résolution du MINLP relaxé, la résolution du problème maître de type MILP génère des problèmes NLP $(S(y^k))$, paramétrés par les variables discrètes y^k, qui sont infaisables. Ainsi, aucune solution faisable au problème initial n'est trouvée au cours de l'exécution. L'accumulation des problèmes continus

infaisables monopolise les ressources de calcul, lequel s'achève sur un temps limite d'exécution. Finalement, dans le cadre de la formulation employée, *DICOPT++* a prouvé son efficacité seulement sur des instances de petites tailles.

1.2 – Résultats de SBB

Les résultats obtenus avec le deuxième solveur, *SBB*, sont donnés dans le tableau 2. Contrairement à *DICOPT++*, une solution optimale a été trouvée pour les onze exemples.

Exemple	1	2	3	4	5	6
Valeur de l'optimum [$]	166079	120777	125146	356610	620638	766031
Temps CPU	< 1 [s]	< 1 [s]	< 1 [s]	< 1 [s]	1 [s]	3 [s]

Exemple	7	8	9	10	11
Valeur de l'optimum [$]	957270	1925887	2894504	3865106	4786804
Temps CPU	4 [s]	35 [s]	4 [min]	29 [min]	6,40 [h]

Tableau 2 – Résultats obtenus avec *SBB*

Les temps de calcul sont très faibles pour les sept premières instances du problème étudié. Ils commencent à augmenter à partir de l'exemple 8, puis croissent très rapidement sur les trois derniers exemples. La figure 1 représente, pour les exemples 5 à 11, l'évolution du logarithme népérien du temps de calcul en fonction du nombre de variables entières.

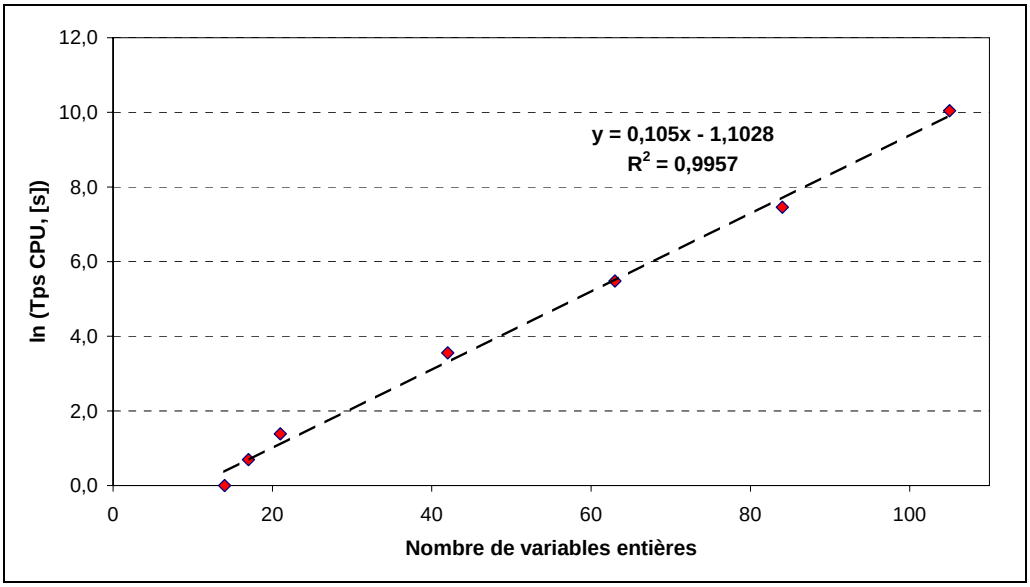


Figure 1 – Evolution du temps de calcul nécessaire à *SBB*

La courbe a une allure pratiquement linéaire. Le coefficient R^2 de la régression linéaire apparaissant sur le graphique le confirme : le temps de calcul augmente de manière exponentielle avec le nombre de variables entières. Cette remarque est en accord avec l'aspect

NP-difficile du problème, évoqué lors de sa formulation. Il est à noter cependant que cette tendance n'est vérifiable qu'à partir d'une certaine taille d'exemples. En dessous de cette taille, les variations du temps de calcul – inférieur à la seconde – ne sont pas détectables (donc de peu d'intérêt).

1.3 – Analyse des résultats

Tout d'abord, le bon comportement des deux méthodes de Programmation Mathématique sur des instances de taille raisonnable (et industriellement réalistes) est mis en évidence par les résultats précédents. Leur performance est d'ailleurs soulignée sur l'atelier 4, qui est un des problèmes existants à partir duquel a été construit le jeu d'exemples. Il a été résolu dans la littérature au moyen de diverses métaheuristiques, la meilleure solution trouvée valant 362130 [$] (avec un AG mais la solution est infaisable, cf. chapitre précédent). Ainsi, le résultat a été amélioré de 1,55 %. Par ailleurs, la certitude d'avoir déterminé un optimum met en relief l'absence de justification de l'emploi d'une méthode stochastique pour résoudre ce problème.

En revanche, la comparaison des deux méthodes déterministes met en évidence la supériorité de *SBB* sur *DICOPT++*, ce dernier étant incapable de résoudre un problème dépassant une quinzaine de variables entières (soit une cinquantaine de variables binaires, vu le mode de représentation des variables entières dans l'environnement *GAMS*, cf. chapitre 2). Cette tendance est d'ailleurs déjà annoncée lors de la résolution de l'exemple 5, que *SBB* traite en approximativement une seconde, tandis que 45 secondes sont nécessaires pour *DICOPT++*.

Cependant, un examen plus attentif montre que la supériorité de *SBB* sur *DICOPT++* repose sur un autre élément. La valeur de l'optimum trouvé par *DICOPT++* à l'exemple 2 est en effet supérieure de 1,4 % à celui déterminé par *SBB*. Le tableau 3 présente les valeurs des variables aux solutions déterminées par chaque solveur. Il est clair que les variables discrètes des première et deuxième étapes discontinues diffèrent dans l'une et l'autre des solutions proposées. Ces dernières représentent ainsi deux optima, correspondant à deux structures d'ateliers différentes, comme cela est illustré sur la figure 2.

Ainsi, l'optimum local déterminé par *DICOPT++* est de moins bonne qualité que celui trouvé par *SBB*.

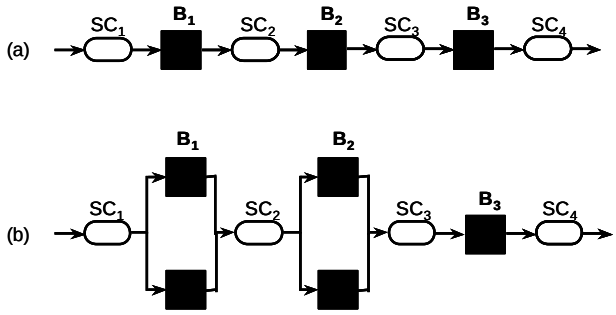


Figure 2 – Structures correspondant aux deux optima déterminés par *DICOPT++* (a) et *SBB* (b)

Pour revenir aux éléments de comparaison théoriques entre les deux méthodes mathématiques évoqués dans le chapitre consacré aux techniques d'optimisation, les résultats obtenus semblent respecter une certaine logique. En effet, ce sont les sous-problèmes NLP, générés par la linéarisation des contraintes, qui sont la cause de l'échec de *DICOPT++*.

D'après les éléments de comparaison donnés dans le chapitre 1, ce dernier devrait théoriquement être plus performant que *SBB* au fur et à mesure que la combinatoire augmente. Mais c'est bien la sévérité des non-linéarités qui pénalise *DICOPT++*. L'accumulation de problèmes continus mal conditionnés par le MILP précédent, donc infaisables, finit par le mettre en échec.

	Solution *DICOPT++*		Solution *SBB*	
	Var. continues	Var. discrètes	Var. continues	Var. discrètes
Discontinu 1	2921,6	***1***	1319,4	***2***
Discontinu 2	4382,5	***1***	1979,0	***2***
Discontinu 3	5843,3	1	2638,7	1
Semi-Continu 1	4006,1	1	2199,8	1
Semi-Continu 2	8023,2	1	4405,6	1
Semi-Continu 3	5071,5	1	2784,8	1
Semi-Continu 4	300,0	1	300,0	1

Tableau 3 – Variables correspondant aux deux solutions différentes proposées pour l'atelier 2

En conclusion de cette partie, la plus performante des méthodes de Programmation Mathématique sur la formulation retenue est indéniablement l'algorithme de Branch & Bound adapté aux problèmes non-linéaires qui est implémenté dans *SBB*. Ce dernier a fourni des solutions optimales pour tous les exemples abordés, ce qui permet d'évaluer la qualité des résultats donnés par l'algorithme génétique, présentés dans la partie qui suivante.

2 – Calculs avec l'AG : variables continues discrétisées

La logique des calculs réalisés avec l'algorithme génétique suit l'ordre chronologique de leur déroulement. En effet, les modifications apportées au codage ou l'étude sur la gestion des contraintes se sont imposées d'elles-mêmes, suite aux problèmes rencontrés et dans l'optique d'améliorer l'efficacité de la méthode. Ainsi, les calculs et leurs résultats sont présentés dans l'ordre dans lequel ils ont été effectués.

Cette première partie concernant les résultats de l'AG démarre donc sur les bases de paramètres classiques (particulièrement concernant la gestion des contraintes et le codage). Puis, le mode opératoire est établi. Les premiers calculs permettront alors de cerner les points présentés dans les chapitres précédents comme étant les enjeux pour une bonne performance de l'algorithme génétique.

2.1 – Elimination des individus infaisables

Concernant la gestion des contraintes, c'est la plus naturelle et simple à implémenter qui a été initialement retenue, à savoir l'élimination des individus infaisables. Le choix du mode de gestion des contraintes implique des procédures de sélection, évaluation et création de la population initiale bien adaptées.

Cette technique d'élimination est donc mise en œuvre par l'application, au niveau de l'étape de sélection de l'AG, de la méthode de la « roulette biaisée de Goldberg » [GOL89]. Cette dernière reproduit fidèlement les principes darwiniens de survie des individus les plus adaptés : les individus les plus forts ont une plus grande probabilité d'être choisis pour participer à la construction de la génération suivante, sans pour autant empêcher la sélection aléatoire de certains individus faibles. Ceci peut s'exprimer comme un tirage de loterie dans lequel chaque individu i occupe un secteur de la roue proportionnel à sa force relative f_i. Cette dernière correspond à la force « absolue » F_i de l'individu rapportée à la somme des forces de tous les individus de la population.

Ce mode de fonctionnement met en relief la nécessité de calculer une force permettant l'évaluation de l'adaptation de chaque individu. Cette force est maximisée par l'AG, il faut donc la formuler pour se rapporter à un problème de maximisation. La technique classique, retenue dans cette étude, s'exprime sous la forme $F_i = C_{max} - C_i$, où C_i est la valeur de la fonction objectif de l'individu i et C_{max} est une constante toujours supérieure à C_i (quel que soit i). Une autre approche associe la force à l'inverse de la fonction objectif : $F_i = 1 / C_i$; mais elle pose problème dès que l'intervalle de variation de C_i comprend zéro, c'est-à-dire

qu'un changement de signe de C_i est possible. Vu les bornes imposées sur les variables, le critère formulé comme le coût d'investissement ne peut s'annuler et cette technique aurait donc pu être adoptée dans le cas présent. Mais, les opérations de mise à l'échelle (*scaling*) qu'elle peut engendrer ont conduit à écarter cette option.

Diverses possibilités existent alors pour le choix de C_{max}, l'objectif étant de conserver F_i positive pour ne pas biaiser le fonctionnement de la roulette. Les plus courantes sont : grande valeur fixée *a priori*, plus grande valeur de la fonction objectif calculée au cours de toute la recherche, plus grande valeur de la fonction objectif dans la population courante. Cette dernière option a été retenue ici car elle apporte en plus un effet de *scaling* tout au long de la recherche, en évitant que les fonctions objectifs C_i soient toutes très petites devant C_{max}, ce qui atténuerait l'effet probabiliste de la roulette. C_{max} variera donc à chaque nouvelle génération. Ainsi, la mise en œuvre de la technique d'élimination des individus infaisables s'effectue dans le calcul de F_i :

$$\begin{cases} F_i = C_{max} - C_i \text{ si l'individu est faisable ;} \\ F_i = 0 \text{ sinon.} \end{cases} \quad (1)$$

L'élimination des individus infaisables entraîne finalement qu'au moins une partie de la population initiale doit être faisable. Dans le cas contraire, l'algorithme reviendrait à une recherche aléatoire puisque toutes les forces seraient nulles, du moins jusqu'à ce qu'une solution faisable soit rencontrée. Il a été choisi ici de générer, de manière complètement aléatoire, une population entièrement composée d'individus faisables.

2.2 – Codage discrétisé séparé

Le codage employé doit permettre de traduire un jeu complet de variables représentant l'atelier, sous la forme d'un vecteur, le chromosome. Celui-ci est composé de gènes, codant une variable sur un ou plusieurs loci. Ainsi, le codage doit nécessairement suivre la logique de la nature des variables représentées. Le problème étudié étant mixte, on différencie les gènes codant des variables continues de ceux codant des variables entières.

Concernant les premières, il est d'usage de les discrétiser de manière à pouvoir leur appliquer des opérateurs génétiques quasiment intuitifs. Ces derniers reposent généralement sur des manipulations de la structure des chromosomes pour le croisement et sur des opérations arithmétiques simples portant sur les valeurs des loci pour la mutation. La méthode de discrétisation la plus courante est le codage binaire, mais peut s'avérer lourde à mettre en œuvre lorsque les variables ne sont pas entières. Une alternative consiste à appliquer la

méthode dite de la boîte de poids, développée dans l'équipe. Celle-ci est particulièrement simple et adaptée lorsque les variables continues sont bornées, ce qui est le cas dans ces travaux. En effet, il est alors possible de les représenter sous forme réduite, comme un réel α compris entre 0 et 1 tel que :

$$V = V_{min} + (V_{max} - V_{min}).\alpha \qquad (2)$$

V_{min} et V_{max} sont respectivement les bornes inférieure et supérieure de la variable V. C'est alors le réel α qui est codé, par l'intermédiaire de ses décimales, au moyen de la méthode de la boîte de poids. Celle-ci fonctionne en représentant un entier compris entre 0 et 9 sur quatre bits, ayant les poids respectifs 1, 2, 3 et 3. Chaque décimale de α est donc codée sur une longueur fixe de quatre bits. Logiquement, plus la précision souhaitée sur la variable est grande, plus le nombre de décimales de α codées au moyen de la boîte de poids sera élevé. Ceci a une influence directe sur la taille du problème.

Dans le cadre de cette étude, les bornes inférieure et supérieure sur les variables continues valent respectivement 250 / 300 L (selon qu'il s'agit d'une étape discontinue / semi-continue) et 10000 L. Etant donné qu'une précision au moins égale à l'unité est souhaitée, le nombre N de décimales à coder doit respecter l'inégalité $10^{-N}\ (10000 - 250) \leq 1$, d'où $N = 4$. Chaque variable continue est donc codée sur 4x4 = 16 bits. Ce codage est illustré par l'exemple présenté sur la figure 3.

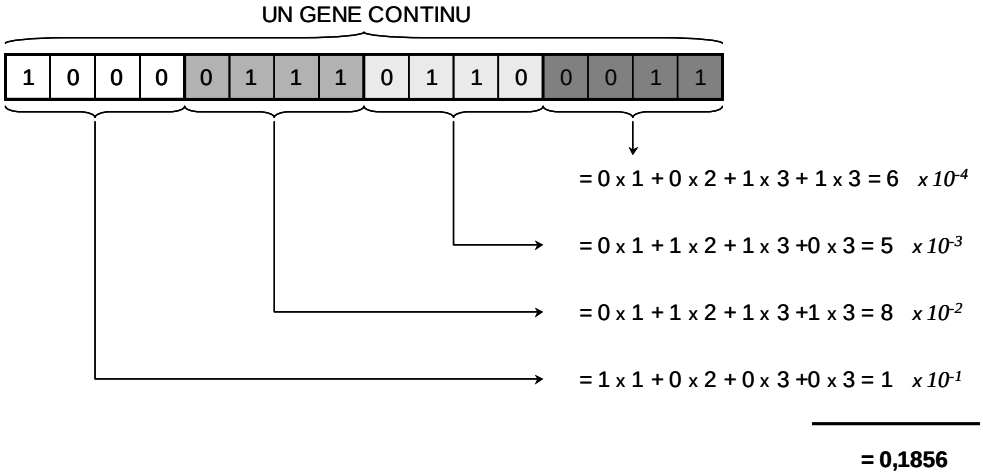


Figure 3 – Exemple de codage avec la boîte de poids

L'avantage de cette méthode est que la somme des poids des bits ne dépasse pas 9, ce qui évite la génération de solutions impossibles. En revanche, une autre forme de biais est introduite par la possibilité de coder certains entiers de plusieurs façons. Cela correspond en quelque sorte à la pondération des probabilités d'obtenir ce chiffre.

Pour les variables entières, il a été choisi de les coder telles quelles, sur un locus dont la valeur est égale à celle de la variable. Comme une variable entière représente ici le nombre d'équipements d'une étape opératoire et que celui-ci est limité à trois dans la formulation, chaque gène-locus entier adoptera la valeur 1, 2 ou 3. Finalement, le chromosome est constitué de deux zones, correspondant aux variables continues et discrètes, mises bout à bout. A titre d'illustration, une configuration particulière l'exemple 1 et le codage associé sont représentés sur la figure 4.

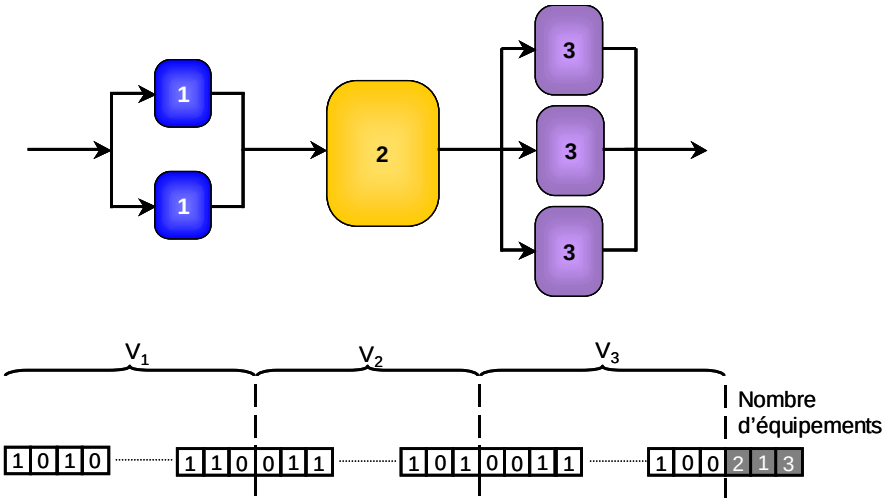


Figure 4 – Un exemple de codage « discrétisé séparé »

Comme cela a déjà été mentionné, les procédures de croisement et de mutation doivent être adaptées à la logique du codage. Ainsi, un opérateur de croisement à un point de coupure a été retenu.

Par contre, deux procédures de mutation distinctes ont été mises en œuvre : (i) mutation par inversion de la valeur du bit sur la partie « continue » du chromosome ; (ii) mutation par soustraction d'une unité (quand cela est possible) sur la zone « discrète ». Ce dernier opérateur tend à diriger naturellement la recherche vers une minimisation du critère puisqu'il signifie le retrait d'un équipement pour une étape. Cependant, il pourrait aussi amener à un blocage sur un optimum local… Les opérateurs génétiques évoqués ici sont détaillés dans l'annexe 1 de ce chapitre.

Remarque : la différence de taille entre les parties « continue » et « discrète » du chromosome risque de créer un biais. En effet, lors du croisement, le choix aléatoire du point de coupure a bien plus de chance d'affecter une variable continue. Cette remarque fera l'objet de la quatrième partie de ce chapitre.

2.3 – Mode opératoire

La nature stochastique des algorithmes génétiques implique que l'évaluation de leur performance ne peut être basée sur l'observation du seul résultat d'une exécution. Celui-ci pourrait être d'une qualité relativement bonne ou mauvaise, sans pour autant traduire l'efficacité générale de l'algorithme. Il est donc primordial, avant d'effectuer les calculs, d'établir un mode opératoire permettant de juger le comportement de l'AG de façon globale et non simplement exceptionnelle.

2.3.1 – Paramètres de l'AG

Tout d'abord, il est indispensable d'effectuer plusieurs exécutions pour chaque exemple, pour pouvoir tirer des conclusions générales sur les résultats de la méthode stochastique étudiée. Pour les premiers problèmes, dont la taille reste raisonnable, le nombre de 100 lancements par instance a été retenu puisque le temps n'est pas un facteur limitant. Puis, malgré des temps de calcul importants, ce nombre de 100 a néanmoins été conservé pour les exemples de grande taille.

Comme précisé dans l'exposé du fonctionnement de l'algorithme génétique, la taille de population et le nombre de générations calculées ont été fixés en fonction de la taille de l'exemple traité. Le réglage de ces paramètres est dicté par un compromis entre l'obtention d'une solution de bonne qualité et un temps de calcul raisonnable. Le choix des valeurs qui leur sont affectées repose essentiellement sur l'expérience. Cela a également été le cas pour les taux de croisement et de mutation, bien que divers tests aient révélé que leur influence était bien moins importante sur la performance de la méthode. Pour les calculs effectués par la suite, les valeurs retenues ont varié, selon les exemples, entre 0,4 et 0,6 pour le taux de survie et 0,3 et 0,4 pour le taux de mutation. Ces valeurs du taux de mutation sont particulièrement élevées vis-à-vis de celles habituellement rapportées dans la littérature. Il n'existe cependant pas, à notre connaissance, d'étude donnant des règles pour fixer ce paramètre dans un cas général. Celui-ci dépend du problème considéré. Des études de sensibilité et des tests préalables ont donc été réalisés sur chaque exemple, justifiant le choix des valeurs retenues et orientant clairement la recherche vers l'exploration et la diversification. Cette remarque est d'autant plus vraie sur les exemples de grande taille, pour lesquels l'espace faisable est souvent difficile à localiser.

L'ensemble des paramètres, pour chaque exemple, est rappelé dans le tableau 4. L'algorithme utilisé a été codé en Fortran 90, dans l'environnement Visual Fortran.

Exemple	Nombre d'exécutions	Nombre de générations	Taille de population
1	100	200	200
2	100	200	200
3	100	200	200
4	100	200	200
5	100	200	200
6	100	500	200
7	100	1000	200
8	100	500	500
9	100	1500	1500
10	100	2000	2000
11	100	3000	6000

Tableau 4 – Paramètres de fonctionnement de l'AG pour le jeu d'exemples

2.3.2 – Critères de performance

De la même manière que pour les méthodes déterministes, l'AG est évalué en fonction de la qualité du résultat trouvé et du temps de calcul nécessaire à la résolution. Ces critères sont confirmés par André et al. [AND01], qui évaluent les résultats d'un algorithme génétique amélioré sur la base de l'écart à l'optimum, de la vitesse de convergence et, finalement, de la robustesse. Ce dernier critère, traduisant la constance de leur algorithme sur un ensemble d'exemples différents, n'est pas directement applicable dans notre cas, dans la mesure où l'on s'est restreint à une seule formulation. Les conclusions en termes de robustesse pourront seulement être énoncées en fonction de la taille des instances traitées.

En revanche, des indicateurs supplémentaires relatifs à la qualité du résultat sont proposés ici, prenant en considération l'aspect stochastique de la méthode. En effet, l'une des perspectives de cette étude est de pouvoir prédire les performances d'un AG en fonction de l'exemple abordé. Dans cette optique, il semble au moins aussi essentiel de connaître le degré de fiabilité de cette méthode que la valeur de la meilleure solution.

Ce degré de fiabilité doit d'une certaine manière exprimer la répétabilité des résultats obtenus. Comme l'algorithme est exécuté plusieurs fois, l'intérêt se porte sur la dispersion des résultats, non en termes de moyenne et d'écart-type mais en terme d'écart à la meilleure solution trouvée. En effet, il est plus intéressant que l'ensemble des résultats de chaque exécution soit regroupé *vers le bas*, i.e. au plus près de l'optimum, plutôt qu'autour de la moyenne arithmétique des solutions. Ainsi, deux critères similaires, appelés par la suite *Dispersion à 2 %* et *Dispersion à 5 %*, ont été calculés pour chaque exemple. Ils expriment le pourcentage des exécutions *i* pour lesquelles le résultat c_i* est distant de moins de 2 % ou 5 %

de la meilleure solution C^* trouvée parmi toutes les exécutions, soit encore $c_i^* \in [C^*, C^*+2\%]$ ou $c_i^* \in [C^*, C^*+5\%]$. Combinant ces critères avec l'écart de la meilleure solution à l'optimum, il sera alors possible d'affirmer que l'AG permet d'atteindre une valeur à une distance connue de l'optimum, avec une assurance également chiffrable.

Les valeurs de 2 % et 5 % ont été retenues en considérant la signification de la fonction objectif, à savoir le coût d'investissement. L'intervalle de 2 % au-dessus de la meilleure solution trouvée signifie que la structure d'atelier déterminée est quasi-optimale, soit pratiquement équivalente pour un décideur. L'intervalle de 5 % est, lui, plus représentatif d'une borne supérieure acceptable, au-delà de laquelle les solutions trouvées ne sont réellement plus valables économiquement parlant.

Notons qu'un critère couramment employé pour illustrer la répétabilité de l'AG est le nombre d'obtentions de la meilleure solution. Il n'est cependant pas retenu ici dans la mesure où l'espace de recherche dû aux variables continues (et à la précision imposée sur celles-ci) est si grand qu'il est quasiment impossible de retrouver plusieurs fois une solution rigoureusement identique.

Dans les cas où l'élimination des individus infaisables est utilisée comme mode de gestion des contraintes, les critères d'évaluation de l'algorithme génétique se limitent à ceux évoqués ci-dessus. En revanche, lorsque la technique de gestion des contraintes ne garantit pas la faisabilité (c'est le cas pour des méthodes telles que le tournoi ou la pénalisation, qui seront développées par la suite), il est également intéressant d'observer l'évolution au cours de la recherche de la proportion de solutions faisables dans la population, ainsi que la quantité d'échecs qui peuvent être rencontrés lors du traitement des exemples les plus complexes (un échec étant une exécution au cours de laquelle aucune solution faisable n'a été repérée).

Le protocole opératoire et le mode d'analyse des résultats étant définis, les paramètres de la méthode étant fixés, les calculs réalisés avec l'AG sont présentés ci-après.

2.4 – Jusqu'à l'exemple 7…

La partie suivante est ainsi centrée sur les résultats obtenus avec l'AG. Des tendances générales, traduisant le fonctionnement de cette méthode stochastique, sont tout d'abord dégagées. Elles sont basées sur une observation plus fine des résultats fournis par le premier exemple. Puis, les solutions pour le jeu d'exemples proposé sont présentées. L'analyse des performances, et plus particulièrement du temps de calcul, amènera cependant à un changement d'orientation.

2.4.1 – Caractéristiques générales de l'AG

Les résultats pour le premier exemple sont tout à fait satisfaisants et révèlent le bon comportement de l'algorithme génétique : la meilleure solution trouvée est presque égale à l'optimum déterminé par les méthodes MP et les dispersions indiquent que c'est également le cas pour la majorité des exécutions (tableau 5). Le temps de calcul est, lui, très faible.

Meilleure solution trouvée [$]	166083
Ecart à l'optimum (%)	< 0,01
Dispersion 2 %	99
Dispersion 5 %	100
Temps CPU	< 1 [s]

Tableau 5 – Résultats pour l'atelier 1

Le comportement global de l'AG est également illustré sur la figure 5, donnant l'évolution classique de la fonction objectif des meilleur et moins bon individus, ainsi que la moyenne sur la population. Les trois indicateurs atteignent un palier à partir de la génération 30, mais une observation plus fine montre que sur les 170 générations restantes, le critère du meilleur individu diminue encore de près de 2 %. Le bruit très important sur la moyenne est dû à la variation, à chaque génération, du nombre de solutions faisables. En effet, la moyenne est calculée sur l'ensemble de la population et non sur les solutions faisables. Or, la force affectée aux solutions infaisables étant nulle, une faible variation de leur nombre peut avoir un impact important sur la moyenne globale de la force de la population.

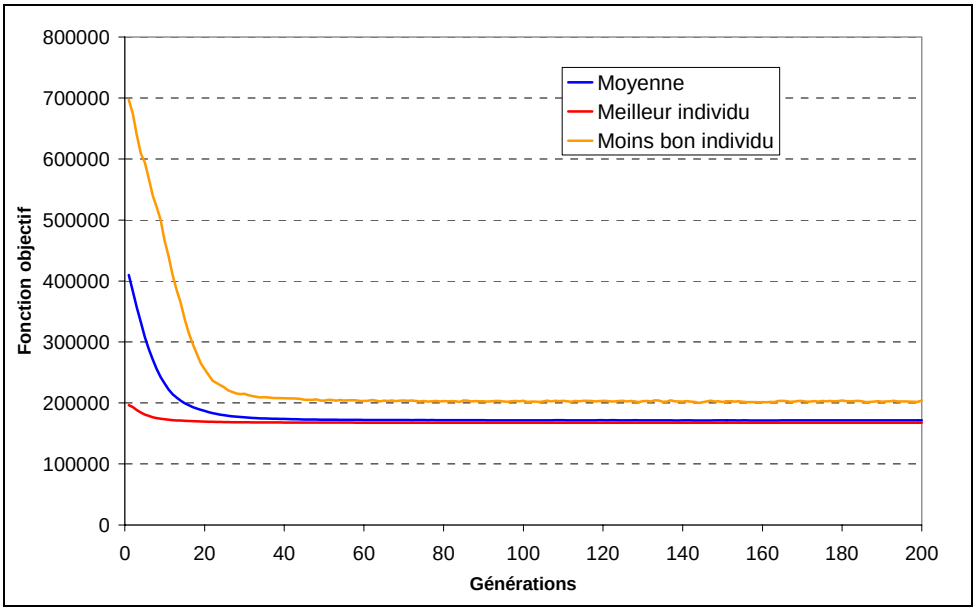


Figure 5 – Evolution du critère au fil des générations

La critique couramment émise à l'encontre des AG est de n'être pas plus efficaces qu'une simple recherche aléatoire. Cette remarque est contredite par la comparaison de la moyenne du critère sur la population entre la première génération (générée aléatoirement) et la dernière : elle passe de 409868 à 170991, signifiant une diminution de 58 %. De la même manière, la diminution de l'écart-type entre ces deux populations extrêmes s'approche de 95 %. Les résultats obtenus sur ce premier exemple permettent donc de valider le comportement de l'algorithme génétique. La simplicité de l'exemple traité est relativisée par l'impossibilité de trouver une bonne solution au moyen d'une recherche aléatoire, ce qui souligne l'efficacité de l'AG.

2.4.2 – Résultats des exemples suivants

Les meilleures solutions trouvées par l'algorithme génétique ainsi que le temps de calcul nécessaire au déroulement d'une exécution sont donnés dans le tableau 6 pour les sept premiers exemples.

	At. 1	At. 2	At. 3	At. 4	At. 5	At. 6	At. 7
Meilleure solution [$]	166083	122493	125208	368798	622566	769048	970376
Temps CPU	< 1 [s]	< 1 [s]	< 1 [s]	2 [s]	35 [s]	3 [min]	1,5 [h]

Tableau 6 – Résultats pour les sept premiers exemples

Il est cependant plus aisé de visualiser les meilleures solutions en termes d'écart à l'optimum déterminé par les méthodes mathématiques, tels qu'ils apparaissent sur la figure 6. Enfin, les dispersions à 2 % et à 5 % évoquées plus haut sont présentées sur la figure 7.

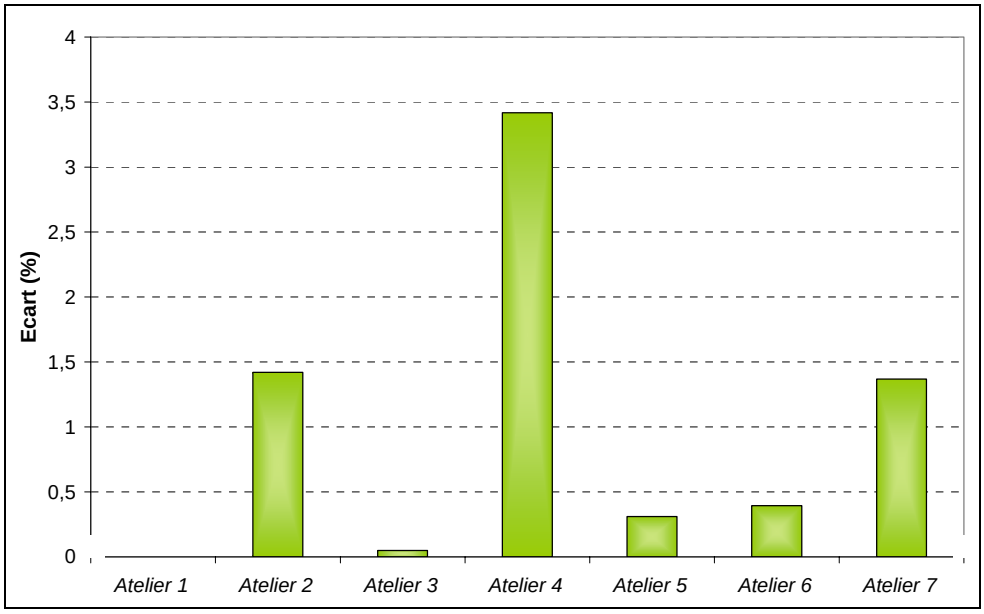


Figure 6 – Ecart entre la solution de l'AG et l'optimum

Si les résultats de l'exemple 1 sont de bonne qualité, la meilleure solution trouvée pour l'atelier 2 est moins satisfaisante, à 1,42 % de l'optimum. L'analyse des variables discrètes de la solution trouvée par l'AG montre que celui-ci reste bloqué sur l'optimum local qui avait piégé le solveur *DICOPT++* (l'écart entre les solutions respectives est inférieur à 0,1 %). Un seul équipement est proposé pour les deux premières étapes discontinues tandis que l'optimum trouvé par *SBB* en compte deux (cf. figure 2). La dispersion démontre que, malgré l'aspect stochastique de l'AG, la très grande majorité des exécutions est piégée par cette difficulté.

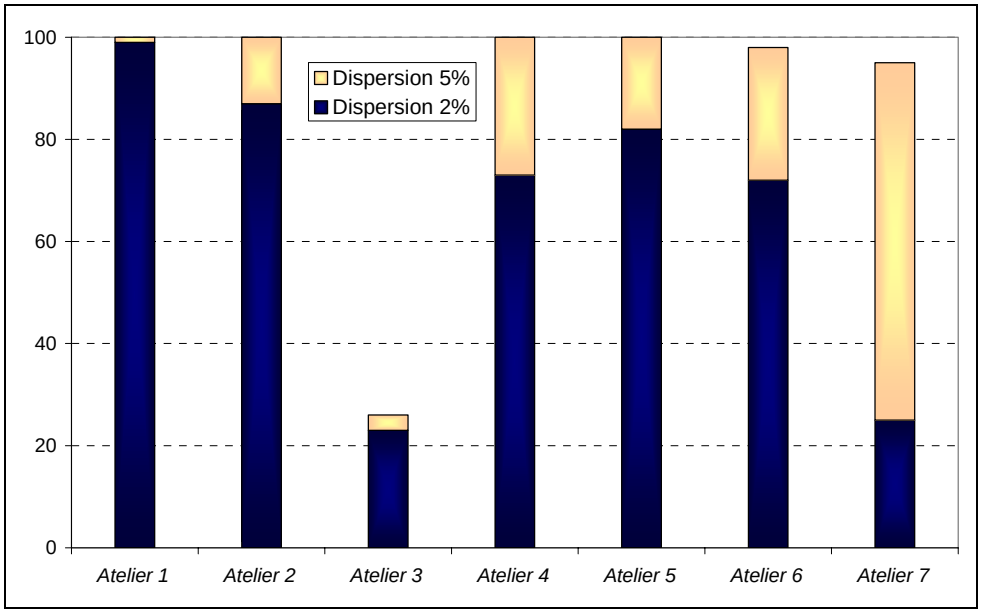


Figure 7 – Dispersion des résultats pour les sept premiers exemples

Pour l'exemple 3, la qualité de la meilleure solution trouvée est de nouveau excellente. Par contre, les dispersions à 2 % et à 5 % sont médiocres. En effet, un optimum local pénalise également la recherche ; il vaut 132246 (soit à 5,67 % de la valeur optimale), et diffère du meilleur optimum trouvé par une seule variable discrète (nombre d'équipements en parallèle pour la deuxième étape discontinue). Cet optimum local a été repéré en fixant le jeu de variables entières grâce aux « mauvaises solutions » de l'AG et en résolvant le problème NLP correspondant. Une partie des exécutions est restée piégée dans cet optimum local, ce qui explique la faiblesse des dispersions. Mais l'AG a réussi cette fois-ci à dépasser cette difficulté, puisque l'optimum déterminé par les méthodes mathématiques a tout de même été localisé à moins de 0,1 %.

Le bilan de l'étude du quatrième exemple montre, lui aussi, que l'AG reste bloqué sur un optimum local à près de 3,5 % de l'optimum trouvé par *SBB* et *DICOPT++*. La figure 8 met en évidence la différence entre les deux optima pour le nombre d'équipements en parallèle des deuxième et troisième étapes opératoires. Les dispersions à 2 % et 5 % sont convenables,

indiquant que pratiquement toutes les exécutions restent bloquées sur cet optimum local. Concernant le temps de calcul, l'atelier 4 est le premier pour lequel la résolution dépasse une seconde, sans pour autant augmenter excessivement.

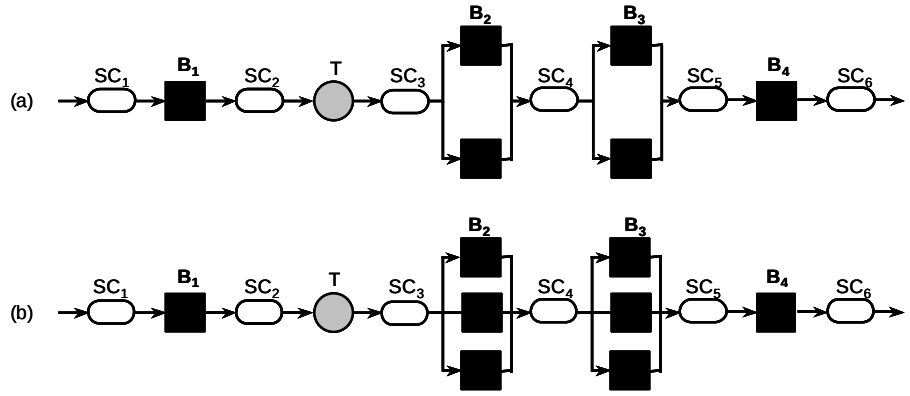


Figure 8 – Structures correspondant à (a) l'optimum donné par les méthodes MP et (b) la solution de l'AG

Sur les exemples 5 et 6, l'AG repère avec précision des solutions proches de l'optimum, à 0,31 % et 0,39 % respectivement. Les dispersions des exécutions autour de la meilleure solution trouvée sont toujours favorables, même si elles diminuent légèrement pour l'atelier 6. Le temps de calcul augmente par contre fortement sur ces deux exemples : il est multiplié par 180 entre les exemples 4 et 6.

Finalement, une relative baisse des performances de l'algorithme génétique est constatée sur l'instance 7. Tout d'abord en termes de qualité, puisque la meilleure solution trouvée est à presque 1,5 % de l'optimum. En revanche, contrairement aux exemples 2 et 4, cet écart important n'est pas expliqué par une mauvaise détermination des variables discrètes, témoignant de la présence d'un optimum local. Par ailleurs, la dispersion à 2 % est divisée par trois par rapport à l'atelier 6, bien que celle à 5 % reste quasiment parfaite. Enfin, le temps de calcul, multiplié par trente, stigmatise la diminution de l'efficacité de l'AG. L'explication de cette tendance est justifiée par la suite.

2.4.3 – Conclusions

D'un point de vue général, les résultats exposés ci-dessus sont d'une bonne qualité. Certains problèmes, liés à l'existence d'optima locaux, viennent nuancer ces conclusions, mais le comportement de l'AG est globalement satisfaisant, sauf en ce qui concerne le temps de calcul. En effet, si ce dernier est pratiquement négligeable pour les exemples les plus simples, il explose littéralement sur les trois dernières instances traitées. Cette tendance risque de compromettre les calculs sur les problèmes de grande taille, le temps de calcul devenant un

facteur limitant. Un changement de stratégie, signifiant une adaptation de l'algorithme génétique est donc indispensable à la poursuite de l'étude.

3 – Stratégies de gestion des contraintes au sein d'un AG

Les calculs effectués jusqu'à présent avec l'algorithme génétique ont prouvé la qualité des résultats obtenus, justifiant la poursuite de l'étude sur les exemples de plus grande taille. En revanche, le temps de calcul augmente considérablement pour les dernières instances traitées, jusqu'à atteindre 1,5 heure par exécution pour la résolution de l'exemple 7. Cette tendance, représentée sur la figure 9 (en échelle logarithmique), est peu marquée sur les premiers exemples mais s'accentue avec l'augmentation du nombre de variables. Le temps de calcul risque donc de devenir rédhibitoire pour la résolution des derniers exemples. La poursuite des calculs est ainsi conditionnée par l'amélioration de l'efficacité de l'algorithme, démarche qui requiert tout d'abord l'explication de cet accroissement du temps de calcul.

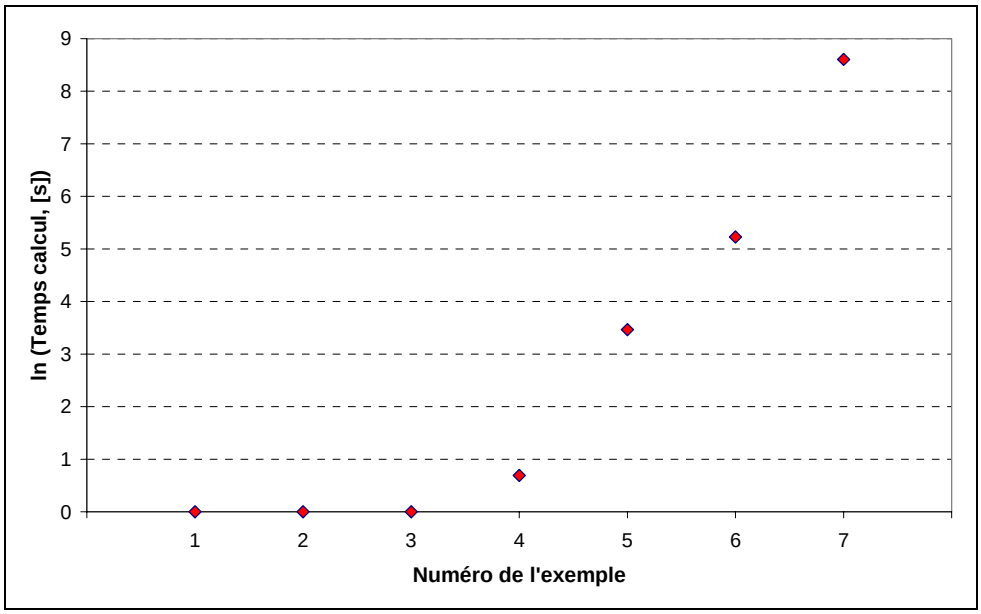


Figure 9 – Evolution du temps de calcul sur les 7 premiers exemples

Comme le temps de calcul est directement lié au nombre d'évaluations de la fonction objectif, l'observation de l'évolution de ce dernier révèle la disproportion croissante entre sa valeur théorique et sa valeur effective (on considère que le nombre théorique d'évaluations est égal au produit du nombre de générations par la taille de population). La figure 10 met en évidence cette tendance, signifiant qu'au cours d'une exécution, le temps réellement passé au déroulement de l'algorithme génétique, à savoir à l'enchaînement des générations, est minime. Logiquement, cette affirmation implique que la génération de la population initiale occupe la majorité des ressources. Ce comportement s'explique simplement par le mode de

gestion des contraintes : comme cela a été précisé précédemment, la technique d'élimination des individus infaisables oblige à créer une population initiale entièrement faisable. Or, cette recherche se fait de manière aléatoire, tâche qui s'avère d'autant plus compliquée que les exemples deviennent plus complexes et plus sévèrement contraints.

En conséquence, le nombre d'individus « visités » pour trouver l'équivalent d'une population qui respecte la contrainte augmente avec la taille des exemples. Ce nombre d'individus infaisables testés pour construire la population initiale apparaît sur la figure 10 comme la différence entre les nombres d'évaluations théorique et effectif de la fonction objectif. S'il est pratiquement négligeable pour les trois premiers exemples, il augmente de façon exponentielle sur les quatre suivants, allant jusqu'à atteindre 70 millions d'individus testés à l'exemple 7 pour un nombre d'évaluations théoriques de la fonction objectif égal à 200000 (soit un rapport de 350).

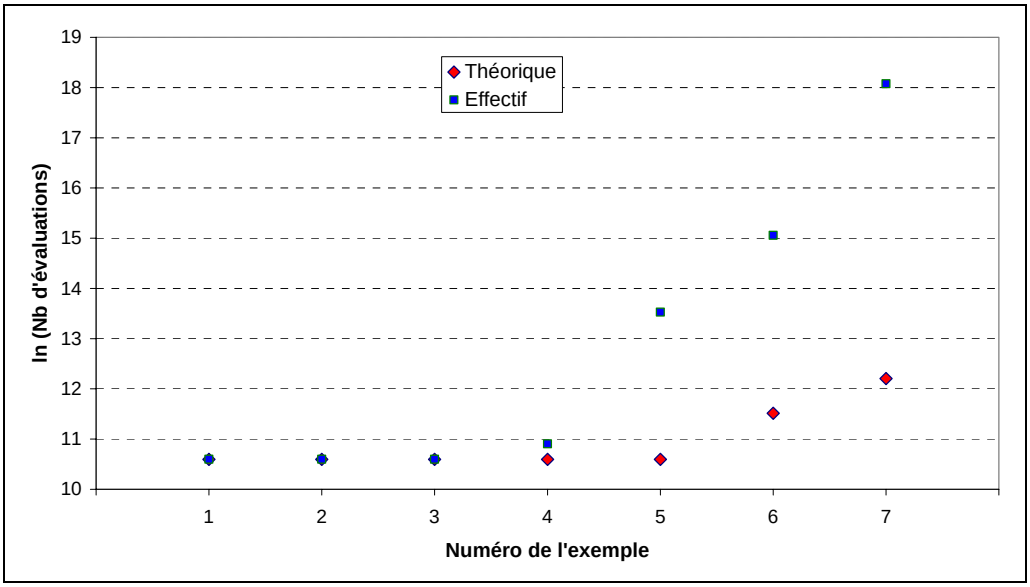


Figure 10 – Nombre d'évaluations du critère

Ces remarques impliquent la nécessité de trouver un mode de gestion des contraintes différent, qui pénalise au minimum la qualité des résultats tout en améliorant l'efficacité de l'algorithme en termes de temps de calcul.

La stratégie suivante est donc proposée : une recherche bibliographique sur les méthodes de gestion des contraintes existantes est menée dans un premier temps, de manière à tester plusieurs d'entre elles sur un exemple déjà traité par élimination. La comparaison des résultats aidera ainsi à déterminer celle qui sera utilisée pour compléter les calculs sur le jeu d'exemples.

3.1 – Analyse préliminaire

La structure initiale des algorithmes génétiques imaginée par Holland [HOL75] ne prévoyait pas la prise en compte de problèmes contraints. Or la plupart des applications, qu'elles soient purement mathématiques ou issues du monde réel – et particulièrement du domaine des procédés et systèmes industriels, est modélisée comme des problèmes soumis à des contraintes plus ou moins sévères. Par conséquent, la communauté des algorithmes évolutifs s'est mobilisée pour proposer des méthodes de gestion des contraintes, notamment au sein des algorithmes génétiques. Un grand nombre de techniques plus ou moins spécifiques aux problèmes traités a ainsi été développé.

Plusieurs revues de l'ensemble des techniques existantes sont disponibles, par exemple dans [COE02a] ou [MIC96a]. Elles distinguent généralement les approches suivantes :

(i) Elimination des individus infaisables.
(ii) Pénalisation de la fonction objectif.
(iii) Notions de dominance.
(iv) Préservation de la faisabilité des individus.
(v) Réparation des individus infaisables.
(vi) Méthodes hybrides.

La méthode de rejet des individus infaisables, étudiée précédemment, a montré ses limites en ce qui concerne l'efficacité de l'obtention d'une solution dès que l'instance atteint une taille et une sévérité des contraintes critiques. Un autre point faible est qu'elle n'exploite aucune information des solutions de l'espace infaisable : il est néanmoins raisonnable de penser que si ces dernières ne sont pas trop éloignées de la frontière de faisabilité, elles peuvent aider à diriger la recherche vers l'optimum global. Cependant, cette technique reste une première approche valable lorsque le problème n'est pas connu, quitte à se réorienter ensuite vers des techniques plus appropriées, comme la présente étude le montre.

3.1.1 – Fonctions de pénalité

La deuxième catégorie citée ci-dessus est sûrement la plus usitée car la plus simple d'implémentation. Elle consiste à transformer un problème contraint en un problème non contraint en ajoutant un terme fonction des contraintes dans le critère à minimiser. Une fois ce principe de base énoncé, il existe une multitude de techniques pour formuler le terme de pénalité. Tout d'abord, il est admis que la pénalisation sera plus efficace si elle est calculée en rapport avec l'importance de la violation de la contrainte, assimilable à la distance à l'espace faisable, plutôt qu'avec le nombre de contraintes violées [RIC89].

Considérant un problème d'optimisation classique cherchant à minimiser $f(x)$, soumis à m contraintes inégalité $g_j(x) \leq 0$ ($j \in \{1,\ldots m\}$), le nouveau problème non contraint formulé au moyen de la méthode de pénalisation dans sa forme la plus générale consiste à minimiser le nouveau critère F suivant :

$$F(x) = f(x) + \sum_{j=1}^{m} R_j . max[0, g_j(x)]^{\beta} \qquad (3)$$

Dans cette expression, β est le plus souvent pris égal à 2 pour réaliser une pénalisation quadratique de l'importance des violations des contraintes. Dans le cas de contraintes égalité $h_k(x) = 0$, ces dernières sont le plus souvent reformulées sous la forme d'inégalités telles que $/h_k(x)/ - \varepsilon \leq 0$. Le coefficient R_j peut alors s'exprimer de diverses manières, variant par leur complexité et leur efficacité selon les problèmes. Des principes généraux peuvent être cependant énoncés et ont guidé les études réalisées pour la mise en œuvre de stratégies de pénalisation efficaces.

Tout d'abord, la première observation souligne que l'optimum, dans la majorité des problèmes, se situe sur la frontière de l'espace faisable. En conséquence, un coefficient de pénalité trop fort exerce une pression sur les individus de manière à les repousser vers l'intérieur de l'espace faisable, leur interdisant de se diriger vers des régions plus prometteuses en termes de qualité de la fonction objectif. Par ailleurs, dans le cas où l'espace faisable est disjoint, un terme de pénalité trop grand peut pousser la population vers l'une des régions faisables et l'y confiner en l'empêchant de traverser des zones infaisables pour aller vers d'autres régions faisables, où l'optimum global se trouve peut-être [WU04]. En revanche, un coefficient de pénalité trop faible amène à effectuer une recherche exhaustive de l'espace infaisable, visitant des zones où la fonction objectif est très faible mais les solutions fortement infaisables, i.e. des optima qui ne sont pas celui du problème contraint initial. Ce comportement est illustré par la figure 11, où $f(x)$ est une fonction objectif mono-variable et $F(x)$ le critère pénalisé ou non suivant le respect de la contrainte $g(x) \leq 0$.

Par ailleurs, en ce qui concerne le déroulement de l'algorithme, il est communément admis qu'il est préférable d'adopter un terme de pénalité assez faible au début de la recherche de manière à explorer la plus grande partie possible de l'espace faisable. Puis, au fur et à mesure que les régions prometteuses sont ciblées, l'augmentation de la pénalisation permet d'intensifier la recherche sur ces zones en obligeant les individus à respecter les contraintes.

A la lumière de ces concepts généraux, de nombreuses variantes de la méthode de pénalisation des individus infaisables ont été développées. Les plus classiques sont rappelées dans [MIC95], comparées dans [MIC96b] et certaines d'entre elles sont évoquées ici. La liste

présentée est cependant loin d'être exhaustive. La technique la plus simple, dite pénalité statique, consiste à affecter une valeur numérique à chaque R_j qui ne variera pas pendant toute la durée de la recherche. Le désavantage évident de cette méthode est la nécessité de fixer autant de paramètres qu'il y a de contraintes, sachant qu'aucune méthodologie pour les régler n'est clairement établie. Une première idée consiste à fixer les coefficients de telle sorte que le terme de pénalité et la fonction objectif aient le même ordre de grandeur. Mais le moment de la recherche où cette égalité doit être vraie est laissée dans le flou puisque la fonction objectif varie pendant une exécution. Une autre piste est la normalisation des contraintes et de la fonction objectif, réduisant à un le nombre de paramètres à ajuster.

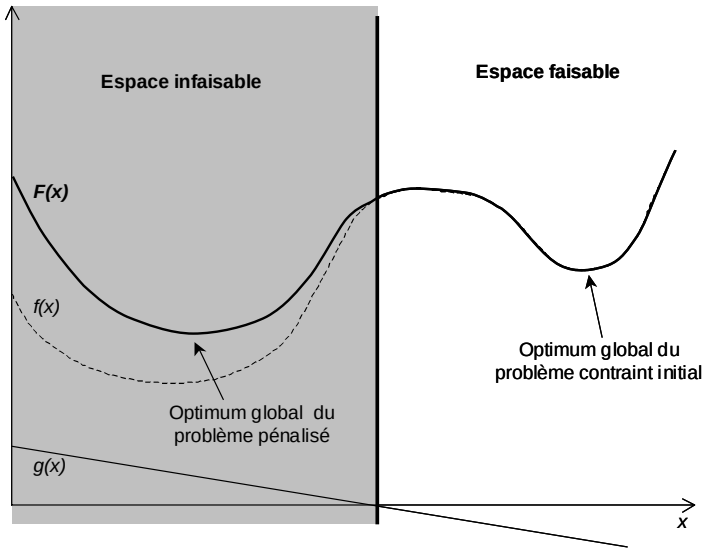


Figure 11 – Trop faible coefficient de pénalité pour un problème mono-variable

Une technique de pénalité statique a été modifiée dans [HOM94] où des niveaux de violation de chaque contrainte sont établis et un paramètre affecté à chaque niveau, soit $m(2l+1)$ paramètres à fixer si on considère l niveaux pour m contraintes. En effet, il faut fixer le nombre de niveaux de violation (un par contrainte), les intervalles des l niveaux de violations (l par contrainte) ainsi que le coefficient de pénalité associé (l par contrainte). Ainsi, dans un problème comportant simplement trois contraintes et trois niveaux de violation, douze paramètres sont déjà nécessaires…

La technique de la pénalité dynamique consiste à exprimer R_j sous la forme $(C \times t)^{\alpha}$ où t est le nombre d'itérations (générations). Les paramètres à ajuster sont ici C et α, des valeurs classiques étant respectivement 0,5 et 2. Cette technique permet d'accroître la pression sur les solutions infaisables au fur et à mesure que la recherche progresse.

Un effet similaire est obtenu en appliquant une méthode s'apparentant au recuit simulé :

$$R_j = 1 / 2\tau \quad (4)$$

où τ est une température qui décroît au cours des générations. Il est donc nécessaire de fixer une température de début τ_i et de fin τ_f, ainsi qu'un schéma de refroidissement pour τ. Cette méthode a deux particularités. Tout d'abord, elle différencie les contraintes linéaires des contraintes non-linéaires. La faisabilité des individus relativement aux premières est maintenue grâce à des opérateurs particuliers, ce qui permet de n'inclure que les secondes dans le terme de « pénalité de recuit ». En outre, la population initiale est composée de clones d'un même individu auquel il est imposé de respecter les contraintes linéaires (dans le but d'appliquer des opérateurs génétiques conservant la faisabilité vis-à-vis des contraintes linéaires).

Des approches différentes, dites de pénalité adaptable, se basent sur l'apprentissage du comportement de la population dans les dernières générations. Ainsi dans [HAD97], si, durant les *k* dernières générations, le meilleur individu a été toujours faisable, alors le coefficient de pénalité est diminué, et inversement. Dans les cas d'indétermination, le coefficient reste inchangé. Cette démarche impose alors de fixer la valeur initiale du coefficient de pénalité et le nombre de générations *k* sur lequel est mené l'apprentissage.

La tendance actuelle repose sur un coefficient de pénalité s'ajustant lui-même (*self-adaptive penalty approach*), par un apprentissage basé sur la recherche en cours, et évitant d'avoir à mener une étude pour fixer un quelconque paramètre. Dans la technique adoptée dans [WU04], les contraintes et la fonction objectif sont normalisées. Pour chaque contrainte *j*, un coefficient de pénalité de la génération *q* est calculé comme étant le produit du coefficient de la génération *q-1* par un facteur dépendant de la proportion d'individus faisables à la génération en cours *q*. Si cette proportion est supérieure à 50 %, le facteur diminue de manière à favoriser les individus situés du côté infaisable de la frontière établie par la contrainte *j*. A l'inverse, si le nombre d'individus faisables est peu important, la valeur du coefficient augmente de manière à rediriger la population vers l'intérieur de l'espace faisable. La valeur initiale est le rapport de la valeur moyenne de la fonction objectif sur celle de la contrainte à la première génération, ce qui induit la normalisation implicite du terme de pénalité. Aucun ajustement de paramètre n'est donc nécessaire à la mise en œuvre de cette technique.

Coello Coello [COE02b] propose également une méthode de pénalité auto-adaptée basée sur la notion de co-évolution. On considère, outre la population classique *P1* d'individus

codant le problème considéré, une population *P2* d'individus qui représentent deux coefficients de pénalité permettant d'évaluer la population *P1*. Le premier, w_1, porte sur la somme de l'importance de la violation de toutes les contraintes (ou distance à l'espace faisable) tandis que w_2 représente le nombre de contraintes violées. Ainsi, chaque individu de *P1* est évalué un nombre de fois égal à la taille de *P2*. La démarche consiste alors à laisser évoluer *P1* sur un certain nombre de générations, puis d'évaluer chaque individu de *P2*, i.e. chaque jeu de deux coefficients de pénalité (figure 12). Grossièrement, cette évaluation est définie comme la moyenne des critères de *P1* évaluée par chaque individu de *P2* (seuls les individus faisables de *P1* sont pris en compte dans le calcul de la moyenne). Puis, *P2* évolue comme dans n'importe quel algorithme génétique, à chaque génération de *P2* correspondant une évolution complète de *P1*. Le désavantage est évidemment le nombre d'évaluations du critère, rendant cette méthode très gourmande en temps de calcul.

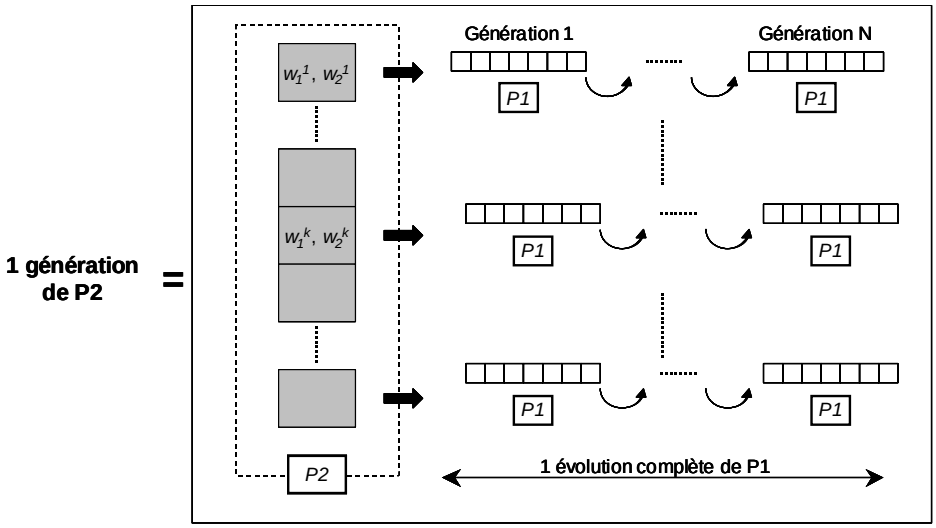


Figure 12 – Pénalité auto-adaptée par co-évolution [COEL02b]

Enfin, la méthode présentée dans [DEB00] se situe à mi-chemin entre les méthodes de pénalisation classique et celles basées sur des notions de dominance (cf. partie suivante). Elle consiste à établir la supériorité des solutions faisables sur celles qui violent une ou plusieurs contraintes en formulant le critère pénalisé sous la forme suivante :

$$\begin{cases} F(x) = f(x) & \text{si } g_j(x) \leq 0,\ j = \{1,\ldots,m\} \\ F(x) = f_{max} + \sum_{j}^{m} max[0, g_j(x)] & \text{sinon.} \end{cases} \quad (5)$$

où f_{max} est la valeur de la fonction objectif de la moins bonne solution faisable dans la population courante. Une étape de sélection par tournoi est réalisée pour mettre en œuvre cette gestion des contraintes particulière, mais il est également possible de l'effectuer au moyen de la classique roulette de Goldberg [COS01]. Ainsi, la majorité des individus étant infaisable au

début de la recherche, la sélection exerce exclusivement une pression dans la direction de l'espace faisable. Une fois que des individus faisables sont trouvés, leur supériorité sur les autres les favorise. Mais sans l'aspect stochastique du tournoi ou de la roulette de Goldberg, l'amélioration de la fonction objectif n'interviendrait qu'à partir du moment où le nombre de solutions faisables serait supérieur au nombre de survivants. Malgré cela, et en plus de la mutation, l'emploi de techniques de maintien de la diversité de la population, telles que le *niching*, est parfois nécessaire pour éviter de voir stagner la recherche autour d'un optimum local.

Nota : le *niching* consiste à éviter de conserver deux solutions représentant deux jeux de variables très semblables. Des considérations métriques (généralement le calcul de la distance euclidienne entre deux points) estiment la proximité entre les deux solutions et empêchent l'une des deux de survivre. Dans le cas de [DEB00], le tournoi entre deux individus faisables n'est autorisé que si la distance euclidienne qui les sépare est inférieure à un seuil fixé préalablement.

Finalement, un échantillon parmi cette profusion de techniques de pénalisation a été évalué dans [MIC96b] sur un ensemble d'exemples-test et si certaines de ces méthodes semblent particulièrement adaptées à certains problèmes, les auteurs finissent par retenir la pénalité statique, technique la plus simple et, surtout, la plus générique.

3.1.2 – Méthodes basées sur des notions de dominance

Cette classe de techniques de gestion des contraintes repose sur les principes de l'optimisation multiobjectif, établis sur le concept de dominance. La première idée est ainsi de redéfinir un problème d'optimisation monocritère contraint en un problème multicritère non contraint, où les m contraintes représentent chacune un critère à minimiser. Des procédures de tri s'appuyant sur la notion de domination d'une solution par une autre au sens de Pareto (x domine y si et seulement s'il est meilleur que y pour au moins un critère, et aussi bon pour les autres) font tendre vers la solution idéale x : $g_j(x) = 0$ pour tout $j=1,\ldots,m$ et $f(x) \leq f(y)$ pour tout y faisable [PAR96].

Ces notions sont reprises dans [COE02c] dans le cadre de l'optimisation monocritère sous contraintes pour énoncer des règles de dominance particulières établissant la supériorité des solutions faisables sur celles qui sont infaisables. Dans le cas d'une minimisation, elles s'écrivent :

1. Une solution faisable domine une solution infaisable ;
2. De deux solutions faisables, celle avec la plus petite fonction objectif domine l'autre ;

3. De deux solutions infaisables, celle caractérisée par la moindre violation de la contrainte l'emporte.

Ces règles sont implémentées dans un tournoi : on peut remarquer que cette technique est finalement exactement la même que celle utilisée par Deb [DEB00], qui formalise ces règles sous forme d'un terme de pénalité ajouté à la fonction objectif.

Silva et Biscaia [SIL03] utilisent une méthodologie tout à fait similaire en optimisation multicritère, ajoutant simplement des rangs supplémentaires de domination. Les contraintes étant normalisées, quatre niveaux de domination sont définis par des intervalles de violation de contrainte dont l'union est [0,1]. Chaque individu est rangé dans un de ces niveaux selon la plus grande violation de contrainte le caractérisant. Puis une valeur entière positive est affectée à chaque niveau de domination et ajoutée comme un terme de pénalité à chaque fonction objectif, préalablement normalisée. La sélection est mise en œuvre au moyen de tris de Pareto successifs retenant les individus non-dominés jusqu'à obtenir le nombre de survivants désiré.

Les deux exemples évoqués ici mettent finalement en évidence à quel point la frontière est ténue entre ce mode de gestion des contraintes et celui fonctionnant par pénalisation de la fonction objectif.

3.1.3 – Autres techniques

La description des autres techniques est regroupée dans cette unique section dans la mesure où elles ne sont applicables que dans le cadre de certaines hypothèses, ou même définies exclusivement pour un problème particulier.

Les techniques préservant la faisabilité de la solution s'appuient en général sur des opérateurs de croisement et de mutation particuliers, caractérisés par leur faculté à construire, à partir d'individu(s) faisable(s), un ou plusieurs individus également faisables. Un exemple est fourni par l'algorithme *GENOCOP* [MIC94a], applicable à des problèmes purement linéaires. Les contraintes égalité sont éliminées par substitution d'un nombre égal de variables, l'espace faisable étant alors un ensemble convexe défini par des inégalités linéaires. Grâce à cette propriété, les opérateurs génétiques consistent en des combinaisons linéaires d'individus qui assurent la faisabilité des solutions ainsi créées. Le maintien de la faisabilité peut aussi être obtenu par des « décodeurs », c'est-à-dire des instructions contenues dans le chromosome dictant une manière de construire une solution faisable.

Par ailleurs, les techniques de réparation des chromosomes infaisables jouissent d'une certaine popularité. Dans de nombreux cas d'optimisation, surtout combinatoire, il est aisé de développer des règles de réparation qui, partant d'un individu infaisable, permettent de modifier plus ou moins sa structure pour construire un individu faisable. Dans [DIE04] par exemple, des procédures de réparation sont prévues pour agir sur des individus dont le chromosome, résultant de mutation et/ou de croisement, n'a pas de signification physique compte tenu du codage employé. En revanche, les règles de réparation sont toujours dédiées au cas particulier du problème étudié et il n'existe pas d'heuristique applicable dans une perspective générale. Une particularité de la méthode de réparation est également la possibilité de remplacer l'individu infaisable par sa version réparée dans la population ou, au contraire, d'utiliser cette version simplement pour l'évaluation de la solution.

Une méthode de réparation généralisée, proposée dans [CHO06], s'appuie sur le développement au premier ordre du vecteur de violation des contraintes ΔV par rapport aux variables d'optimisation x :

$$\Delta V = \nabla_x V.\Delta x \quad \Rightarrow \quad \Delta x = \nabla_x V^{-1}.\Delta V \tag{6}$$

Ainsi, connaissant la valeur de la violation de chaque contrainte et en approximant numériquement le gradient de la violation, il est théoriquement possible de déterminer le vecteur Δx de réparation de l'individu infaisable. Il est cependant probable qu'une telle méthode, malgré son ambition de généricité, ne soit applicable que dans certains cas, restreints en nombre, où les fonctions et la nature des variables mises en jeu soient assez favorables.

Cette dernière technique est à ranger parmi les méthodes hybrides, au même titre que l'intégration des paramètres de Lagrange dans une fonction de pénalité, l'application de concepts tirés de la logique floue, etc… Il est conseillé de se référer à [COE02a] pour plus de détails.

3.2 – Modes de gestion des contraintes envisagés

Il serait exhaustif de traiter tous les modes de gestion des contraintes évoqués plus haut. Seuls ceux paraissant facilement applicables et adaptés au problème traité sont donc testés dans cette partie. La démarche employée consiste à compiler les performances des diverses méthodes selon une démarche identique à celle de la partie 2.

Par contre, seule une instance servira d'exemple-test. Sa taille est choisie de manière à confronter l'algorithme à une certaine difficulté de résolution, à éprouver réellement son efficacité, sans pour autant être pénalisé par des temps de calcul prohibitifs. C'est donc l'exemple 5 qui a été retenu pour les calculs suivants. Les résultats déjà obtenus avec la technique d'élimination des individus infaisables serviront de référence pour la qualité des résultats. Pour rappel, l'optimum déterminé par les deux méthodes de Programmation Mathématique vaut 620638 [$].

3.2.1 – Pénalisation des individus infaisables

La technique de pénalisation des individus infaisables employée dans cette étude est la plus classique, c'est-à-dire avec un coefficient de pénalité statique. Le critère minimisé par l'algorithme génétique s'écrit alors :

$$\begin{cases} F(x) = f(x) & \text{si } \sum_{i}^{I} H_i \leq H, \\ F(x) = f(x) + \rho\,(H - \sum_{i}^{I} H_i)^2 & \text{sinon.} \end{cases} \qquad (7)$$

A titre de rappel, H est l'horizon de temps imparti pour la production des I produits et H_i le temps de production effectif pour chacun d'entre eux (cf. chapitre 2). La méthode de sélection, inchangée par rapport à la technique d'élimination, reste la roulette de Goldberg. En revanche, la création de la population initiale n'est plus soumise à la contrainte de faisabilité qui caractérisait la méthode d'élimination. L'écueil majeur pénalisant l'algorithme du point de vue du temps de calcul est ainsi évité.

L'objectif n'est pas ici de mettre en œuvre une technique sophistiquée et particularisée sur certains exemples (plusieurs ont été citées dans l'analyse bibliographique précédente : coefficient dynamique ou avec analogie avec le recuit simulé, par exemple…), mais bien d'évaluer une méthode classique et couramment employée dans des problèmes similaires. Ainsi, une pénalité statique est retenue : la valeur du coefficient de pénalité ρ est fixée dès le début de l'exécution et ne varie pas au cours de la recherche. Bien entendu, plus la valeur de ρ est élevée, plus le poids accordé au second terme, c'est-à-dire à la minimisation de la violation de la contrainte, est important. A l'inverse, si le coefficient de pénalité prend des valeurs faibles, la recherche risque de tendre vers des solutions minimisant fortement le critère initial d'investissement, mais étant loin de satisfaire la contrainte temporelle. Plusieurs valeurs de ce coefficient ont donc été testées, de manière à trouver le meilleur compromis entre la qualité de la solution et sa faisabilité.

Tout d'abord, le premier objectif recherché, i.e. la réduction du temps de calcul, est atteint. En effet, le nombre d'évaluations de la fonction objectif est ici bien égal au produit du nombre de générations par la taille de la population, ce qui permet de ramener le temps de 35 secondes avec la technique d'élimination à un peu plus de 2 secondes. Concernant la qualité des résultats (cf. tableau 7), ils sont bien en accord avec les tendances prévues. Pour de faibles valeurs du coefficient de pénalité, la meilleure solution trouvée est très largement infaisable. Ceci explique la valeur de l'investissement correspondant, de très loin inférieure à celle de l'optimum. Au fur et à mesure que le coefficient de pénalité augmente, l'importance de la violation de la contrainte diminue. Parallèlement, le critère augmente logiquement jusqu'à arriver à des valeurs proches de celle de l'optimum lorsque la meilleure solution est faisable.

	$\rho = 10^{-4}$	$\rho = 10^{-2}$	$\rho = 1$	$\rho = 10^{2}$	$\rho = 10^{4}$
Meilleure critère pénalisé	257024	544547	620137	622799	622245
Investissement réel correspondant [$]	207820	486506	619464	622783	622245
Ecart à l'optimum (%)	-66,52	-21,61	-0,19	0,35	0,26
Violation contrainte (%)	369,7	40,15	0,43	0,01	0

Tableau 7 – Meilleure solution pour divers coefficients de pénalité

Aucune solution faisable n'est trouvée pendant l'exécution pour les deux premières valeurs de ρ (10^{-4} et 10^{-2}). Pour les trois plus grandes, l'évolution de la proportion de solutions faisables dans la population au cours de la recherche est montrée sur la figure 13.

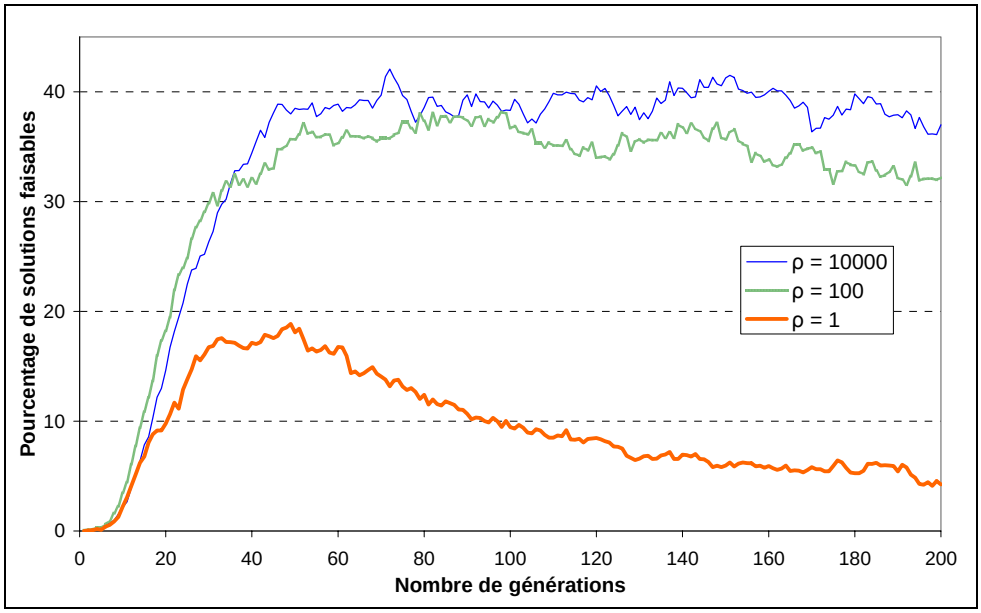


Figure 13 – Evolution du nombre de solutions faisables

Après une cinquantaine de générations, cette proportion stagne entre 30 % et 40 % pour $\rho = 10^{2}$ et $\rho = 10^{4}$ mais chute quand ρ est égal à 1. Avec ce dernier coefficient de pénalité, la

meilleure solution faisable trouvée est égale à 621973, ce qui est une valeur tout à fait intéressante. Elle n'apparaît logiquement pas sur le tableau 7 dans la mesure où le meilleur critère correspond à une solution (légèrement) infaisable.

Finalement, les dispersions à 2 % et 5 % augmentent quand la valeur du coefficient de pénalité diminue (figure 14) puisqu'il est plus facile de trouver des solutions à faibles valeurs du critère. Pour $\rho = 10^{-4}$, la valeur des dispersions devient ainsi comparable à celle obtenue par la technique d'élimination. Pour $\rho = 1$, il est intéressant de noter que la dispersion ramenée aux solutions faisables trouvées à chaque exécution est assez élevée : 58 % des exécutions trouvent une solution faisable à moins de 2 % de la valeur 621973 relevée plus haut, et 100 % à moins de 5 %.

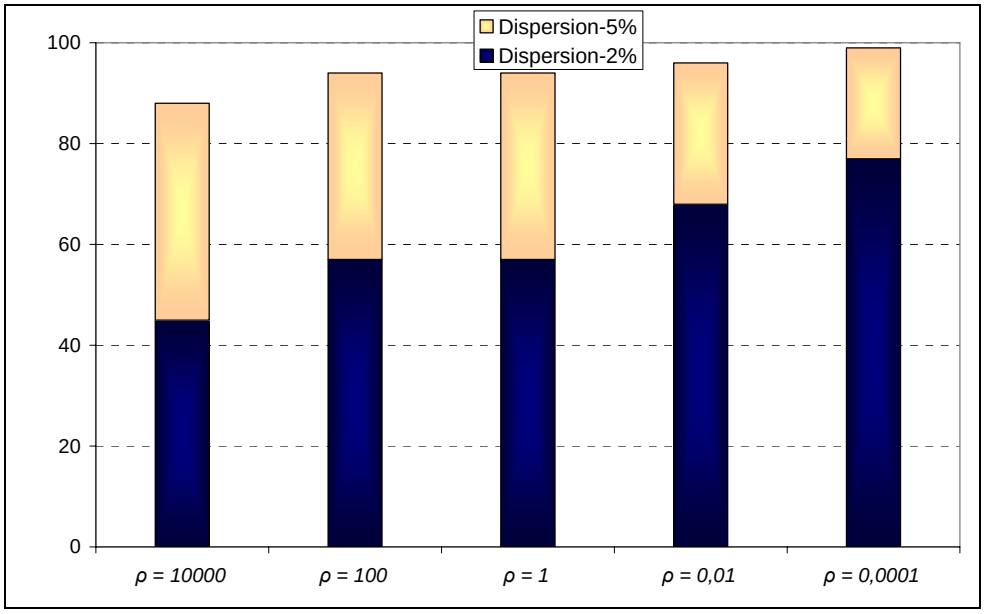


Figure 14 – Dispersions pour les différentes valeurs de ρ

Ainsi, l'efficacité de ce mode de gestion est démontrée en termes de temps de calcul. Certaines valeurs du coefficient de pénalité fournissent même des dispersions satisfaisantes. Mais le choix du facteur de pénalité optimal est gêné par l'antagonisme entre qualité du critère et faisabilité de la solution. Il ne faut pas négliger cette seconde caractéristique, car si elle n'est pas essentielle pour la détermination d'une solution proche de l'optimum dans cet exemple de taille moyenne, elle le devient lorsque la complexité des problèmes augmente. Ceci amène donc à retenir préférentiellement les grandes valeurs du coefficient de pénalité. En passant de $\rho = 10^4$ à $\rho = 10^2$, le gain en dispersions à 2 % et 5 % est élevé sans perdre un nombre excessif de solutions faisables, ce qui conduit à penser que le compromis le plus approprié est la seconde option.

3.2.2 – Relaxation des bornes supérieures discrètes

L'idée de cette approche provient de l'antagonisme entre la contrainte temporelle, imposant de produire une quantité donnée des I produits en un temps fixé, et la limite supérieure du nombre admissible d'équipements par étape. En effet, une augmentation de la borne supérieure sur m_j et n_k (cf. chapitre précédent) signifie que le nombre d'équipements en parallèle par étape opératoire est moins restreint : d'un point de vue physique, cela engendre une productivité plus élevée et facilite donc la satisfaction de la contrainte de production.

Par conséquent, si la borne supérieure N_{max} est relaxée par un entier supérieur à sa valeur initiale, la localisation – de manière aléatoire – de solutions satisfaisant la contrainte pour construire la population initiale est facilitée. Cette technique correspond en fait à repousser un peu les limites de l'espace faisable. Par ailleurs, on sait que, par la suite, la minimisation du critère de coût par l'algorithme génétique fera tendre le nombre d'équipements en parallèle vers de faibles valeurs, à l'intérieur de l'espace faisable initial.

La borne supérieure initiale N_{max} vaut 3 pour cette instance. Comme cette dernière est relativement peu complexe, on fait varier N_{max} entre 4 et 6 afin de vérifier la bonne influence de l'AG sur la faisabilité des solutions, justifiant la validité de la méthode de relaxation de la borne supérieure sur les variables discrètes. Les graphiques de la figure 15 représentent l'évolution, au cours des générations, de la proportion de solutions faisables (a) au sens large, i.e. ne respectant que la contrainte temporelle ; et (b) au sens strict, respectant la borne supérieure initiale sur les nombres d'équipements en parallèle (en plus de la contrainte de production). Il est bien entendu que, de toute façon, la borne supérieure relaxée est respectée puisqu'intégrée dans le codage.

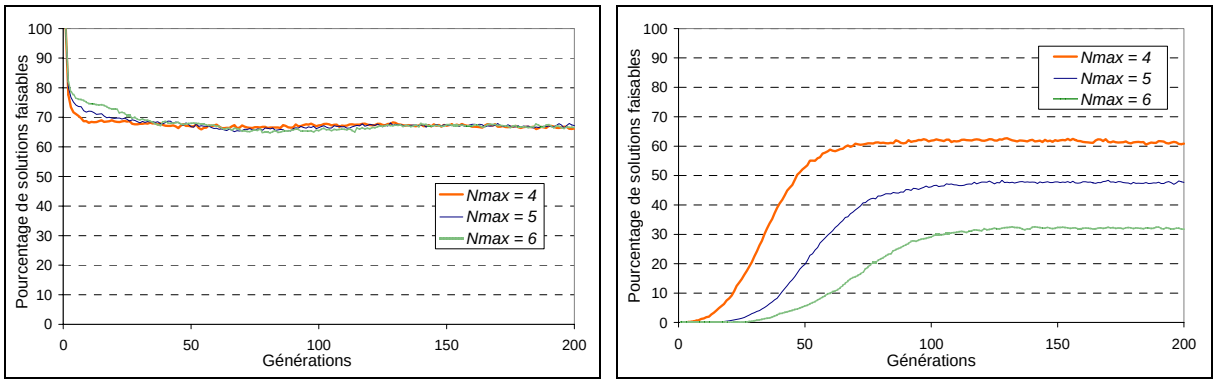


Figure 15 – Evolution du pourcentage de solutions faisables
(a) au sens large (b) au sens strict

Le comportement de la roulette de Goldberg, impliquant la création d'une population initiale entièrement faisable (ici au sens *large*), est observable sur la figure 15a. Les courbes

correspondant à N_{max} = 4, 5 et 6 sont pratiquement confondues. Le pourcentage de solutions faisables au sens large diminue inévitablement au fur et à mesure que les procédures stochastiques de croisement et de mutation créent des individus infaisables. Dans les trois cas testés, cette proportion se stabilise entre 65 % et 70 % dès la 40ème génération, très en deçà du nombre de générations maximal.

Sur la figure 15b, la proportion de solutions faisables au sens strict suit logiquement une évolution opposée. La population initiale ne comprend aucune solution respectant la limite de 3 équipements du problème initial, ce qui est l'effet recherché. Puis l'algorithme génétique dirige vraisemblablement la recherche vers une minimisation du nombre d'équipements puisque les trois courbes s'élèvent pour atteindre finalement un palier. La valeur de ce palier dépend de l'importance de la relaxation de la borne supérieure : la pression vers l'espace réellement faisable est bien entendu d'autant moins forte que N_{max} est élevé. Les valeurs des paliers sont ainsi égales à 32 %, 48 % et 62 % pour une borne supérieure relaxée respectivement à 6, 5 et 4 équipements par étape opératoire.

Remarque : il est à noter qu'aucun échec n'est constaté pour les trois cas de relaxation. Ceci confirme la capacité à réduire le nombre d'équipements en parallèle tout en respectant la contrainte temporelle de production.

En se penchant sur la qualité des résultats présentés dans le tableau 8, les meilleures solutions trouvées sont toutes très proches de l'optimum et ne permettent donc pas de tirer de conclusions. En revanche, la figure 16 affirme la meilleure fiabilité des relaxations plus faibles. Cet effet est peu étonnant dans la mesure où leur plus grande sévérité rapproche leur comportement de la technique d'élimination, également représentée à titre de référence. Néanmoins, ceci n'empêche pas la dispersion à 2 % de chuter de moitié entre la technique d'élimination et celle de relaxation avec N_{max} = 4, prouvant une perte d'efficacité conséquente. Pour la valeur extrême (N_{max} = 6), les résultats sont finalement loin d'être satisfaisants.

	N_{max} = 4	N_{max} = 5	N_{max} = 6
Meilleure solution [$]	622433	621322	621906
Ecart à l'optimum (%)	0,29	0,11	0,20
Temps CPU [s]	4,5	2,7	2,3

Tableau 8 – Qualité des solutions pour la technique de relaxation

L'option la plus viable semble donc être une relaxation assez faible de la borne supérieure, mais celle-ci a un prix en temps de calcul. La dernière ligne du tableau 8 indique que celui-ci est presque divisé par deux en passant de N_{max} = 4 à N_{max} = 6. Certes, l'écart

absolu est relativement peu important mais il est facilement imaginable que l'ampleur de cette différence serait toute autre sur un exemple de plus grande taille. Il n'est en outre pas inutile de rappeler que le temps nécessaire à la résolution du même exemple par la technique d'élimination était globalement de huit à quinze fois plus élevé (35 secondes), ce qui souligne l'atteinte de l'objectif visant à améliorer la rapidité de l'exécution de l'AG.

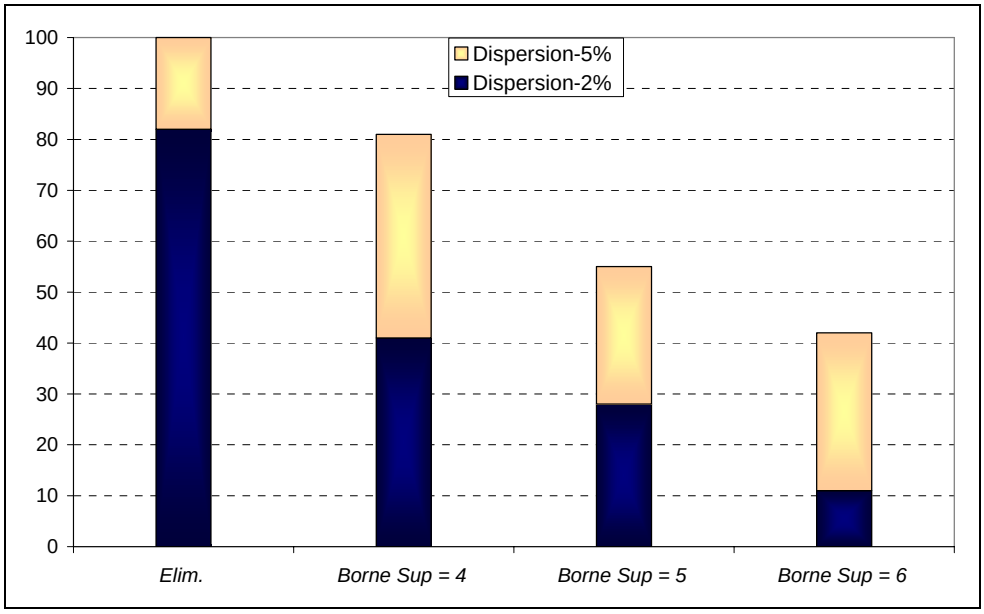


Figure 16 – Dispersions à 2 % et 5 %

Les écarts de temps de calcul entre les différentes options envisagées peuvent comme précédemment être mis en relation avec le nombre d'individus visités avant de pouvoir créer une population initiale faisable (voir figure 17) : cette dernière valeur est d'autant plus faible que la relaxation de la borne supérieure s'accentue. Pour le dernier cas (N_{max} = 6), le nombre d'individus infaisables testés pour générer la population initiale est divisé par 100 par rapport à la technique d'élimination, devenant ainsi pratiquement négligeable devant le nombre total d'évaluations de la fonction objectif.

3.2.3 – Tournoi basé sur des notions de dominance

La méthode employée ici est en tous points semblable à celle proposée dans [COE02c]. Elle repose sur la distinction faite entre solutions faisables et solutions infaisables établie par les règles de dominance évoquées dans la partie 3.1.2. Ces règles sont implémentées au sein d'un tournoi servant comme procédure de sélection. Chaque tournoi fonctionne en choisissant au hasard *NPart* participants. Les règles de dominance permettent de déterminer les *NVainq* vainqueurs, qui seront les individus de la génération en cours qui vont survivre. Un nombre de tournois suffisant pour atteindre le nombre d'individus survivants (fixé par les paramètres de l'AG) est réalisé à chaque étape de sélection.

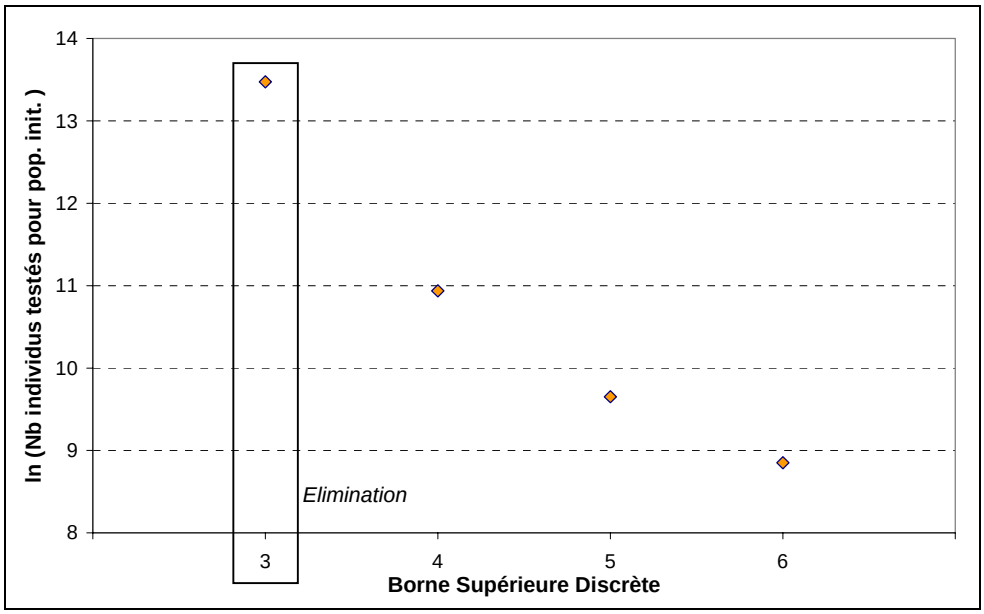


Figure 17 – Nombre d'individus visités pour créer la population initiale

Bien que, de manière intuitive, il soit prévisible que la pression de la sélection sera d'autant plus forte que l'écart entre *NPart* et *NVainq* sera élevé, diverses combinaisons ont été testées pour déterminer le jeu le plus performant de paramètres. Une option particulière envisagée, appelée *tournoi unique* (T.U.) par la suite, consiste à effectuer un seul tournoi par étape de sélection, ce qui revient à affecter la taille de population à *NPart* et le nombre de survivants à *NVainq*. Le temps de calcul, logiquement identique pour tous les cas considérés, est égal à 2 secondes. En effet, comme pour la méthode de pénalisation, les nombres effectif et théorique d'évaluation du critère effectif sont bien égaux.

Les résultats du tableau 9 montrent que la meilleure solution obtenue est, pour chaque version du tournoi, très proche de l'optimum (entre 0,23 % et 0,65 %). Ce critère ne permet pas de tirer de conclusions significatives sur la supériorité de certaines versions sur les autres. Par contre, la dernière ligne de ce tableau met d'ores et déjà en évidence les faiblesses des combinaisons les moins sélectives *a priori* : le nombre d'échecs est non nul pour le tournoi *(3,2)* et concerne près de la moitié des exécutions pour le tournoi *(5,4)*. Vu la complexité moyenne de l'exemple 5, cette tendance n'est guère encourageante pour l'application à des instances de plus grande taille.

Les deux critères qui permettent alors de départager les différentes options entre elles sont donc les dispersions à 2 % et à 5 %, ainsi que le nombre de solutions faisables. Il est vrai que ce nombre de solutions faisables n'est pas forcément significatif dans cet exemple, dans la mesure où sa difficulté modérée n'empêche pas de trouver, quel que soit le cas de figure, des solutions quasi-optimales. En revanche, ce critère prend toute son importance lorsque,

pour des exemples de taille supérieure, la localisation d'individus faisables proches de la solution est une condition pour la détermination d'un résultat de bonne qualité.

	(NPart=2,NVainq=1)	*(3,1)*	*(3,2)*	*(4,1)*
Meilleure solution [$]	622269	622921	622246	624671
Ecart à l'optimum (%)	0,26	0,37	0,26	0,65
Pourcentage d'échecs	0	0	7	0

	(4,2)	*(5,1)*	*(5,2)*	*(5,4)*	*T.U.*
Meilleure solution [$]	622035	623819	623840	622638	623093
Ecart à l'optimum (%)	0,23	0,51	0,52	0,32	0,40
Pourcentage d'échecs	0	0		45	0

Tableau 9 – Résultats pour les différentes versions de tournoi

Le graphique de l'évolution des solution faisables au fil des générations, disponible sur la figure 18, semble confirmer que les tournois *(3,2)* et *(5,4)* sont peu performants. Pour ces deux versions, la proportion de solutions faisables dans la population reste faible. Il faut cependant nuancer cette remarque car le nombre de solutions faisables apparaissant sur le graphique est en fait une moyenne, calculée sur les 100 exécutions de l'AG sans tenir compte des échecs. Cela signifie, par exemple que pour le tournoi *(5,4)*, caractérisé par un taux d'échec égal à 45 %, il faut diviser la proportion d'individus faisables montrés sur la figure 18 par (1–0,45) pour obtenir la proportion moyenne d'individus faisables dans une exécution sans échec.

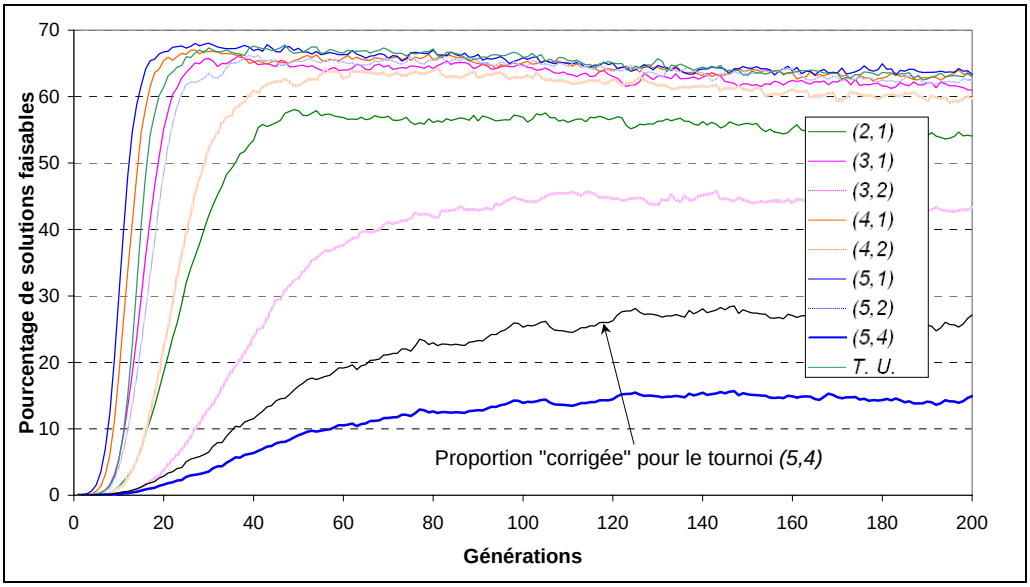


Figure 18 – Evolution du pourcentage de solutions faisables

Cette proportion « corrigée » pour le tournoi *(5,4)*, toujours inférieure à 30 %, est illustrée sur la figure 18 à titre d'exemple, comme cela aurait également pu être fait pour le tournoi

(3,2). Cette analyse met en évidence une double inaptitude de cette version du tournoi. Tout d'abord, elle est incapable de repousser les individus vers la région faisable lorsque celle-ci est encore indéterminée ; mais aussi, quand un point faisable a été trouvé, elle ne parvient pas à pousser la population vers la zone déjà repérée pour y intensifier le calcul.

En ce qui concerne les autres combinaisons, toutes semblent assez performantes. Elles montrent des pourcentages de solutions faisables pratiquement identiques au cours de l'évolution de la recherche, bien que les tournois *(2,1)* et *(4,2)* soient un peu en deçà des autres. Il est à remarquer que les positions des courbes peuvent être ordonnées selon le rapport croissant *NPart/NVainq* : on obtient plus de solutions faisables pour des tournois caractérisés par un rapport plus élevé, ce qui est conforme à la prévision réalisée antérieurement.

L'histogramme des dispersions (figure 19) confirme la mauvaise qualité du tournoi *(5,4)*. Une analyse comparable à la précédente, corrigeant les valeurs des dispersions en fonction du taux d'échec, peut être effectuée. Malgré tout, elle ne change pas les conclusions négatives formulées précédemment. Par contre, la combinaison *(5,1)*, i.e. la plus performante en termes de faisabilité des solutions, arrive en avant-dernière position. Ce comportement décevant peut être expliqué par la trop grande importance donnée à l'intensification, puisque moins de chances sont données à des individus faibles de passer avec succès l'étape de sélection.

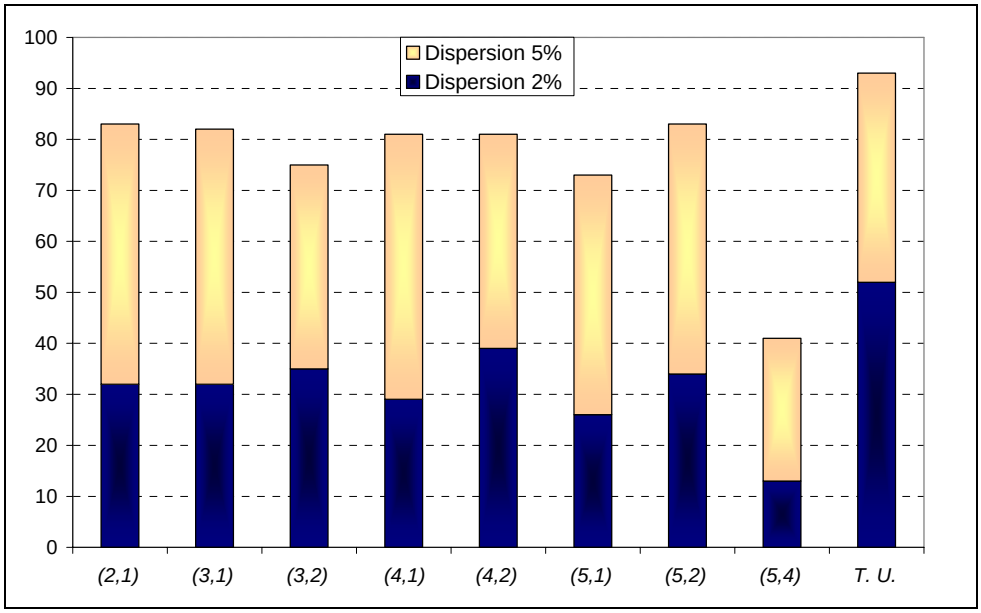


Figure 19 – Dispersions pour les différentes versions de tournoi

Pour les meilleures combinaisons, la méthode de tournoi unique montre une supériorité indéniable sur les autres, puis vient la version *(4,2)*. Entre ces deux extrêmes, les qualités statistiques des résultats fournis par les autres versions de tournoi sont plus ou moins similaires. Le compromis le plus avantageux semble donc être sans équivoque la méthode du

tournoi unique. Ses performances, tant en termes de dispersion que de solutions faisables rencontrées au cours de la recherche, font partie des meilleures parmi les différentes versions de tournoi testées. En revanche, à titre de comparaison avec les résultats obtenus avec la méthode d'élimination, on constate que la qualité globale n'est pas aussi bonne que celle de cette dernière.

Enfin, il est important de noter que le fonctionnement du tournoi unique supprime l'effet stochastique du tournoi. Celui-ci est normalement obtenu par le tirage aléatoire parmi la population des individus participants. Au contraire, dans le cas du tournoi unique, seuls les meilleurs individus sont sélectionnés. En outre, son mode opératoire sous-tend que la pression dirigeant vers la minimisation de la fonction objectif n'est vraiment exercée qu'à partir du moment où tous les individus survivants sont faisables. Ces deux aspects risquent d'amener à une dégradation de la diversité, pénalisante dans le cas de problèmes plus contraints.

Pour éviter ce problème, il est conseillé dans [DEB00] d'employer une technique de *niching* (cf. 3.1.1). Dans [COE02c], une autre méthode de maintien de la diversité des individus lui est préférée. Elle consiste à appliquer un tournoi *(2,1)* avec une probabilité S_r. Dans le cas où le nombre aléatoire tiré est supérieur à S_r, le vainqueur du tournoi est désigné aléatoirement. C'est cette méthode qui est retenue et adaptée au tournoi unique utilisé ici. En accord avec les règles de dominance déjà citées, les meilleurs individus sont déterminés. Puis, chaque survivant est choisi parmi les vainqueurs du tournoi unique avec une probabilité égale à S_r. Si le nombre aléatoire tiré est supérieur à S_r, le survivant est choisi aléatoirement parmi la population entière.

Il est suggéré dans [COE02c] d'employer des valeurs de S_r comprises entre 0,8 et 1. Divers tests ont été effectués pour tester l'influence de ce paramètre, seulement dans le cas du tournoi unique. Les diagrammes de la figure 20 montrent que les résultats obtenus ne sont pas convaincants, puisque tant pour la dispersion que pour le nombre de solutions faisables, le tournoi unique simple (S_r = 1) est le plus efficace.

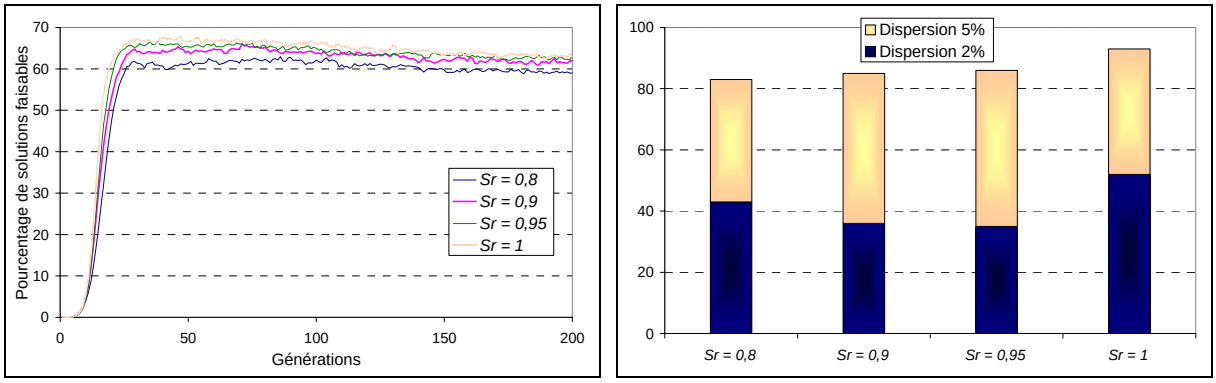


Figure 20 – Etude sur S_r

Mais l'inefficacité de cette technique de diversification ne doit pas être généralisée : l'instance servant d'exemple-test étant d'une complexité moyenne, favoriser la diversification n'est pas nécessaire pour trouver de bons résultats. En revanche, cette technique sera utilisée par la suite, pour la résolution d'exemples de très grande taille.

3.2.4 – Stratégie multiobjectif

La dernière option envisagée pour une gestion efficace des contraintes est une stratégie multiobjectif. Comme cela a été précisé dans l'analyse préliminaire, cette méthode consiste purement et simplement à considérer la violation de la contrainte comme une deuxième fonction objectif à minimiser. Le problème résultant est donc de type bicritère et non contraint. Le premier critère est la fonction objectif initiale F_1 et le deuxième s'écrit sous forme quadratique :

$$F_2(x) = (H - \sum_{i}^{I} H_i)^2 \qquad (8)$$

La prise en compte des deux critères dans une stratégie d'optimisation multiobjectif induit une modification de la technique de sélection initiale. La nouvelle méthode consiste en *deux* roulettes de Goldberg servant, avec l'effet probabiliste habituel, à retenir les meilleurs individus pour chaque critère. Le taux de survie est alors divisé en deux, chacune des procédures de roulette servant à la sélection de la moitié des individus survivants.

Remarque : Dans des stratégies multiobjectif classiques, l'objectif, s'inscrivant dans une optique d'aide à la décision, est l'obtention du meilleur compromis entre des critères antagonistes. Il a été montré dans [AGU05] que cette méthode des « deux roulettes » n'est pas la plus efficace car chacune des roulettes « tire » les individus dans une direction parallèle à l'axe du critère auquel elle correspond. Le compromis n'est cependant pas l'effet recherché ici.

Une fois le critère d'arrêt atteint, un tri est effectué sur toutes les solutions visitées au cours de la recherche pour ne garder que les individus non-dominés au sens de Pareto. Le diagramme obtenu pour une exécution de l'algorithme est montré sur la figure 21. Le front de Pareto, composé de 104 solutions, est très nettement visible. Cependant, le nombre de solutions finales non-dominées et faisables est nul. La solution violant le moins largement la contrainte (10 %) est à presque 25 % de l'optimum du problème initial. Les deux solutions les plus proches de la valeur optimale violent la contrainte de 56 % et 58 %, tout en proposant une diminution du critère « coût » de 2,36 % et 3,86 % respectivement. Ces solutions peuvent

certes être utilisées dans un objectif d'aide à la décision, mais ne sont que peu intéressantes dans le cadre de cette étude.

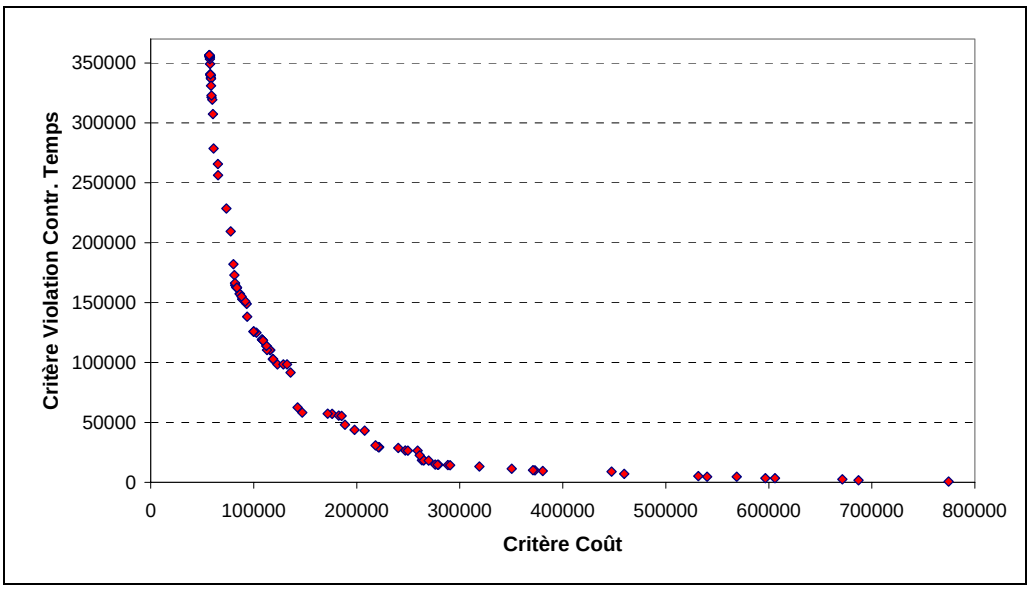


Figure 21 – Solutions non-dominées au sens de Pareto

Par ailleurs, le temps de calcul (essentiellement dû au tri de Pareto final) pour une exécution est de l'ordre de 20 minutes, ce qui est très élevé par rapport aux autres techniques. Les résultats ne semblent donc pas concluants pour cette méthode, qui n'a pas été retenue pour la suite des calculs.

3.2.5 – Conclusions

En conclusion de cette étude sur les techniques de gestion des contraintes au sein d'un algorithme génétique, rappelons tout d'abord que les objectifs étaient de réduire le temps de calcul (par des stratégies ne nécessitant pas la création d'une population initiale entièrement faisable) tout en conservant, dans la mesure du possible, la qualité des résultats, jusque là satisfaisante. Le tableau 10 résume les performances des différentes techniques testées. La distance de la meilleure solution trouvée à l'optimum n'y apparaît pas car il a été démontré que toutes les techniques, à l'exception de la stratégie multiobjectif, étaient équivalentes sur ce point.

Trois des quatre méthodes abordées nécessitent le réglage de paramètres de fonctionnement : coefficient de pénalité, borne relaxée pour les variables discrètes ou modalités de tournoi… ce qui constitue un désavantage vis-à-vis de la méthode d'élimination. En revanche, si l'on excepte le cas particulier du fonctionnement multicritère, l'objectif de réduction du temps de calcul est atteint pour toutes les méthodes envisagées. Les méthodes de

pénalisation et de tournoi conduisent à des temps de calcul comparables et nécessairement les plus faibles étant donné que le nombre d'évaluations de la fonction objectif est réduit à sa valeur théorique. Pour la relaxation, la perte d'efficacité générée par la création d'une population initiale faisable (au sens large) est limitée, mais elle le serait sûrement moins pour des instances plus complexes que celle qui a servi d'exemple-test.

	Dispersions	Faisabilité	Temps CPU	Paramètre à fixer
Elimination	+ + +	+ +	– –	non
Pénalisation	+	–	+ +	oui
Relaxation	–	–	+	oui
Tournoi classique	+	+ +	+ +	oui
Tournoi unique	+ +	+ +	+ +	non
Multiobjectif	– –	– –	– –	non

Tableau 10 – Récapitulatif sur la gestion des contraintes

Concernant la qualité des solutions, la première conclusion à tirer est qu'aucune des méthodes abordées ne parvient à égaler la technique d'élimination. Ces calculs justifient donc le fonctionnement par élimination dès le moment où le temps ne constitue pas un goulot d'étranglement pour le calcul. La qualité des résultats fournis par la technique de pénalisation est très intéressante pour deux valeurs du coefficient de pénalité, mais le nombre de solutions faisables n'est pas convaincant. La méthode de relaxation de la borne supérieure des variables discrètes donne dans certains cas des solutions acceptables en qualité mais au prix d'un temps de calcul qui, on l'a dit, pourrait devenir rédhibitoire. En outre, la nécessité de jouer sur la valeur de la borne supérieure pour déterminer le meilleur compromis entre temps de calcul et qualité de la solution est peu engageante.

Quant aux tournois basés sur des notions de dominance, la meilleure option est le tournoi unique. Ce dernier implique une seule procédure de tournoi par étape de sélection et ne nécessite donc l'ajustement d'aucun paramètre. L'observation du nombre des solutions faisables visitées, de la dispersion des résultats donnés par les différentes exécutions, comme celle de la performance en temps de calcul ont finalement amené à retenir cette méthode pour la suite des calculs. Les instances 7 à 11 seront dès lors traitées au moyen de ce mode de gestion des contraintes.

3.3 – Ateliers 7 à 11

L'exemple 7 a déjà été résolu avec une technique d'élimination en tant que mode de gestion des contraintes, mais il est de nouveau traité dans cette partie pour valider la nouvelle approche par tournoi unique. Puis, les résultats sur les instances de plus grandes tailles sont exposés pour achever finalement les calculs sur le jeu d'exemples fixé dans le protocole

opératoire retenu. Alors qu'il n'apparaissait pas auparavant, le taux d'échec pour les exécutions est utilisé ici comme un nouveau critère pour juger de l'efficacité de l'algorithme génétique. Finalement, les conclusions des calculs effectués avec l'AG sont présentées.

3.3.1 – Résultats

En accord avec les conclusions de la partie précédente, le tableau 11 met en relief sur l'exemple 7 la supériorité de la gestion des contraintes par élimination sur celle de tournoi unique en ce qui concerne la qualité de l'ensemble des exécutions (i.e. les dispersions car les meilleures solutions trouvées sont équivalentes). En revanche le temps de calcul est sans équivoque à l'avantage du tournoi unique.

	Elimination	Tournoi unique
Meilleure solution [$]	970376	967672
Ecart à l'optimum (%)	1,36	1,09
Dispersion 2 %	25	16
Dispersion 5 %	95	52
Taux d'échec (%)	-	1
Temps CPU	1,5 [h]	17 [s]

Tableau 11 – Elimination et tournoi unique sur l'exemple 7

Le tableau 12 récapitule les résultats pour les exemples 7 à 11, tous résolus avec une technique de tournoi unique comme mode de gestion des contraintes. L'évolution de l'écart entre les meilleurs résultats trouvés et l'optimum est tracée sur la figure 22. Elle illustre bien les difficultés qu'éprouve l'AG, au fur et à mesure de la complexification des problèmes traités, à s'approcher de la solution optimale. Il est effectivement de plus en plus compliqué de déterminer, avec l'augmentation de la taille des exemples, le jeu de variables discrètes correspondant à l'optimum trouvé par *SBB*. L'AG n'y parvient d'ailleurs pas pour les instances 8 à 11.

Remarque : la technique de diversification proposée dans la partie sur la gestion des contraintes (voir 3.2.3), basée sur une application probabiliste du tournoi, a été appliquée pour les instances 11 et 12. Quelques tests ont en effet permis de montrer que le choix d'une probabilité $S_r = 0{,}95$ amenait à une meilleure qualité des résultats.

	At. 7	At. 8	At. 9	At. 10	At.11
Meilleure solution [$]	967672	1963746	2971715	3988974	5000234
Temps CPU	17 [s]	50 [s]	17 [min]	50 [min]	4,6 [h]

Tableau 12 – Résultats pour la deuxième partie du jeu d'exemples

La baisse de l'efficacité de l'AG est largement imputable au nombre de moins en moins important de solutions faisables rencontrées au cours de l'exécution. D'une manière générale, l'écueil principal consiste en fait à trouver une solution faisable au début de la recherche, écueil qui engendre une proportion élevée d'échecs.

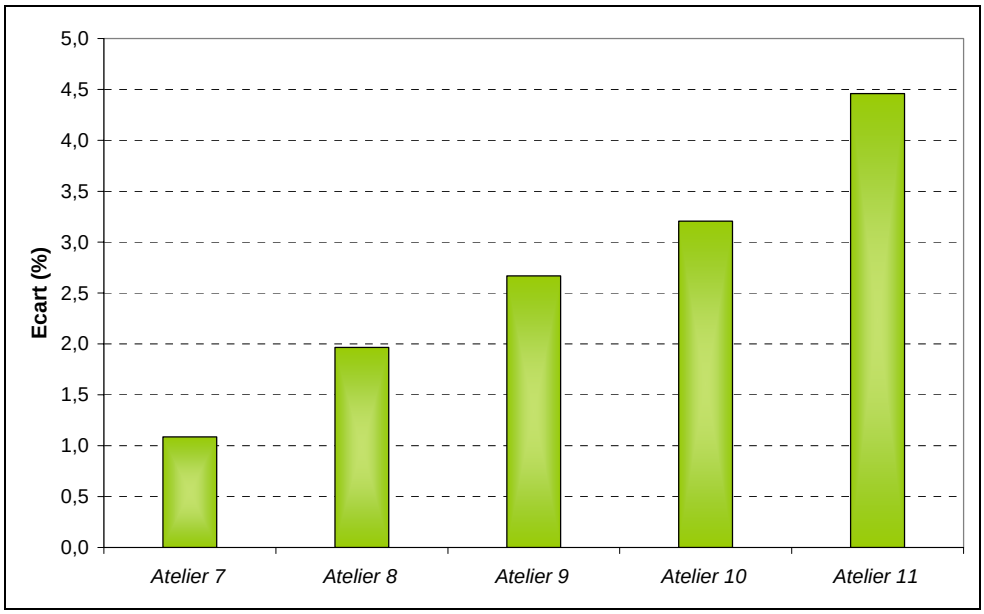


Figure 22 – Ecart entre la solution de l'AG et l'optimum de *SBB*

Ces tendances sont tracées en pointillés sur la figure 23, en excluant l'atelier 9 qui ne les respecte pas. Il est en effet possible de constater que les résultats de cette instance sont nettement meilleurs que pour l'atelier 8, allant à l'encontre de la baisse générale des performances de l'AG sur ces derniers exemples.

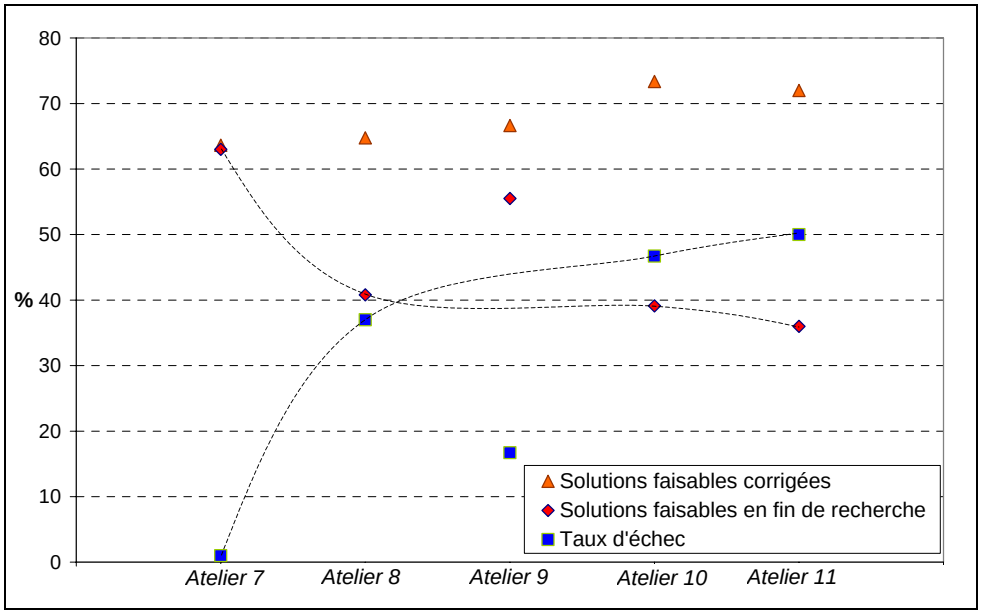


Figure 23 – Taux d'échec et solutions faisables en fin de recherche

La correction effectuée, ne prenant en compte que les exécutions réussies, démontre que la proportion de solutions faisables en fin de recherche reste plus ou moins constante, quelle que soit l'instance abordée. Cette remarque confirme la responsabilité du nombre d'échecs dans la défaillance de l'algorithme : s'il n'était pas pénalisé par ces derniers, les performances de l'AG serait bien plus satisfaisantes.

Ces conclusions s'étendent à la qualité générale des exécutions puisqu'une analyse similaire peut être réalisée relativement aux dispersions à 2 % et 5 %. En incluant les échecs, la figure 24a révèle une chute progressive de la régularité de l'AG en termes de qualité de la solution trouvée, mis à part pour l'atelier 9. Sur le dernier exemple, le meilleur résultat trouvé est à 4,46 % de l'optimum et aucune autre exécution ne trouve de solution à moins de 2 % de ce résultat, lui-même éloigné du minimum. La correction effectuée relativement au nombre d'échecs, apparaissant sur la figure 24b, permet néanmoins de nuancer ce bilan assez négatif : si les dispersions à 2 % restent globalement faibles, au moins la moitié des exécutions trouvent un résultat à moins de 5 % de la meilleure solution trouvée. Une fois encore, cette correction selon le taux d'échec atténue la différence des résultats de l'exemple 9 avec les autres. Cependant, ces corrections ne doivent pas masquer les défaillances de l'AG : les échecs sont indéniablement une preuve de son manque d'efficacité.

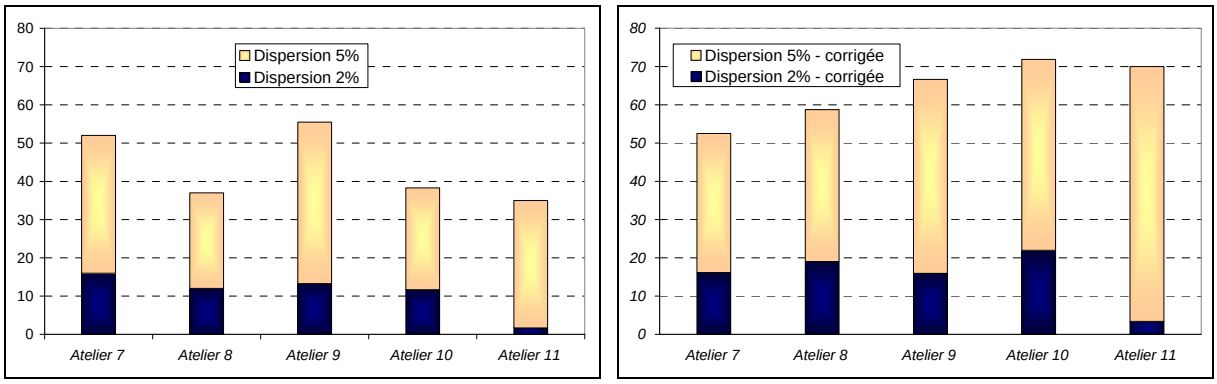


Figure 24 – Dispersions à 2 % et 5 %
(a) Sans correction **(b) Avec correction**

En ce qui concerne le temps de calcul, s'il augmente fortement au fil des exécutions, il reste tout de même comparable au temps nécessaire au module *SBB* pour la résolution et l'AG est même le plus performant sur la dernière instance. L'évolution, sous forme logarithmique, des temps de calcul caractéristiques des deux méthodes d'optimisation pour ces derniers exemples est donnée sur la figure 25.

Finalement, les résultats de l'AG seraient globalement positifs si la difficulté à localiser des solutions faisables en début de recherche ne représentait un obstacle si pénalisant. En effet, les corrections effectuées, prenant en compte le taux d'échec pour chaque instance,

montrent que le nombre de solutions faisables et les dispersions sont relativement stables au fil des exemples. Dès lors qu'il trouve un individu faisable, l'AG a donc un comportement lui permettant d'atteindre avec une certaine fiabilité des solutions à moins de 5 % de la valeur optimale.

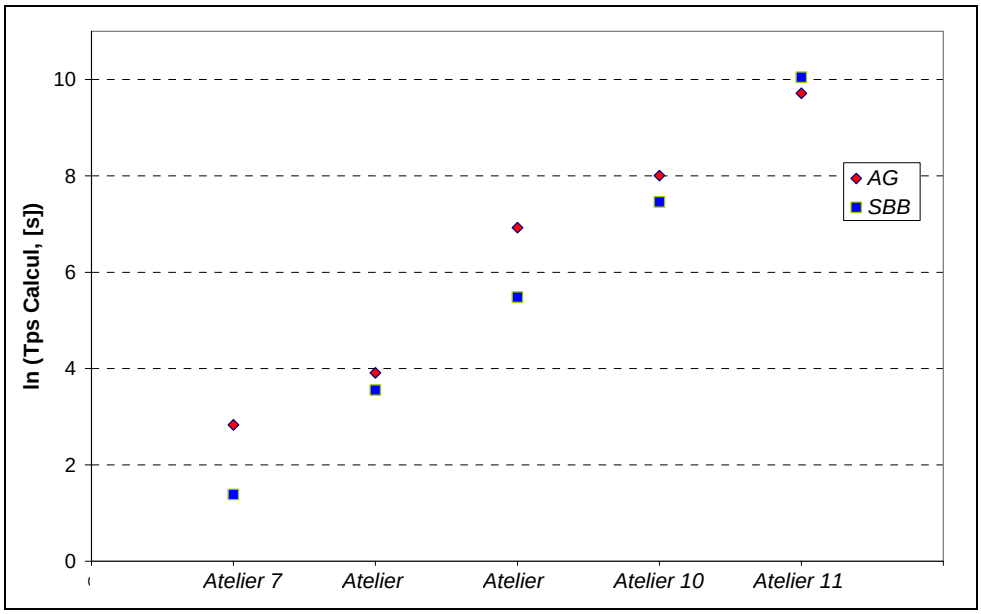


Figure 25 – Evolution du temps de calcul pour l'AG et pour *SBB*

Néanmoins, cet assujettissement à la localisation d'un individu faisable est contraignante : pour le dernier exemple, une exécution de 4,6 heures ne permet pas d'assurer l'obtention d'un quelconque résultat. Le mode opératoire du tournoi semble cependant construit pour pallier ce genre de difficulté, puisqu'il tend à exercer une pression sur les solutions infaisables en direction de la région faisable.

3.3.2 – Conclusions

Finalement, en conclusion sur les calculs effectués avec l'algorithme génétique, les performances de ce dernier sont assez hétérogènes sur l'ensemble du jeu d'exemples défini préalablement. Il est possible de distinguer différents groupes d'instances révélant un comportement particulier de l'AG :

- Tout d'abord sur les exemples 1, 5 et 6, l'AG obtient sans conteste des résultats très satisfaisants. La meilleure solution trouvée est à moins de 0,5 % de l'optimum donné par les méthodes de programmation mathématique. Les dispersions à 2 % et 5 %, proches de 90 % et 100 % respectivement, assurent une bonne répétabilité des résultats.

- Puis l'AG est confronté à deux exemples, les ateliers 3 et 7, pour lesquels il trouve des solutions encore proches de l'optimum mais qui posent problème. En effet, les résultats sur ces instances sont caractérisés par une baisse relative des dispersions à 2 % et 5 %. Pour l'exemple 7, qui a été traité au moyen de deux méthodes de gestion des contraintes, cette tendance est confirmée puisque les résultats concordent dans les deux cas.

- Un troisième groupe est composé des exemples 2 et 4 : pour ces deux derniers, et malgré leur taille plus que raisonnable, l'AG reste bloqué sur des optima locaux, se distinguant par des jeux de variables entières différents de ceux des optima déterminés par *SBB*. Malgré la procédure classique de diversification par mutation et le brassage des gènes impliqué par le croisement, la métaheuristique n'est pas capable de s'en extraire comme le prouve l'excellent niveau des dispersions à 2 % et 5 % : l'AG retombe pratiquement toujours sur ces valeurs sous-optimales.

- Enfin, le dernier ensemble d'exemples se compose des quatre ateliers de plus grandes tailles. Pour ces derniers, l'AG trouve des solutions de plus en plus éloignées de l'optimum, avec une dispersion des exécutions de moins en moins bonne. Les jeux de variables entières des solutions trouvées diffèrent de ceux caractérisant l'optimum. L'analyse précédente des résultats a permis de démontrer que les performances de l'AG sur ces exemples sont lourdement pénalisées par la difficulté à trouver une première solution faisable.

Il apparaît donc finalement que la complexité des exemples n'est pas nécessairement celle qu'on aurait pu imaginer au début. La taille des instances traitées n'explique pas toujours les difficultés rencontrées par l'AG pour trouver de manière efficace une bonne solution. Les quatre groupes d'exemples présentés ci-dessus semblent plus représentatifs d'une complexité croissante.

En ce qui concerne le temps de calcul, l'AG est généralement assez performant : sur les premiers exemples, un lancement équivaut à une exécution d'un des solveurs de *GAMS*. Puis le temps de calcul de l'AG augmente rapidement à cause de la gestion des contraintes par élimination. En revanche, dès l'exemple 7, pour lequel le tournoi unique a été adopté, les temps de calcul de l'AG redeviennent comparables à ceux du solveur *SBB*. Cela dit, la comparaison des temps de calcul des méthodes déterministes et de l'AG ne peut négliger le caractère stochastique de ce dernier. L'atteinte du meilleur résultat trouvé par l'AG est conditionnée par les dispersions des exécutions, ce qui s'exprime, par exemple pour l'atelier 5 : « une exécution de l'AG durant 35 secondes peut atteindre une solution valant 622566 [$] (i.e. à 0,30 % de l'optimum) et promet de trouver un résultat à moins de 2 % (resp. 5 %) de

cette valeur avec une sécurité de 82 % (resp. 100 %) ». Il a été vu que sur les derniers exemples, une exécution ne garantit même pas l'obtention d'une solution faisable, le taux d'échec atteignant 50 % pour l'atelier 11 : le temps de calcul intéressant de l'AG doit alors être relativisé en fonction de ses performances sur l'exemple considéré.

Ces analyses ont mis en évidence la nécessité d'améliorer la qualité des résultats de l'algorithme et plus particulièrement de régler le problème posé par la vérification du jeu de variables discrètes, empêchant l'AG d'atteindre des solutions plus proches de l'optimum. Un travail de réflexion a dirigé les efforts sur le codage et ainsi amené à proposer des modifications qui sont présentées dans la partie suivante.

4 – Modifications du codage

Les parties précédentes ont montré la supériorité des méthodes déterministes sur l'algorithme génétique en ce qui concernait la qualité des solutions obtenues. En effet, si le module *SBB* parvient à fournir des solutions optimales pour les onze exemples étudiés, les performances de l'AG sur certaines instances plus complexes (pas nécessairement par leur taille, comme cela a été vu) sont décevantes. Le mode de gestion des contraintes et les paramètres de fonctionnement de l'algorithme ayant été ajustés au mieux, une solution est cherchée en direction du codage. La partie suivante expose les deux types de modification testés et leurs résultats respectifs.

4.1 – Codage discrétisé croisé

4.1.1 – Définition du codage

De manière à améliorer les résultats obtenus, une première idée a consisté à modifier simplement la structure des chromosomes. Plus précisément, la modification apportée concerne la disposition des gènes au sein du chromosome.

Cette idée se base simplement sur des considérations relatives à la taille des différents types de gènes. Comme cela a déjà été mentionné, ceux codant une variable continue comportent, pour satisfaire la précision requise, 16 bits. Par contre, les gènes « entiers » ne comptent qu'un locus. En conséquence, en ajoutant une simple étape opératoire, on augmente la partie continue de 16 bits et la partie entière d'un seul. Au vu de la structure du chromosome, rappelée sur la figure 26, il est évident que l'action du croisement à un point de coupure a peu de chance de porter sur la partie entière : les gènes correspondants sont donc peu susceptibles d'être brassés efficacement lors de la recombinaison de deux chromosomes

parents (cf. annexe 1). En outre, cette tendance s'accentue si la taille de l'atelier augmente. L'exploration correcte de l'espace de recherche ne semble alors pas pouvoir être assurée.

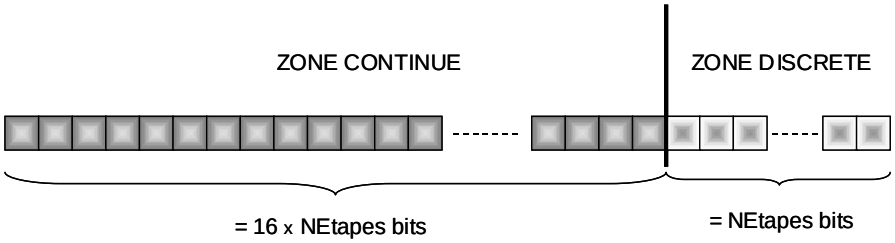


Figure 26 – Structure d'un chromosome avec le premier codage

De manière à éviter ce problème, la technique de codage a été conservée de manière identique, et seule la disposition des gènes au sein du chromosome a été modifiée. Ainsi, les variables continues sont toujours codées sur 16 bits au moyen de la boîte de poids et les variables discrètes sur un locus. Par contre, les gènes discrets ont été répartis sur l'ensemble de la longueur du chromosome, de manière à être eux aussi concernés par le croisement à un point de coupure. Ceci est réalisé en codant, côte à côte dans le chromosome, le gène continu du volume et le gène discret du nombre d'équipements de chaque étape opératoire. La structure du chromosome résultant est présentée sur la figure 27.

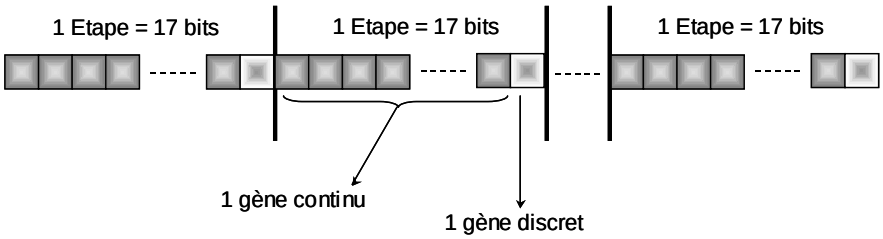


Figure 27 – Structure d'un chromosome avec le deuxième codage

Le croisement à un point de coupure agit alors logiquement tant sur les variables continues que sur les variables entières (voir annexe 1). Par opposition au premier codage « discrétisé séparé », le nouveau codage présenté est appelé par la suite « discrétisé croisé ». La figure 28 représente ce nouveau codage, pour une configuration particulière de l'atelier 1, qui avait été déjà utilisée dans la partie 2 (cf. figure 4) pour illustrer le codage « discrétisé séparé » employé au début de l'étude.

Les variables sont ainsi codées selon leur ordre d'apparition dans la recette permettant la synthèse des différents produits. Les calculs sont de nouveau effectués pour les 11 ateliers et les résultats comparés à ceux obtenus précédemment pour vérifier l'efficacité de l'amélioration apportée au codage.

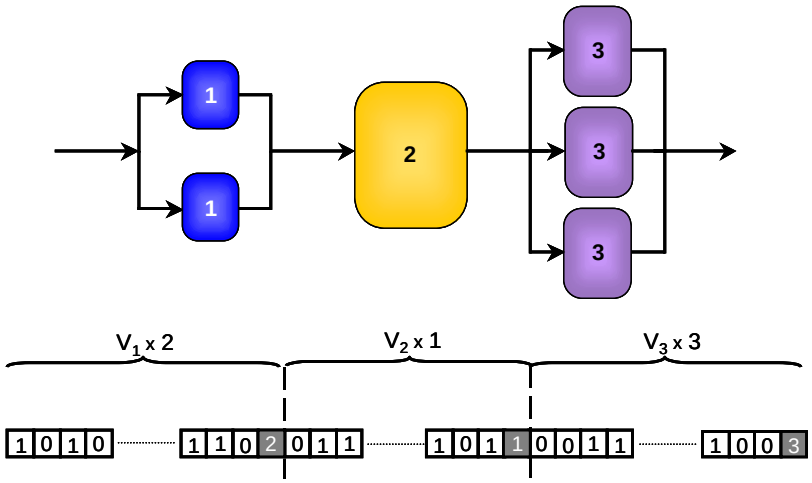


Figure 28 – Un exemple de codage « discrétisé croisé »

4.1.2 – Résultats

Tout d'abord, une précision s'impose : seuls les résultats concernant la qualité des solutions sont présentés ici. En effet, étant donné que la modification réalisée n'affecte pas la longueur du chromosome, le temps de calcul pour chaque exemple ne varie pas par rapport aux calculs effectués dans la partie précédente. Ils ne figurent donc pas explicitement dans ce qui suit. Le tableau 13 donne les solutions trouvées pour chaque exemple au moyen de l'AG au sein duquel a été implémenté le nouveau codage. Pour une meilleure lisibilité, les écarts à l'optimum sont représentés sur la figure 29 tant pour le codage discrétisé séparé que pour le codage discrétisé croisé, à des fins de comparaison.

	At. 1	At. 2	At. 3	At. 4	At. 5	At. 6
Résultat [$]	166092	122528	125207	369774	622682	768176

	At. 7	At. 8	At. 9	At. 10	At. 11
Résultat [$]	968921	1965039	2979603	3982220	5014656

Tableau 13 – Résultats du codage discrétise croisé

L'examen de la figure 29 démontre que les écarts entre la meilleure solution trouvée par chaque codage et l'optimum sont très proches les uns des autres. Il est donc clair que l'effet escompté, à savoir un brassage plus efficace des gènes codant les variables entières, n'a pas été atteint. Une analyse détaillée, exemple par exemple, n'est donc pas faite puisqu'elle serait identique à celle effectuée auparavant. Contrairement à ce qui était prévu, la modification n'a pas non plus aidé l'AG à s'extraire des optima locaux des exemples 2 et 4. Enfin, sur les

exemples de plus grosse taille, les jeux de variables discrets corrects, c'est-à-dire correspondant à ceux de l'optimum localisé par *SBB*, n'ont pas été déterminés.

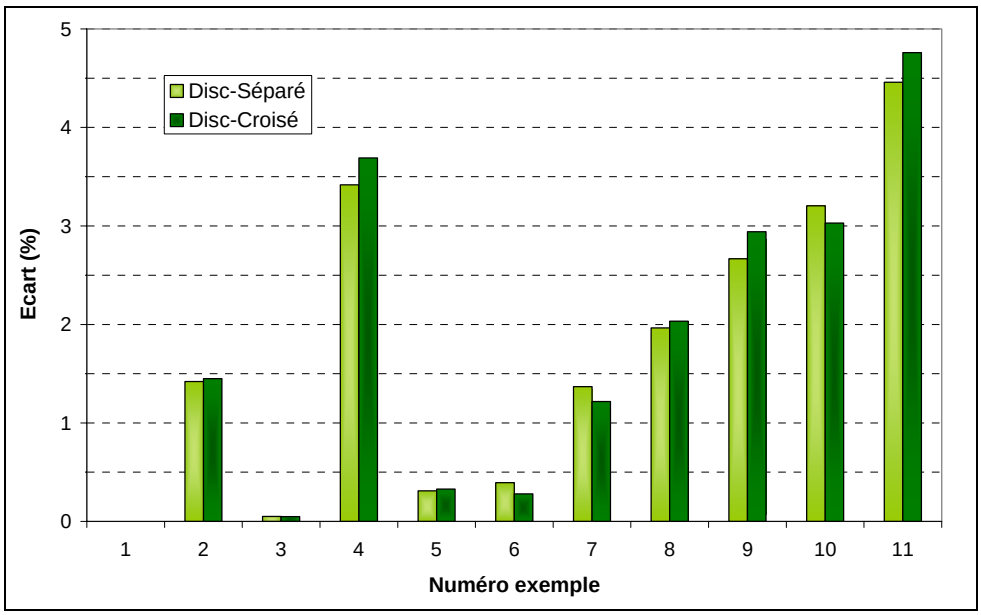


Figure 29 – Ecarts à l'optimum pour les deux codages discrétisés

En s'intéressant aux critères d'évaluation de la qualité de résultats prenant en compte l'aspect stochastique de l'AG, il semble que les conclusions ne sont cependant pas si négatives. Sur les premiers exemples, les dispersions à 2 % et 5 % tracées sur les figures 30 ressemblent également à celles obtenues avec le premier codage. Par contre, dès l'exemple 8, un changement est notable : les résultats de chaque exécution sont d'une manière générale plus proches de la meilleure solution trouvée qu'avec le codage discrétisé séparé.

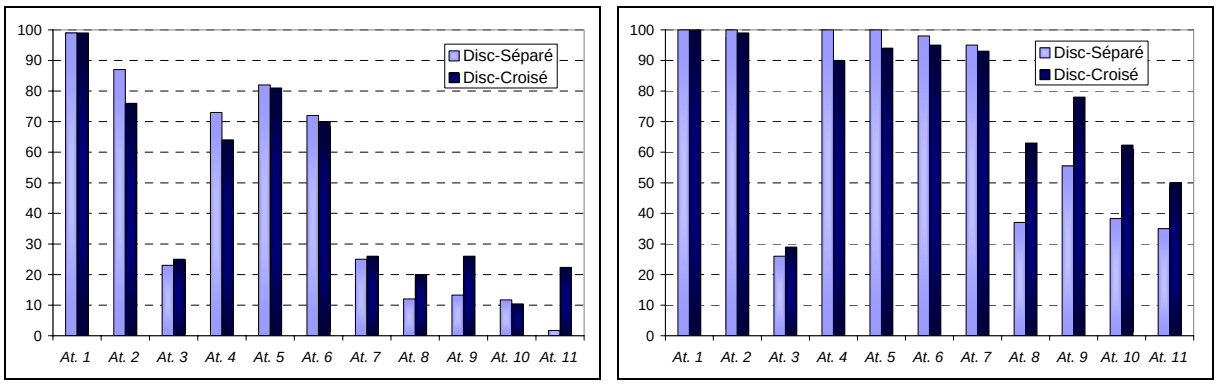


Figure 30 – Dispersions pour les deux types de codage
(a) à 2 % **(b) à 5 %**

Il découle donc de l'observation de la figure 30 que, si la modification du codage n'est pas parvenue à diminuer l'écart entre meilleure solution de l'AG et optimum, elle a permis d'augmenter la qualité moyenne des exécutions. Sur la dernière instance, par exemple, 50 %

des exécutions déterminent un résultat à moins de 5 % de la meilleure solution, et 22 % à moins de 2 % ; avec le codage discrétisé séparé, ces valeurs étaient égales à 35 % et 2 %. Il est aussi possible de remarquer que les résultats de l'exemple 9 sont de bonne qualité par rapport à ceux des autres exemples.

De la même manière que pour le codage discrétisé séparé, la baisse des performances de l'AG avec le codage discrétisé croisé sur les derniers exemples (écart à l'optimum et dispersions des résultats) peut être mise en relation avec le nombre de solutions faisables et le taux d'échec. Sur la figure 31, leur tracé met bien en relief une évolution opposée de ces deux critères. La correction du nombre de solutions faisables en fonction du taux d'échec souligne également que celui-ci garde une valeur presque constante si l'on ne considère que les exécutions donnant une solution.

Les deux lignes en pointillés continues rappellent les tendances du premier codage pour le nombre de solutions faisables et pour le taux d'échec (en excluant l'atelier 9 qui se situait au-dessus de la tendance). Ce dernier critère augmente nettement moins brusquement avec le deuxième codage. Vue l'invariance du rapport des deux indicateurs mentionnés, la croissance plus lente du taux d'échec explique la pente de descente plus douce pour la proportion de solutions faisables. Ainsi, le moins grand nombre d'échecs explique l'amélioration de la dispersion des résultats pour le codage discrétisé croisé.

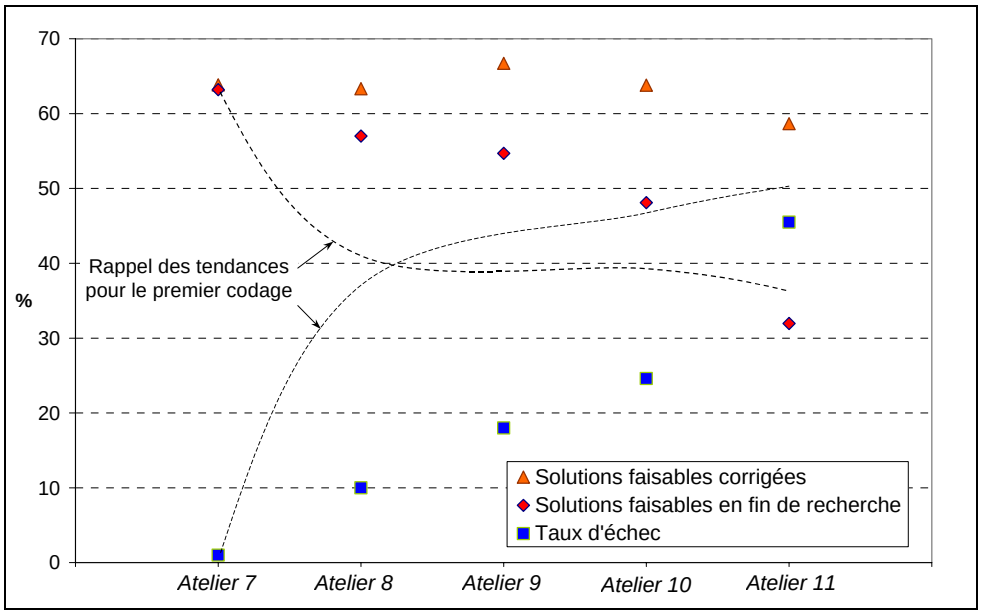


Figure 31 – Taux d'échec et solutions faisables en fin de recherche

Les résultats sont néanmoins loin d'être concluants, puisque la diminution espérée de la distance à l'optimum n'a pas été obtenue. Seule une légère amélioration de la répétabilité des résultats est à noter. Par ailleurs, la meilleure exploration de l'espace de recherche, grâce à

une recombinaison réorientée en direction des variables entières, n'a pas réussi à supprimer les échecs : même si le nombre de ces derniers est moins élevé qu'avec le codage précédent, il reste important pour les derniers exemples (45 % pour l'instance 11). Une autre modification du codage, plus radicale cette fois-ci, est donc proposée dans la partie suivante.

4.2 – Codage mixte réel-entier

4.2.1 – Représentation des variables réelles

Les résultats obtenus avec le codage discrétisé, même améliorés en mélangeant variables continues et entières, ne sont donc pas entièrement satisfaisants. Sur les exemples de grande taille, la garantie d'obtenir une solution faisable n'est pas assurée par une exécution de 4,6 [h] tandis que le solveur *SBB* donne une solution optimale. Par ailleurs, l'algorithme génétique continue à être perturbé par des solutions sous-optimales qui, même pour des problèmes de petite taille, faussent la convergence vers l'optimum.

L'explication la plus naturelle de ce comportement réside dans la nature mixte du problème et la discrétisation adoptée pour le codage des variables continues. Ainsi, alors que l'ajout d'une variable continue ne semble pas perturber le module de Programmation Mathématique, il influe au contraire fortement sur l'AG. En effet, chaque variable réelle signifie 16 bits, c'est-à-dire un facteur 2^{16} à intégrer dans la « pseudo-combinatoire », adaptée au mode de représentation des variables dans l'AG. Ainsi, l'espace de recherche généré par les variables continues est considérable et la taille du chromosome pénalise lourdement l'AG, tant en termes de qualité de la solution que de temps de calcul.

La réponse à cet écueil passe par une diminution de la taille occupée par les variables continues dans le chromosome. Cette remarque a logiquement dirigé vers un fonctionnement employant des loci à valeurs réelles pour représenter les variables continues : chaque variable réelle occupera donc le même espace dans le chromosome qu'une variable entière, i.e. un locus. Cependant, même si la taille du chromosome est énormément réduite, la taille de l'espace de recherche reste importante. Il ne faut en effet pas perdre de vue qu'une discrétisation est inévitablement imposée aux variables continues, liée à la précision du langage de programmation utilisé, le Fortran 90. Une clé de l'efficacité de ce nouveau de type de codage reposera donc grandement sur les opérateurs génétiques qui lui sont associés. Etant donné qu'il mélange au sein du chromosome loci réels et loci entiers, ce type de codage sera appelé par la suite « codage mixte ». Il est représenté sur la figure 32 pour une configuration de l'exemple 1, déjà employée pour illustrer les méthodes de codage précédentes.

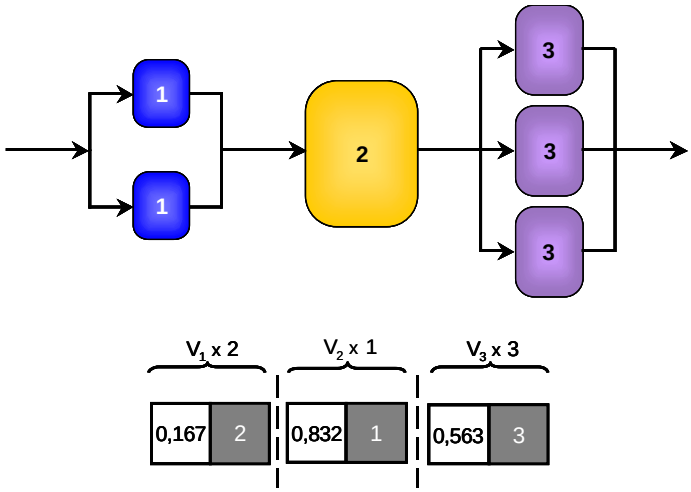


Figure 32 – Un exemple de codage « mixte »

4.2.2 – Chromosomes à valeurs réelles

L'attention de la communauté des algorithmes évolutifs a jusqu'à présent été tournée pour l'essentiel vers les codages de type binaire ou similaires, nécessitant de toute manière une discrétisation des variables continues. Mais, l'utilisation de plus en plus fréquente de loci à valeur réelle a prouvé la supériorité de ces derniers dans des applications variées. Par exemple, un codage binaire classique et un codage réel sont comparés dans [DEB00] : les calculs effectués prouvent, dans le cadre étudié, la supériorité du second sur le premier. Les travaux de Michalewicz, qui font référence, sont également basés sur des chromosomes réels [MIC94b].

4.2.2.1 – Croisement

Compte tenu de l'immédiateté de la réalisation du codage, établissant l'équivalence entre locus, gène et variable, l'enjeu des AG à codage réel repose sur l'efficacité des opérateurs génétiques de croisement et mutation. Concernant le premier, les procédures habituellement utilisées pour les codages basés sur une discrétisation des variables continues (un point de coupure, *k* points de coupure, croisement uniforme…) sont évidemment applicables [ELL02]. Elles font cependant perdre l'effet de diversification sur le ou les gènes sur lequel/lesquels était coupé le chromosome. Le plus souvent, les techniques de croisement employées en codage réel sont réalisées gène par gène. Elles peuvent reposer sur des combinaisons de leur valeur respective.

Ainsi, on peut citer les croisements arithmétique ou géométrique [MIC96b] représentés respectivement par les équations (9) et (10) :

$$\begin{cases} y_k^{(1)} = \alpha.x_k^{(1)} + (1-\alpha).x_k^{(2)} \\ y_k^{(2)} = (1-\alpha).x_k^{(1)} + \alpha.x_k^{(2)} \end{cases} \quad (9)$$

$$\begin{cases} y_k^{(1)} = [x_k^{(1)}]^{\alpha}.[x_k^{(2)}]^{(1-\alpha)} \\ y_k^{(2)} = [x_k^{(1)}]^{(1-\alpha)}.[x_k^{(2)}]^{\alpha} \end{cases} \quad (10)$$

où $x_k^{(1)}$ et $x_k^{(2)}$ représentent les k^{ème} gènes des deux parents et $y_k^{(1)}$ et $y_k^{(2)}$ ceux des enfant créés, α étant un paramètre fixe compris entre 0 et 1. Ces techniques sont généralisables à m parents et m enfants en prenant m coefficients α_j tels que $\alpha_1 + \ldots + \alpha_m = 1$.

Un autre type de technique basé sur une combinaison linéaire entre les deux parents est mise en œuvre dans le *α-Blend Crossover* (BLX-α) citée dans [DEB95]. Pour chaque gène k des parents $x^{(1)}$ et $x^{(2)}$, un seul enfant y est généré dont le k^{ème} gène est tiré aléatoirement dans l'intervalle $[x^{(1)}-\alpha(x^{(2)}-x^{(1)}),\ x^{(2)}+\alpha(x^{(2)}-x^{(1)})]$, avec nécessairement $x^{(1)} < x^{(2)}$. Si $\alpha = 0$, l'individu enfant est tiré aléatoirement dans l'intervalle $[x^{(1)}, x^{(2)}]$. Dans le cas contraire, l'intervalle dans lequel est choisi y n'est pas restreint par cet aspect que Deb et Agrawal [DEB95] qualifient de *contractant*, soulignant que des tests font prévaloir la valeur $\alpha = ½$.

D'autres méthodes s'aident de l'évaluation de la fonction objectif des parents pour déterminer un enfant adapté au mieux. La technique dite *Direction Based Crossover* (DBX) proposée dans [ARU05] crée un enfant unique y à partir de deux individus $x^{(1)}$ et $x^{(2)}$, sachant que $x^{(2)}$ est meilleur que $x^{(1)}$ (c'est-à-dire $f(x^{(2)}) \leq f(x^{(1)})$ dans le cas d'une minimisation) et r est un nombre aléatoire compris entre 0 et 1 :

$$y = r.(x^{(2)}-x^{(1)}) + x^{(2)} \quad (11)$$

D'une façon similaire, la technique de croisement-simplexe (SPX) mentionnée dans [MIC96b] et [RAG05] sélectionne $m > 2$ parents aléatoirement et calcule l'isobarycentre w des m–1 meilleurs. Un unique enfant est construit par réflexion du parent le moins bon (celui qui est exclu du calcul de l'isobarycentre) par rapport à w.

Finalement, l'opérateur de croisement retenu ici pour un AG à valeurs réelles est celui proposé dans [DEB95]. La procédure SBX (*Simulated Binary Crossover*) consiste, comme son nom l'indique, à reproduire l'effet d'un croisement à un point de coupure mis en œuvre

pour un codage binaire classique. De la même manière que pour un croisement arithmétique, deux enfants sont créés par combinaison linéaire des deux parents.

La particularité est que les coefficients de pondération sont définis par une distribution de probabilité polynomiale d'ordre *n*, calculée de manière à reproduire deux caractéristiques principales du croisement à un point de coupure pour un codage binaire :

- La moyenne de la variable *k* calculée pour deux parents $0,5.(x_k^{(1)}+x_k^{(2)})$ est égale à celle calculée pour les deux enfants résultants $0,5.(y_k^{(1)}+y_k^{(2)})$.

- Si la procédure de croisement est de nouveau appliquée aux enfants avec un point de coupure identique à celui qui a été utilisé pour les créer, on retrouve les deux parents originaux.

A partir de la distribution de probabilité résultante et de l'introduction d'un aspect stochastique, les coefficients de pondération pour la combinaison linéaire des deux parents sont générés. L'ordre de la distribution polynomiale retenue ici est $n = 1$. La procédure complète de calcul des coefficients est décrite en annexe 1. Elle est appliquée avec une probabilité supplémentaire p_x, de manière à ne perturber qu'une partie des chromosomes.

Les gènes codant des variables entières sont soumis au même type de procédure. Comme les gènes résultants auront *a priori* une valeur réelle, c'est l'entier immédiatement inférieur à cette valeur qui est pris. Il est intéressant de noter que, cette méthode de croisement étant appliquée gène par gène, la position des variables entières dans le chromosome importe peu.

4.2.2.2 – Mutation

Le plus usité des opérateurs de mutation est, d'après [MIC96b], une procédure dite *Gaussienne*, qui modifie toutes les composantes du vecteur solution en y ajoutant un bruit aléatoire, suivant une loi normale centrée $N(0,\sigma)$. Les démarches diffèrent ensuite selon le choix de l'écart-type. Une mutation uniforme est aussi possible en modifiant un seul gène de l'individu, en lui affectant une valeur aléatoire (distribution de probabilité uniforme) à l'intérieur de son intervalle de variation.

Une autre option consiste en une mutation non-uniforme, souvent caractérisée par une plus forte probabilité de créer un individu proche de l'individu initial. Par ailleurs, cette probabilité peut augmenter avec le nombre de générations : ainsi, l'opérateur permet une

exploration uniforme de l'espace au début de la recherche, puis elle devient plus locale et centrée sur l'individu initial lors des générations ultérieures.

Des formulations différentes sont proposées dans [RAG05] et [MIC96b], seule celle correspondant à la deuxième référence est présentée ici :

$$\begin{cases} y_k = x_k + \Delta(t, x_k^U - x_k) & \text{si un nombre entier tiré aléatoirement est égal 1,} \\ y_k = x_k + \Delta(t, x_k - x_k^L) & \text{si un nombre entier tiré aléatoirement est égal 0,} \end{cases} \quad (12)$$

où x_k^L et x_k^U sont respectivement les bornes inférieure et supérieure du gène x_k, et t le nombre de générations. La fonction $\Delta(t, a)$ retourne une valeur dans l'intervalle [0, a] avec une probabilité que cette valeur soit proche de 0 qui augmente avec t. Une stratégie possible est :

$$\Delta(t, a) = a.r(1 - t/T)^b \quad (13)$$

r est un nombre tiré aléatoirement, T le nombre maximal de générations et b un paramètre déterminant le degré de non-uniformité.

Finalement, le choix de la procédure de mutation pour ces travaux s'est d'abord orientée vers une méthode proposée par Deb et Goyal [DEB96], complémentaire de la procédure de croisement adoptée ici [DEB95]. Elle est basée sur une perturbation du gène initial, suivant une distribution de probabilité polynomiale d'ordre n et dont la moyenne est égale à la valeur du gène initial. Mais des études de sensibilité réalisées pour ajuster l'ordre n de la distribution ont mis en évidence que la valeur la plus adaptée était zéro. La technique adoptée correspond donc à une distribution uniforme sur l'ensemble de l'intervalle de définition de chaque variable. L'intervalle de validité de la distribution est [-1,1], mais peut-être bien sûr étendu à $[x_k^L, x_k^U]$. Au contraire de la procédure de croisement, cet opérateur est simplement appliqué aux gènes continus du chromosome. Pour les gènes entiers, la méthode reste identique à celle utilisée précédemment, i.e. soustraction d'une unité à la valeur du locus quand cela est possible. La procédure de mutation relative aux gènes continus est explicitée avec plus de détails dans l'annexe 1 de ce chapitre.

4.2.3 – Résultats

Contrairement à la précédente, la modification du codage présentée ici a une incidence sur le temps de calcul. Le passage à des loci à valeur réelle implique, comme c'est l'effet recherché, une réduction de la taille du chromosome : la modification entraîne une division

par 16 de la longueur des gènes codant les variables continues et implique la manipulation de tableaux de taille réduite au sein de l'algorithme. La procédure de décodage se résume par ailleurs à sa plus simple expression, puisque on a « *gène = locus = variable* ».

En conséquence, le gain en temps de calcul est très significatif, comme le prouve le tableau 14, récapitulant les résultats de l'AG dans lequel est implémenté le codage mixte. L'évolution du temps de calcul pour le traitement de chaque atelier, comparée à celle obtenue avec les deux codages précédents, est représentée dans la figure 33 sous forme logarithmique.

	At. 1	At. 2	At. 3	At. 4	At. 5	At. 6
Meilleure solution [$]	166089	120799	125147	356635	621724	767182
Temps CPU	< 1 [s]	< 1 [s]	< 1 [s]	1 [s]	11 [s]	58 [s]

	At. 7	At. 8	At. 9	At. 10	At. 11
Meilleure solution [$]	958778	1958190	2966731	3961817	4988541
Temps CPU	8 [s]	19 [s]	5,6 [min]	14 [min]	1,23 [h]

Tableau 14 – Résultats du codage mixte

L'amélioration du temps de calcul grâce au codage mixte y est mise en évidence, notamment pour que les exemple complexes. Sur le graphique, une frontière est tracée pour permettre de visualiser quel mode de gestion des contraintes est employé pour la résolution de chaque exemple. Cette délimitation explique la baisse du temps de calcul entre les ateliers 6 et 7.

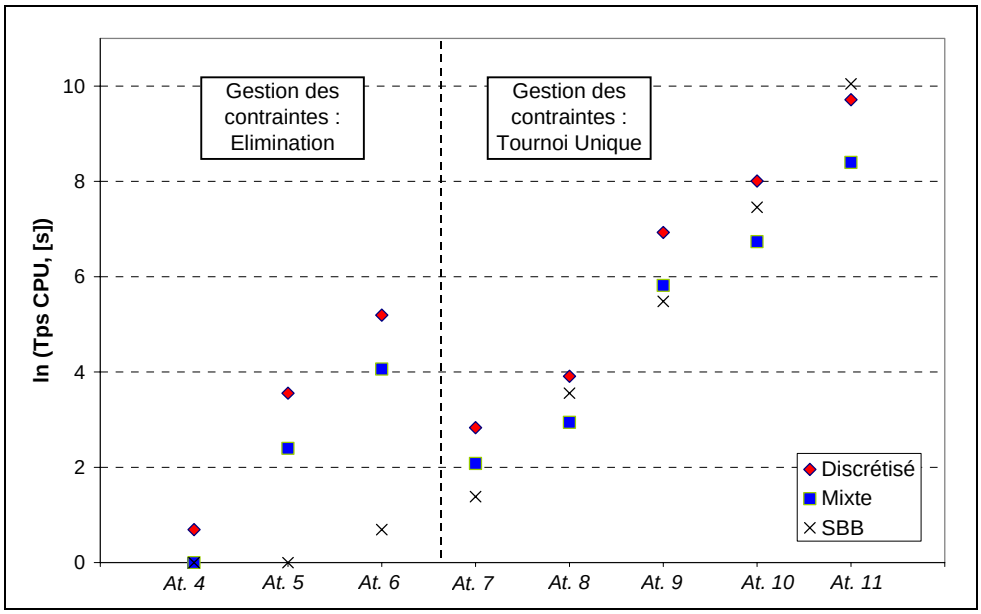


Figure 33 – Temps de calcul

Par ailleurs, il est possible de vérifier que l'écart entre les deux courbes augmente avec la taille des exemples : en effet, la réduction de la taille du chromosome, due à la nouvelle forme de codage, est d'autant plus importante que le nombre de variables continues est élevé. Le temps de calcul de l'AG avec ce dernier codage est aussi comparé, sur la figure 33, avec le temps nécessaire à la résolution pour le module *SBB* : à partir du moment où le tournoi unique est adopté pour gérer les contraintes, l'AG est plus rapide que la méthode de Programmation Mathématique (mis à part pour l'exemple 9, où la différence entre les deux est très réduite).

Les solutions trouvées sont disponibles dans le tableau 14. L'écart à l'optimum est de nouveau tracé sous forme d'histogramme sur la figure 34, en parallèle avec la meilleure des solutions trouvées avec les deux codages précédents (qui n'est pas le même pour chaque exemple). Sur l'exemple 1, pour lequel les résultats ont toujours été excellents, la meilleure solution trouvée n'est pas significative de l'effet du codage mixte.

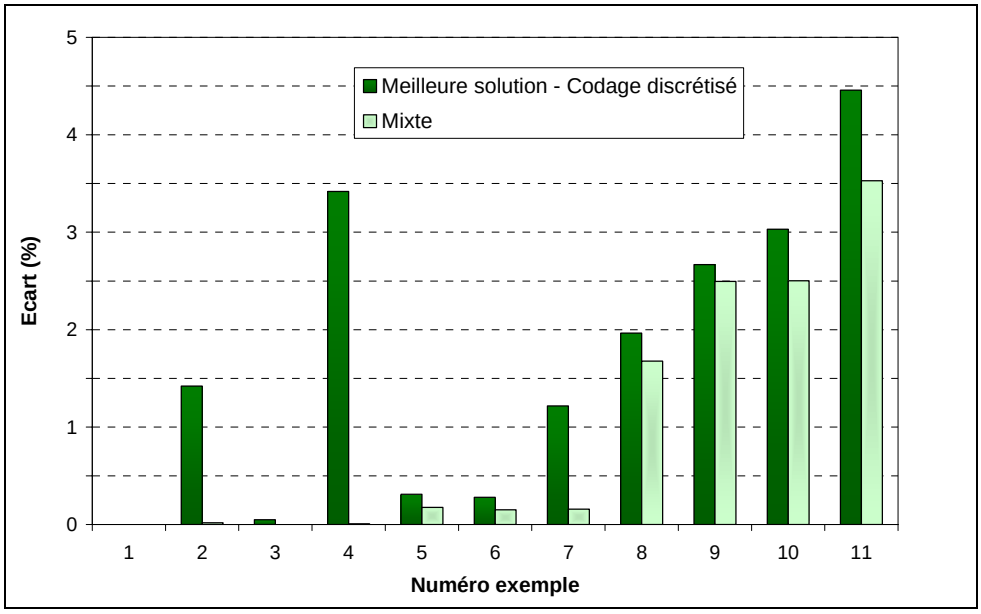


Figure 34 – Ecarts à l'optimum

Les dispersions à 2 % et à 5 %, représentées sur la figures 35 (avec les résultats du codage discrétisé croisé seulement), ont peu d'intérêt, elles aussi. Pour les exemples 5 et 6, identifiés comme étant de moindre complexité pour l'AG, les résultats sont améliorés de même avec une faible amplitude, tant en termes d'écart à la solution que de dispersions. Il faut cependant noter que, si l'écart à l'optimum diminue, cela signifie que l'ensemble des exécutions, évalué par les dispersions à 2 % et à 5 %, est d'autant plus proche de la solution. Ainsi, à dispersions fixées, un écart plus faible à l'optimum de la meilleure solution trouvée signifie que les exécutions sont plus « bas ».

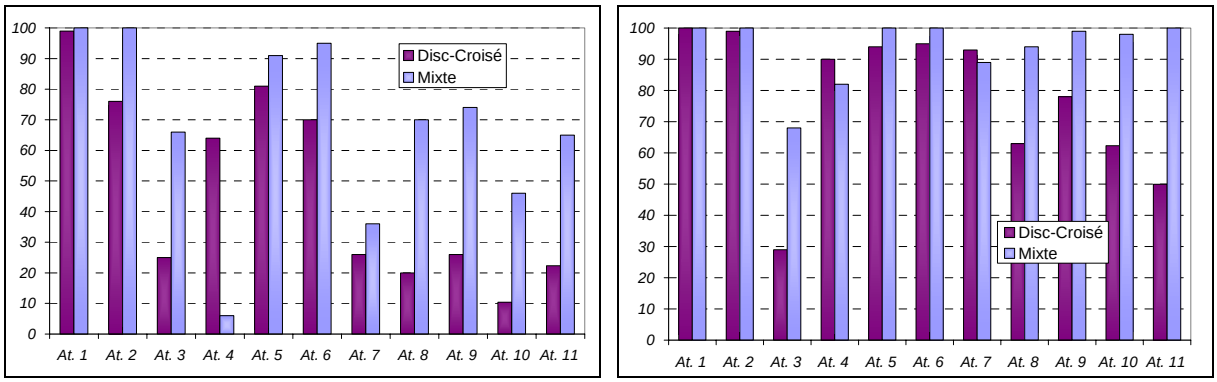


Figure 35 – Dispersions pour les codages mixte et discrétisé croisé
(a) à 2 % **(b) à 5 %**

Ce rappel sur la différence entre qualité relative et absolue des résultats est illustrée par un exemple imaginaire présenté sur la figure 36 : dans le cas 1, la dispersion à 2 % est moins bonne que dans le cas 2, mais comme la meilleure solution de l'AG est inférieure (plus « basse »), la qualité absolue des solutions de chaque exécution est globalement meilleure. Par conséquent, l'amélioration des dispersions pour les exemples 5 et 6 est accentuée par la diminution de l'écart à l'optimum.

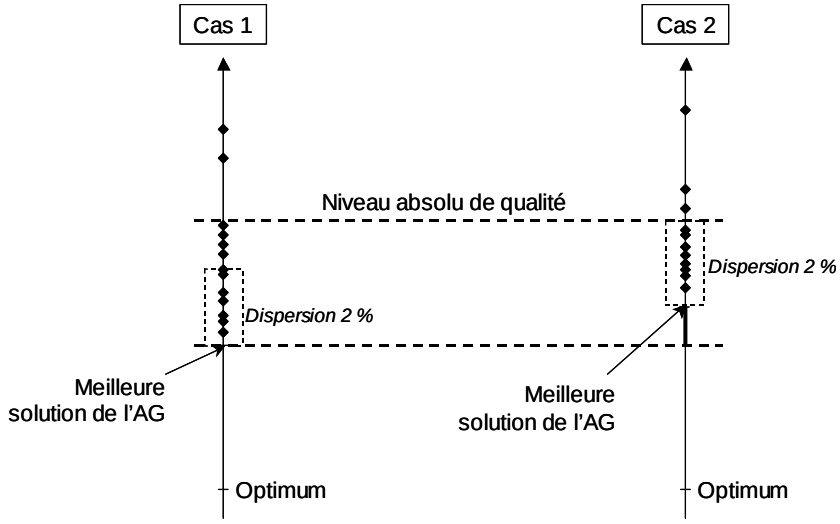


Figure 36 – Illustration des qualités relative et absolue des résultats

En revanche, la nette amélioration du résultat pour l'exemple 2 met en évidence la capacité de l'AG à s'extraire maintenant de la solution sous-optimale sur laquelle il restait piégé auparavant. Cette remarque est confirmée en vérifiant le jeu de variables discrètes, identique à celui de l'optimum de *SBB*. Par ailleurs, les dispersions à 2 % et à 5 % montrent que toutes les exécutions parviennent à localiser cet optimum sans être gênées par la présence de l'optimum local. Les résultats sur l'exemple 4 sont similaires à ceux de l'exemple 2 : l'optimum local est évité et l'AG détermine une solution à moins de 0,01 % de l'optimum. Par contre, comme cela avait déjà été observé pour l'exemple 3 avec les codages discrétisés,

les dispersions à 2% et à 5 % démontrent que la détermination de l'optimum reste une tâche ardue pour l'AG. En fait, 6 % des exécutions seulement parviennent à déterminer un jeu de variables entières correct. Même si la dispersion à 5 % est convenable, elle ne représente essentiellement que des résultats proches de l'optimum local évoqué plus haut.

Pour l'exemple 3, l'algorithme avec les codages discrétisés réussissait déjà à éviter l'optimum local, mais une proportion importante des exécutions y restait néanmoins piégée. Avec le codage mixte, l'optimum des méthodes de programmation mathématique est détecté sans difficulté excessive, puisque les dispersions à 2 % et à 5 % atteignent toutes deux presque 70 %. L'exemple 7 posait le même type de problème à l'AG avec, en outre, un écart à l'optimum plus conséquent (1,22 %). La figure 34 met en évidence l'amélioration concernant la distance à la solution optimale, mais les dispersions tracées sur la figure 35 montrent une timide augmentation pour celle à 2 % et même une certaine régression pour celle à 5 %. Le comportement des dispersions doit cependant être nuancé, comme cela a été dit plus haut, par l'amélioration de la meilleure solution trouvée : dans l'absolu, l'AG est globalement performant sur cet exemple, même si la dispersion à 2 % est un peu faible.

Finalement, les résultats sont mitigés pour les exemples 8 à 11. En effet, si la solution est toujours légèrement meilleure qu'elle ne l'était avec les deux autres codages, elle reste assez éloignée de l'optimum dans tous les cas. L'obtention espérée du jeu correct de variables entières n'a pas abouti. Par contre, les dispersions à 2 % et à 5 %, sont, elles, de très bonne qualité vu la taille des instances abordées.

Il est possible de mettre en parallèle ces résultats avec les proportions de solutions faisables en fin de recherche (figure 37) : ce dernier critère demeure faible sur les exemples les plus complexes. Il l'est même plus qu'avec le codage discrétisé croisé. En revanche, l'un des principaux progrès amenés par le codage mixte est l'absence totale d'échecs : toutes les exécutions finissent par donner une solution faisable. Le temps de calcul est alors valorisable sans aucune réserve. Par exemple, il n'est ainsi plus nécessaire de conditionner par un taux d'échec l'affirmation suivante : pour le dernier atelier, une exécution de l'AG durant 1,23 [h] garantit l'obtention d'une solution à moins de 5 % de la meilleure solution trouvée avec une probabilité égale à 1. Comme la meilleure solution trouvée est elle-même distante de 3,53 % de l'optimum, l'AG donne la certitude d'obtenir une solution d'une qualité raisonnable en un temps bien inférieur à *SBB* (qui effectue une exécution en 6,4 [h] pour la onzième instance).

Finalement, les résultats de l'AG avec le codage mixte sont satisfaisants dans la mesure où cette dernière option a permis de dépasser certains obstacles qui pénalisaient les performances de cette méthode d'optimisation. Tout d'abord, les problèmes liés à la présence

d'optima locaux semblent avoir été réglés pour des instances de taille raisonnable. Jusqu'à l'exemple 7, l'efficacité de l'AG est excellente pour la qualité de la solution et pour le temps de calcul. Sur les problèmes de plus grande taille, le grand nombre d'échecs impliquait une qualité contestable du temps de calcul puisque ce dernier n'était pas synonyme de l'obtention fiable d'un résultat : ce premier écueil a été dépassé grâce au codage mixte

Par contre, un dernier point négatif reste non résolu au moyen de l'approche par codage mixte, à savoir la détermination d'une solution très proche de l'optimum. Il est vrai que, jusqu'à l'exemple 10, l'écart entre optimum et meilleure solution de l'AG ne dépasse jamais 2,5 %. Cet écart n'est donc réellement important que pour le dernier atelier, qui se situe déjà à la limite d'une taille industriellement réaliste (105 étapes opératoires). La défaillance de l'AG sur ce point est donc à relativiser.

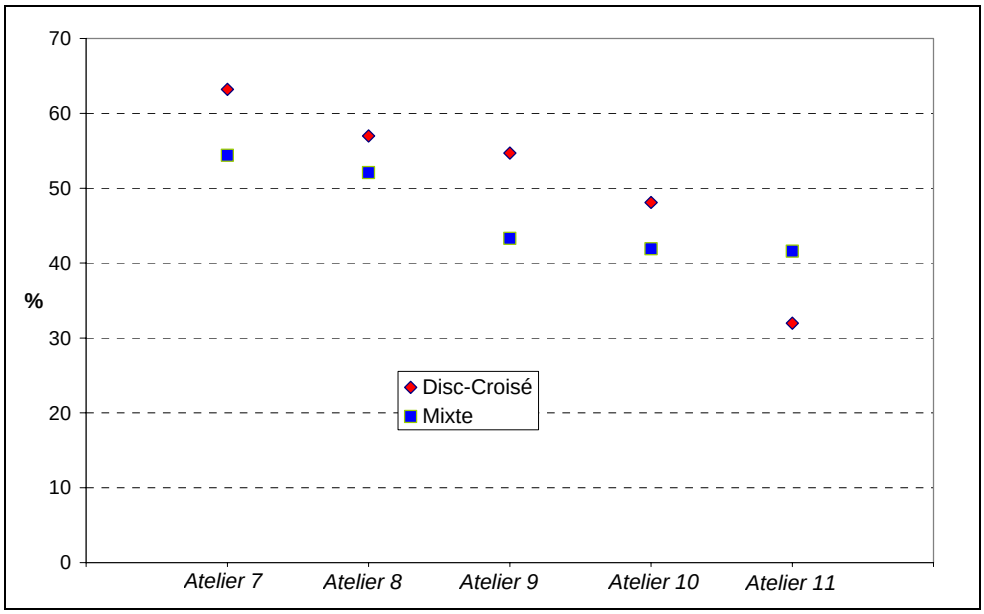


Figure 37 – Proportion de solutions faisables en fin de recherche

Mais le point le plus pénalisant réside dans le fait que l'AG ne trouve pas le bon jeu de variables entières. Les calculs précédents semblaient indiquer que c'était le nombre réduit de solutions faisables rencontrées au cours de la recherche, dû à la proportion importante d'échecs, qui en était la cause ; mais avec le codage mixte, les échecs ne peuvent plus expliquer la faible proportion de solutions faisables puisque aucun n'est constaté. Cette tendance semble indiquer qu'un effort supplémentaire pour créer davantage de solutions faisables doit être effectué, de manière à conférer à l'AG à codage mixte une efficacité sans nuance. Une étude sur des exemples de grande taille, dédiée à une amélioration de la technique du tournoi unique avec l'utilisation du codage mixte, serait utile en ce sens.

Conclusion

Ce chapitre a exposé les calculs effectués sur le jeu de onze exemples-test proposé pour évaluer l'efficacité des trois méthodes d'optimisation étudiées dans ces travaux. En premier lieu, les méthodes déterministes ont fait preuve d'une capacité inégale à traiter entièrement le jeu de problèmes. *DICOPT++* s'est en effet relativement bien comporté sur les cinq premières instances, trouvant des solutions optimales en des temps raisonnables. Mais il est mis en défaut tout d'abord par un optimum local duquel il ne peut s'extraire dans l'atelier 2, puis par la taille des problèmes traités : dès l'exemple 6, une exécution limitée à 24 heures ne lui permet de fournir aucun résultat. Cet échec est imputable à la répétition de trop nombreux sous-problèmes NLP infaisables, paralysant le déroulement de l'algorithme des Approximations Externes. En revanche, le module de résolution *SBB* a donné de bien meilleurs résultats : il parvient à déterminer une solution optimale pour les onze problèmes sans difficulté majeure, avec, de manière logique, un temps de calcul exponentiellement croissant. Dans le cadre de la formulation retenue, l'algorithme de Branch & Bound, modifié pour permettre la résolution de sous-problèmes continus non-linéaires, a donc prouvé sa supériorité sur l'algorithme OA/ER/AP.

Dans un deuxième temps, la présentation des calculs effectués avec l'algorithme génétique a nécessité la définition préalable d'un mode opératoire adapté à son caractère stochastique. Les procédures de fonctionnement initialement choisies, i.e. un codage discrétisé séparé et l'élimination des individus infaisables, ont été ensuite exposées. Les premiers calculs ont révélé un comportement globalement correct, mais pénalisé par un temps de résolution augmentant de manière à compromettre la faisabilité des calculs dès l'exemple 7. La cause de cette augmentation considérable ayant été identifiée, l'examen de plusieurs techniques de gestion des contraintes a été effectué sur une instance de taille raisonnable : pénalisation statique, relaxation des bornes supérieures sur les variables discrètes, tournois basés sur des notions de dominance et enfin stratégie multiobjectif. L'unanimité des critères (répétabilité des résultats, nombre de solutions faisables trouvées) a amené à choisir une technique de *tournoi unique*, i.e. impliquant une seule procédure de tournoi par étape de sélection, pour effectuer les calculs sur les cinq derniers exemples.

Le bilan des résultats sur l'ensemble des problèmes laisse cependant entrevoir les limitations de l'AG, liées principalement à l'existence d'optima locaux et à un important nombre d'échecs sur les exemples de grande taille, empêchant l'AG de trouver une solution proche de l'optimum de *SBB*. Une des causes possibles a été attribuée au codage. Par conséquent, deux nouveaux types de codages ont été proposés. Le premier, basé sur un simple réagencement des gènes à l'intérieur du chromosome, ne parvient pas à dépasser les deux

obstacles mentionnés. Les performances de l'AG sont néanmoins améliorées pour des exemples de taille intermédiaire et supérieure. Par contre, le codage mixte (emploi de loci à valeurs réelles pour coder les variables continues) amène une réelle amélioration puisque l'algorithme arrive à s'extraire des optima locaux et plus aucun échec n'est constaté. Mais la distance à l'optimum reste problématique pour le dernier exemple, la meilleure solution trouvée par l'AG restant assez éloignée. Ainsi, le mode de fonctionnement idéal pour l'algorithme génétique est un codage mixte réel-entier (ce qui paraît finalement logique au vu de la nature du problème…) et une gestion des contraintes par élimination lorsque le temps de calcul le permet et par tournoi unique sinon.

Un des objectifs de ces travaux, à savoir la détermination des paramètre optimaux de fonctionnement de l'algorithme génétique pour la résolution de problèmes contraints en variables mixtes, a été atteint. Ces considérations ne doivent néanmoins pas faire perdre de vue l'autre objectif, i.e. la justification du choix d'une méthode d'optimisation en fonction de l'instance traitée, formulée selon le modèle choisi pour cette étude. Les performances des trois méthodes sont donc comparées dans le tableau 15.

Exemples	*DICOPT++*		*SBB*		Algorithme Génétique		
	Tps	Qlté	Tps	Qlté	Tps	Qlté	Rép
1	+	+	+	+	+	+	+
2	+	–	+	+	+	+	+
3	+	+	+	+	+	+	0
4	+	+	+	+	+	+	–
5	–	+	+	+	0	+	0
6			+	+	–	+	0
7			+	+	0	+	–
8			0	+	+	0	0
9			+	+	0	0	0
10			0	+	+	0	0
11			0	+	+	–	0

Tableau 15 – Tableau récapitulatif

Comme cela a été fait jusqu'à présent, les critère d'évaluation de chaque méthode sont le temps de calcul (« Tps ») et la qualité de la solution trouvée (« Qlté »), auxquels s'ajoute la répétabilité des résultats pour l'AG (« Rép »). La valeur « + » signale la meilleure performance possible (meilleur temps de calcul, solution optimale ou dispersions proches de 100 %) tandis la valeur « – » indique une contre-performance (temps de calcul médiocre, optimum local, répétabilité très faible). L'indicateur « 0 » équivaut à une performance convenable sans être exceptionnelle. Ce tableau prouve finalement la supériorité du Branch & Bound sur les autres méthodes. Même si l'AG est parfois plus performant en temps de calcul,

le temps maximal de 6,4 heures nécessaire à *SBB* ne semble pas excessif pour obtenir, quelle que soit l'instance traitée, une solution optimale du problème.

Deux réserves sont cependant à émettre sur ces conclusions, sans pour autant les contredire. Tout d'abord, dans le tableau 15 ne figure aucune mention à l'effort nécessaire à la mise en œuvre des méthodes d'optimisation testées. L'AG est simple à implémenter, relativement générique, son fonctionnement est facile à comprendre et adaptable *a priori* à n'importe quel type de problème (même si ajouter de la connaissance à la procédure d'optimisation facilite bien entendu sa tâche). En revanche, les méthodes de Programmation Mathématique nécessitent un effort de formulation important, basé sur la considération des propriétés mathématiques des fonctions mises en jeu. Ce temps de mise en oeuvre, d'autant plus élevé pour un novice, est sans doute une contrepartie à l'efficacité *a posteriori* des méthodes déterministes.

Par ailleurs, une des préoccupations de cette étude est le traitement de problèmes réalistes : nous sommes toutefois conscients que la plus grande taille d'atelier a été essentiellement étudiée pour pousser à l'extrême les performances numériques des méthodes. Ainsi, dans la continuité de cette logique, il semblerait normal de prendre en compte une contrainte classique du monde industriel, ayant une conséquence immédiate dans le modèle utilisé : les tailles d'équipements ne varient généralement pas de manière continue entre leurs bornes. Le plus souvent, des gammes de volume sont proposées par les fournisseurs. Ceci implique une discrétisation des variables continues selon des intervalles plus ou moins larges. Dans ce cas de figure, il n'est pas possible d'affirmer que les conclusions énoncées plus haut seront encore vérifiées. Le chapitre suivant traite donc ce nouveau problème.

CHAPITRE 3
ANNEXE 1

OPERATEURS GENETIQUES

L'opérateur de croisement doit répondre à deux attentes. Tout d'abord, il doit permettre de conserver la qualité du patrimoine génétique de la population, en respectant l'hypothèse que deux individus forts donnent, par leur recombinaison, deux individus forts également. Par ailleurs, de par son aspect aléatoire, la procédure de croisement a comme objectif de diversifier la population et, ainsi, d'améliorer l'exploration de l'espace de recherche. La procédure de mutation consiste à modifier de manière aléatoire la valeur de certains gènes du chromosome. Au contraire du croisement, elle est toujours mise en oeuvre localement, sur un bit ou locus – de manière probabiliste bien entendu, et non sur la globalité du chromosome.

Cette annexe présente plus en détails les procédures mises en œuvre au sein des opérateurs génétiques utilisés. Comme cela a été précisé dans le chapitre 3, ces derniers suivent nécessairement la logique du codage employé. Une première partie est donc consacrée aux opérateurs relatifs aux codages basés sur une discrétisation des variables continues (i.e. codages discrétisés « séparé » et « croisé », selon les dénominations employées précédemment), tandis qu'une seconde s'emploie à décrire ceux concernant le codage pour lequel les variables continues sont représentées par des loci à valeur réelle. Dans chacun des cas, les procédures de croisement et de mutation sont explicitées.

1 – Codages discrétisés

Dans un premier temps, les variables continues sont donc discrétisées puis codées sur des bits au moyen de la méthode de la boîte de poids déjà présentée. Ces bits cohabitent, au sein du chromosome, avec des loci à valeur entière, représentant la valeur des variables discrètes.

1.1 – Procédure de croisement

Il existe un grand nombre d'opérateurs de croisement pour des codages sur bits, qui peuvent tout à fait être combinés avec des loci entiers. Les deux opérateurs les plus classiquement adoptés pour ce type de codage sont le croisement à k-points et le croisement uniforme. Tout d'abord, deux individus parents sont choisis aléatoirement dans la population courante. Puis, dans le premier cas, k entiers inférieurs à la taille d'un chromosome et supérieurs à 1 sont générés, également de manière aléatoire. Les deux chromosomes sont alors découpés en k +1 zones, qui sont réparties entre deux nouveaux chromosomes pour créer deux individus enfants. Un grand nombre de variantes au croisement à k-points, permettant des améliorations plus ou moins substantielles de l'exploration de l'espace de recherche, sont disponibles dans la littérature [BAU97] mais elles ne seront pas présentées ici.

En revanche, le croisement uniforme considère successivement chaque bit ou locus. Pour chacun d'entre eux, la valeur de celui du parent 1 ou du parent 2 est affectée à l'enfant 1 ou à l'enfant 2, selon un nombre tiré aléatoirement. Ceci revient donc à un croisement à *k*-points pour lequel *k* varierait aléatoirement pour chaque occurrence de la procédure de croisement. La méthode retenue dans le cadre de cette étude est le cas le plus simple de croisement à *k*-points, c'est-à-dire le croisement à un point de coupure présenté sur la figure 1.

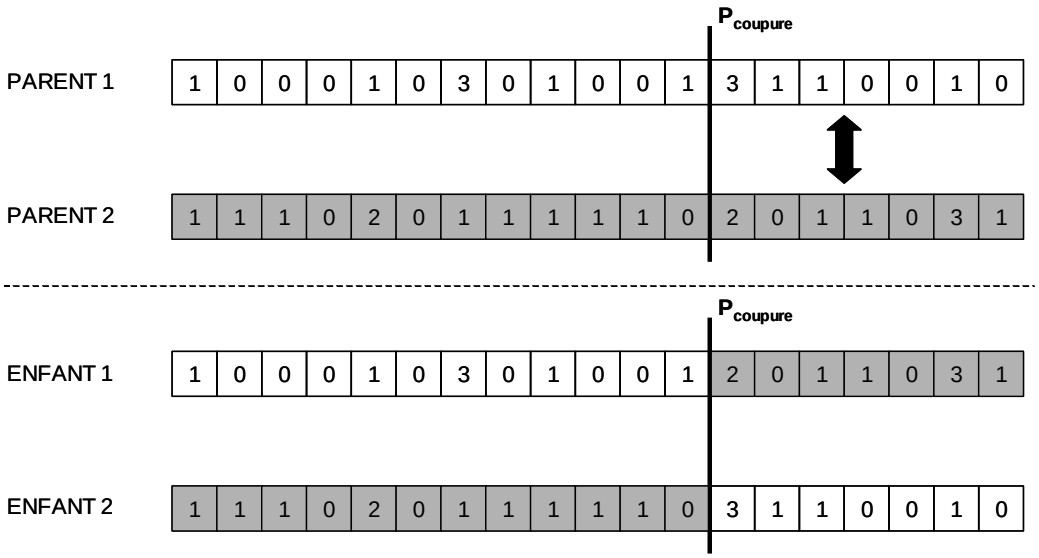


Figure 1 – Croisement à un point de coupure

La remarque étant à l'origine de la modification du premier type de codage (discrétisé séparé) est illustrée par la figure 2.

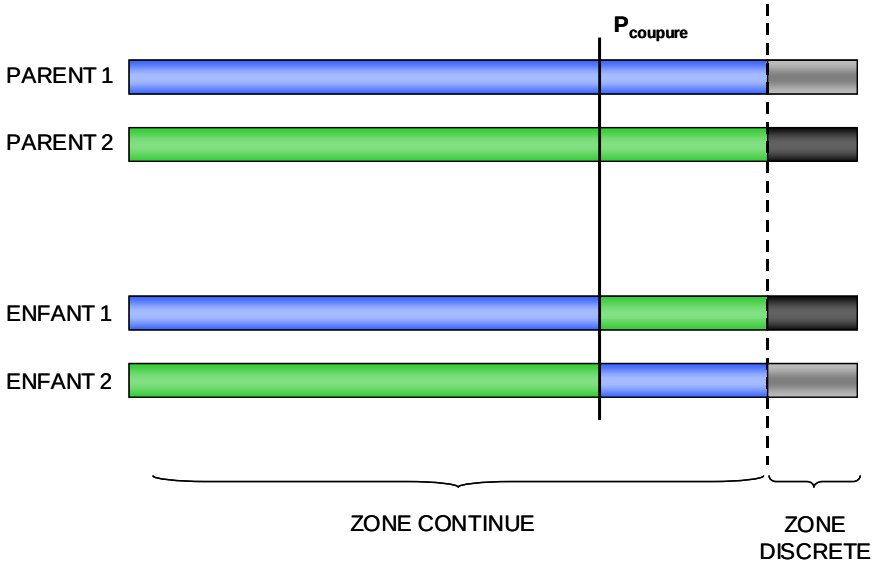


Figure 2 – Effet du croisement, codage avec séparation des variables mixtes

La zone continue du chromosome étant seize fois plus grande que la zone discrète, le point de coupure, déterminé aléatoirement, a de très grandes chances de tomber à l'intérieur de la première, laissant la seconde inchangée. Au contraire, dans le cas du codage « croisé », pour lequel les variables continues et entières sont mélangées, on remarque aisément sur la figure 3 que le croisement à un point de coupure permet un brassage efficace des deux types de gènes.

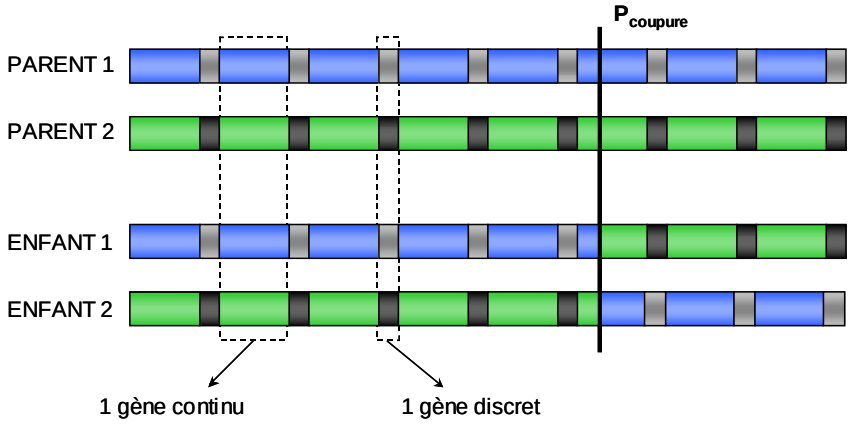


Figure 3 – Effet du croisement, codage avec mélange des variables mixtes

1.2 – Procédure de mutation

Concernant la mutation, deux cas sont à prendre en compte, dus à la nature des bits ou loci modifiés. Dans le cas d'un bit, la valeur de celui-ci est remplacée par son complément binaire. En revanche, dans le cas des loci représentant le nombre d'équipements par étage, ceux-ci sont diminués d'une unité. Bien sûr, étant donné que toutes les étapes opératoires doivent exister, cette procédure n'est appliquée que dans le cas où la valeur du locus est strictement supérieure à un. La procédure est illustrée par un exemple concret donné sur la figure 4.

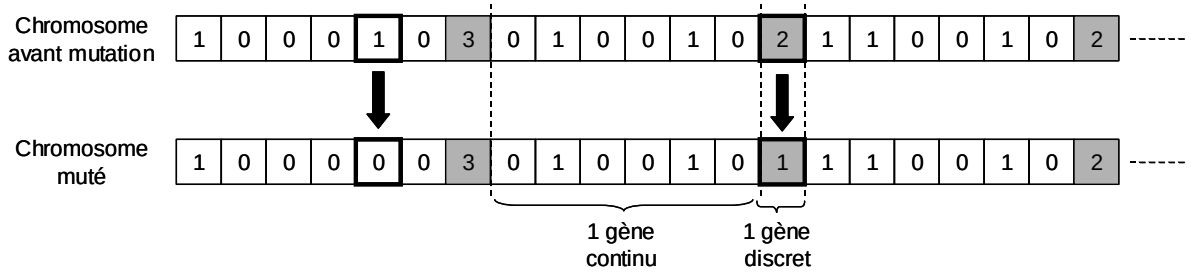


Figure 4. – Exemple de procédure de mutation

2 – Codage mixte réel-discret

Le cas des chromosomes mixtes est traité dans cette partie. Le croisement SBX [DEB95] est tout d'abord détaillé. Dans un deuxième temps, la procédure de mutation proposée dans [DEB96] est exposée, sachant qu'elle n'est appliquée qu'aux loci réels du chromosome. Pour les loci à valeur entière, codant le nombre d'équipements en parallèle à chaque étape, la procédure de mutation précédente a été conservée.

2.1 – Croisement SBX

De la même manière que pour la majorité des opérateurs de croisement appliqués à des loci à valeur réelle, le croisement SBX fonctionne en créant deux loci enfants à partir d'une combinaison linéaire des loci parents (cf. chapitre 3, équation (9)). La particularité du croisement SBX repose sur le choix des facteurs de pondération pour chaque locus parent. Ce choix est effectué en se basant sur l'analyse d'un *facteur d'expansion*, évaluant la capacité de l'opérateur de croisement à étaler où regrouper les loci enfants $y^{(i)}$ par rapport aux loci parents $x^{(i)}$. Il est défini dans [DEB95] comme ci-après :

$$\beta = \left| \frac{y^{(1)} - y^{(2)}}{x^{(1)} - x^{(2)}} \right| \tag{1}$$

Ainsi, si $\beta < 1$, le croisement est dit « contractant » tandis qu'à l'inverse, $\beta > 1$ dénote un croisement « étalant ». Lorsque $\beta = 1$, le croisement est stationnaire. Se penchant plus particulièrement sur le cas du croisement à un point de coupure pour un codage binaire, Deb et Agrawal [DEB95] utilisent deux de ses propriétés, évoquées dans le chapitre 3, pour identifier la distribution de la probabilité $C(\beta)$ de l'occurrence de β, tracée sur la figure 5.

La courbe de la figure 5 met en évidence la séparation entre les cas « contractant » et « étalant », ainsi qu'une tendance générale : il est très probable que les enfants soient créés près de l'un des deux parents. Cependant, la discontinuité pour le cas d'un croisement stationnaire ($\beta = 1$) ne facilite pas l'utilisation de cette distribution de probabilité. Par conséquent, il est proposé dans [DEB95] de faire l'hypothèse d'une distribution polynomiale de la forme :

$$\begin{cases} C(\beta) = 0{,}5(n+1)\beta^n \\ C(\beta) = 0{,}5(n+1)\dfrac{1}{\beta^{n+2}} \end{cases} \tag{2}$$

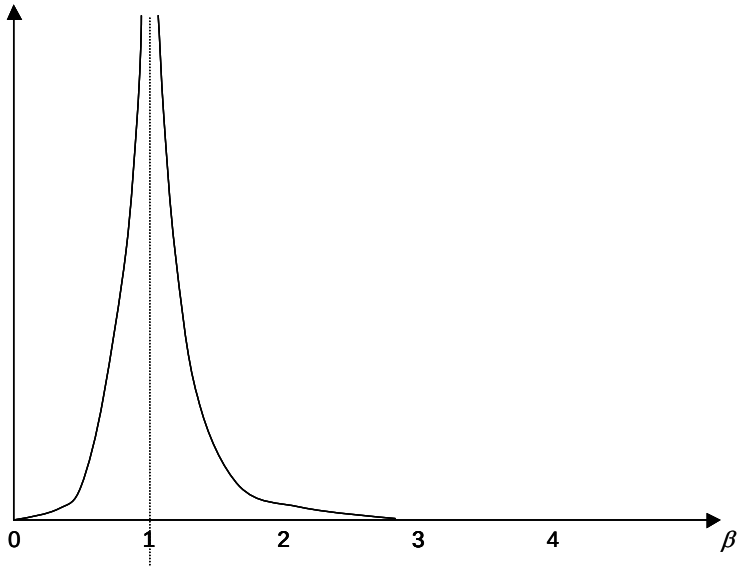


Figure 5 – Distribution de probabilité C(β), cas contractant et étalant

De grandes valeurs de *n* vont logiquement renforcer la tendance évoquée précédemment, à savoir une probabilité de plus en plus importante de créer des individus enfants proches des individus parents. Au contraire, si *n* prend de petites valeurs, la distribution tendra vers une allure plus uniforme. La distribution polynomiale est tracée sur la figure 6 pour différentes valeurs de *n*.

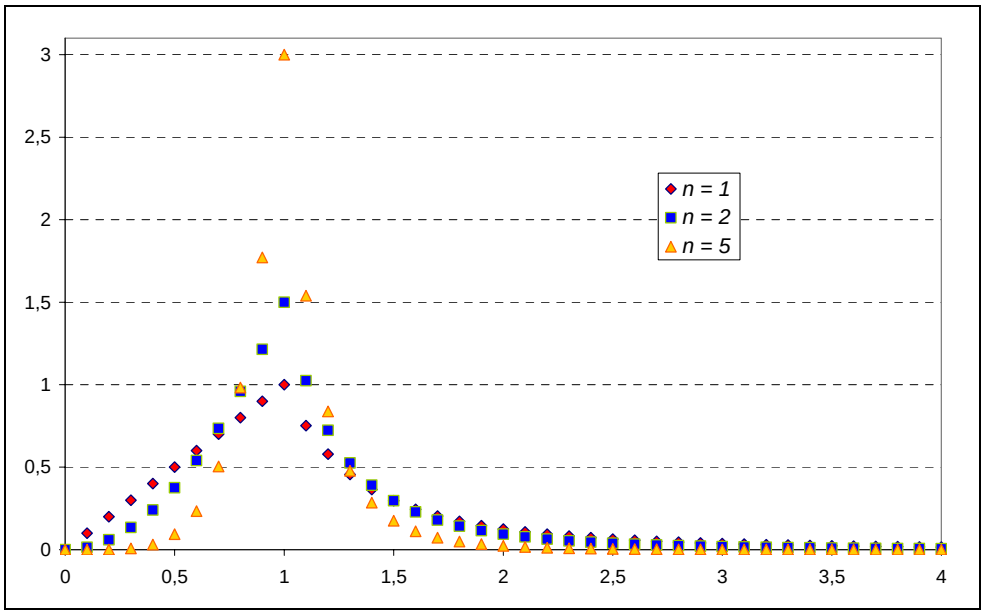


Figure 6 – Distribution polynomiale pour le croisement SBX

De l'analyse précédente peut alors être déduite une procédure particulière de croisement, simulant effectivement les propriétés de dispersion du croisement à un point de coupure pour un codage binaire. Un nombre réel *u* est tiré aléatoirement et l'abscisse β^* correspondant à une probabilité égale à *u* selon la distribution $C(\beta)$ est calculée grâce à la formule :

$$\begin{cases} \beta^* = (2u)^{\frac{1}{n+1}} & \text{si } u \leq 0{,}5 ; \\ \beta^* = \left[\dfrac{1}{2(1-u)}\right]^{\frac{1}{n+1}} & \text{sinon.} \end{cases} \quad (3)$$

Pour le cas de variables bornées, une modification doit être apportée à l'équation (3), qui devient alors :

$$\begin{cases} \beta^* = (\alpha u)^{\frac{1}{n+1}} & \text{si } u \leq 1/\alpha ; \\ \beta^* = (0{,}5-\alpha_1 u)^{\frac{1}{n+1}} & \text{sinon.} \end{cases} \quad (4)$$

Avec :

$$\alpha_1 = 2-\beta^{-(n+1)} \quad (5)$$

$$\alpha_2 = 1+\frac{2}{y^{(2)}-y^{(1)}} min\left[x^{(1)}-x^U, x^L-x^{(2)}\right] \quad (6)$$

Enfin, les individus enfants sont créés de la manière suivante :

$$\begin{cases} y^{(1)} = 0{,}5.[(1+\beta^*).x^{(1)} + (1-\beta^*).x^{(2)}] \\ y^{(2)} = 0{,}5.[(1-\beta^*).x^{(1)} + (1+\beta^*).x^{(2)}] \end{cases} \quad (7)$$

Cette procédure est appliquée à tous les loci avec une probabilité p_x, qu'ils aient une valeur réelle ou entière. Mais dans le dernier cas, seule la partie entière des loci résultants est conservée puisque la procédure retourne typiquement des valeurs réelles. La probabilité p_x sert à contrôler la perturbation des chromosomes en la limitant à une partie des gènes, comme c'est le cas pour un croisement à un point de coupure appliqué à un codage binaire classique.

La nécessité d'ajuster deux paramètres pour la procédure de croisement SBX a amené à effectuer certaines études de sensibilité. Celles-ci ont montré que les meilleurs résultats étaient obtenus avec une probabilité p_x égale à 0,2 (alors qu'elle vaudrait 0,5 pour un croisement classique adapté à un codage binaire) tandis que l'ordre n de la distribution polynomiale le plus adéquat vaut 1. Vu l'allure des courbes tracées sur la figure 6, cette valeur de n correspond à une probabilité plus importante de créer des individus éloignés de leur parents, c'est-à-dire conduisant préférentiellement à l'exploration de l'espace de recherche.

2.2 – Mutation paramétrée

La mutation choisie dans un premier temps s'applique simplement aux loci à valeur réelle. Elle est construite comme une perturbation du vecteur initial x avec une amplitude Δ_{max} :

$$y = x + \delta.\Delta_{max} \tag{8}$$

Etant donné que la variable x est bornée par x^L et x^U, il est trivial que Δ_{max} soit égal à la différence $(x^U - x^L)$. De manière similaire au croisement SBX, Deb et Goyal [DEB96] proposent de faire obéir le facteur de perturbation δ à une loi de distribution de probabilité polynomiale, de la forme :

$$P(\delta) = 0{,}5.(n+1)(1-|\delta|)^n \tag{9}$$

La dépendance en n de la loi précédente, valide sur l'intervalle [-1,1], est tracée sur la figure 7.

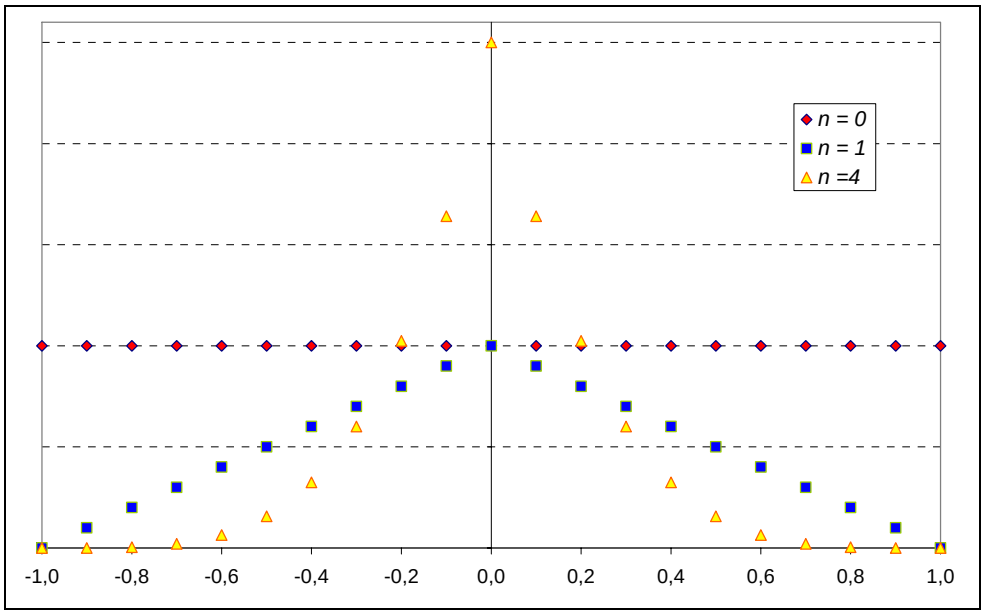


Figure 7 – Distribution polynomiale pour la procédure de mutation

Pour des variables bornées, un nombre réel u est tiré aléatoirement et l'abscisse correspondante selon $P(\delta)$ vaut :

$$\begin{cases} \delta^* = \left[2u+(1-2u)(1-\delta')^{n+1}\right]^{\frac{1}{n+1}}-1 & \text{si } u \le 0{,}5 ; \\ \delta^* = 1-\left[2(1-u)+2(u-0{,}5)(1-\delta')^{n+1}\right]^{\frac{1}{n+1}} & \text{sinon.} \end{cases} \tag{10}$$

Une étude de sensibilité a été également réalisée pour choisir une valeur adéquate de l'ordre n de la distribution polynomiale. Il est finalement apparu que les meilleurs résultats étaient indéniablement obtenus pour une valeur de n égale à 0 qui, si l'on se reporte à la figure 6, correspond à une distribution uniforme de δ. Cette dernière permet d'atteindre de manière équiprobable tous les points de l'intervalle de définition de la variable impliquée par la procédure de mutation. Par conséquent, de la même manière que pour la procédure de croisement SBX, une option privilégiant l'exploration de l'espace de recherche est retenue.

CHAPITRE 4

PROBLEMES EN VARIABLES ENTIERES

Dans la formulation retenue jusqu'à présent pour représenter le problème de conception d'ateliers discontinus, les tailles des équipements (volumes V_j pour les équipements discontinus et capacités de traitement R_k pour les appareils semi-continus) sont représentées par des variables continues.

Une précision équivalente à l'unité (codages discrétisés séparé et croisé), voire à plusieurs décimales (codage mixte et solveurs de *GAMS*), leur a été affectée lors de la résolution numérique. Or, il va de soi que, dans le monde réel, des appareils définis avec un tel degré de précision n'existent pas. Au contraire, ceux proposés par les équipementiers suivent des gammes de volume (ou de capacité de traitement, ou de tout autre indicateur de taille équivalent). Par conséquent, un progrès vers le réalisme du modèle peut être apporté en changeant la nature de ces variables qui, jusqu'alors continues, deviennent discrètes. Cette approche est d'ailleurs celle adoptée dans les travaux précédents de l'équipe, au sein de simulateurs à événements discrets (voir [DIE04], [BAU97] ou [BER99]).

Ce chapitre, dédié à l'étude de ce cas, se divise en deux parties. Le mode opératoire relatif au traitement du nouveau problème est exposé dans un premier temps. Puis, la partie suivante présente les résultats obtenus avec les trois méthodes d'optimisation, suivant deux modes différents de discrétisation des variables continues.

1 – Présentation des nouvelles conditions opératoires

La mise en œuvre du nouveau mode de représentation de l'atelier passe par une simple modification du modèle employé jusqu'à présent. L'hypothèse (ix) émise dans la partie 1.2 du chapitre 2 est remplacée par la suivante : les volumes des appareils peuvent adopter un ensemble discret de valeurs entières, défini dans les données du problème. Ainsi, dans le nouveau problème résultant, toutes les variables d'optimisation sont maintenant entières. Il est précisé que des algorithmes sont dédiés au traitement de problèmes en variables purement discrètes (la méthode de Ford-Fulkerson, par exemple), mais ils n'ont pas été retenus pour la raison suivante : si toutes les variables d'optimisation sont entières, de nombreuses variables internes au modèle sont toujours réelles (tailles de lots et temps de traitement, par exemple). Par conséquent, l'emploi des techniques d'optimisation en variables mixtes utilisées dans les chapitres précédents reste valide et la méthode de Ford-Fulkerson ne sera pas testée ici. Les quelques adaptations nécessaires à leur application au nouveau problème sont présentées ci-dessous, tandis que la méthodologie adoptée pour le développement des calculs est détaillée par la suite.

1.1 – Mise en œuvre dans les méthodes d'optimisation

Le changement de nature des variables continues a des répercussions de faible ampleur sur les méthodes d'optimisation. Tant pour la modélisation dans *GAMS* qu'en ce qui concerne l'implémentation de l'algorithme génétique, la modification est sans effet sur les parties des programmes autres que celles réservées à la définition des variables. Dans les deux cas, les bornes inférieures des variables continues dans le problème initial ont été conservées. En revanche, les bornes supérieures ont été parfois augmentées légèrement à cause du mode opératoire retenu (voir partie 1.2).

1.1.1 – Méthodes déterministes

Dans *GAMS*, les volumes et capacités de traitement des équipements sont représentés de la même manière que les nombres d'équipements par étape, c'est-à-dire en créant des variables binaires y_{jp} associées, pour chaque étape j, à l'élément p de la gamme de volumes. Si y_{jp} vaut 1, alors l'équipement considéré a la taille t_p. Par ailleurs, pour chaque étape opératoire, une contrainte est rajoutée de manière à forcer les équipements à ne pouvoir adopter qu'une et une seule taille, i.e. pour chaque j, il y existe un unique p^* tel que $y_{jp^*} = 1$. Considérant les volumes V_j, qui peuvent prendre les P valeurs possibles t_1 à t_P, l'expression analytique des principes formulés ci-dessus s'écrit au moyen des équations suivantes :

$$V_j = \sum_{p=1}^{P} y_{jp}.t_p \qquad \text{avec} \quad y_{jp} \in \{0,1\},\ j=1,\ NEtapes \tag{1}$$

$$\sum_{p=1}^{P} y_{jp} = 1 \qquad j=1,\ldots,\ NEtapes \tag{2}$$

1.1.2 – Algorithme génétique

Dans le cas de l'algorithme génétique, le codage employé précédemment, pour lequel les variables continues sont discrétisées, est réutilisé tel quel, avec une discrétisation plus large. Ainsi, c'est toujours une variable réduite, comprise entre 0 et 1, qui code l'indice de la gamme de volume avec une précision dépendant du nombre total de gammes possibles : considérant qu'un volume V_j peut adopter un ensemble de valeurs discrètes de cardinal P, la précision minimale sur la variable réduite correspondante est $1/P$.

Par conséquent, ce nouveau codage est équivalent au codage discrétisé des chapitres précédents, avec un pas de discrétisation plus large. La méthode de la boîte de poids peut donc être reprise telle quelle, seul le nombre de décimales codées de la variable réduite étant inférieur. Concernant la disposition des gènes codant les taille ou nombre d'équipements dans

une étape, c'est celle propre au codage discrétisé croisé (cf. chapitre précédent) qui a été retenue. Toutes les procédures de mutation et de croisement concernées ont également été conservées identiques. Relativement à la gestion des contraintes, une technique d'élimination sur les premières instances, puis un tournoi unique pour les suivantes, ont été retenus. Cependant, la transition entre les deux méthodes ne s'est pas toujours située entre les exemples 6 et 7 comme cela avait été le cas auparavant. Ce point sera développé par la suite.

1.2 – Méthodologie

De nouveaux calculs sont donc effectués dans l'optique de mettre à l'épreuve, sur des problèmes en variables purement discrètes, les méthodes d'optimisation qui avaient jusqu'alors été étudiées sur des problèmes en variables mixtes. Le même jeu d'exemples est de nouveau résolu au moyen des trois méthodes d'optimisation. Pour compléter les données du problème, il faut simplement fixer le mode de discrétisation des tailles d'équipements.

Les exemples précédemment traités dans l'équipe ([DIE04] ou [DED01]) proposent systématiquement trois tailles, une petite, une moyenne et une grande. Dans le cadre de cette étude, l'objectif est de décrire la globalité de l'intervalle défini par les bornes inférieure et supérieure utilisées quand les tailles sont des variables continues. Cet intervalle est assez large puisqu'on a d'une part V_{min}=250 [L] et R_{min}=300 [L.h^{-1}] et, d'autre part, V_{max}=10000 [L] et R_{max}=10000 [L.h^{-1}]. En conséquence, cinq gammes de volume sont proposées ici, valant 250, 2500, 5000, 7500 et 10000 [L] pour les équipements discontinus ; 300, 2500, 5000, 7500 et 10000 [L.h^{-1}] pour les équipements semi-continus.

Plusieurs remarques s'imposent. Tout d'abord, d'un point de vue mathématique, il est utile de vérifier que la fonction de discrétisation n'a pas d'influence sur la solution trouvée : quelle que soit la discrétisation employée, le jeu de variables devrait logiquement varier de façon minime, autour de celui correspondant à l'optimum en variables mixtes déterminé précédemment.

Par ailleurs, l'objectif est également d'éprouver les trois méthodes sur des problèmes de plus en plus complexes. Ainsi, dans un deuxième temps, les variables de taille des équipements sont discrétisées suivant des intervalles de 10, 100, 500 et 1000 unités ([L] ou [L.h^{-1}]). La borne inférieure reste systématiquement fixe dans, égale à celle fixée pour les variables continues. En revanche, la borne supérieure varie de manière minime selon les cas, dans la mesure où l'amplitude $(V_{max}–V_{min})$ ou $(R_{max}–R_{min})$ de l'intervalle global doit être un multiple du pas de discrétisation.

Ainsi, on a :

- V_{max}= 10000 [L] et R_{max}=10000 [L.h^{-1}] lorsque l'intervalle de chaque gamme est égal à 10 unités.
- V_{max}= 10050 [L]. et R_{max} =10000 [L.h^{-1}] lorsque l'intervalle de chaque gamme est égal à 100 unités.
- V_{max}= 10250 [L] et R_{max} =10300 [L.h^{-1}] lorsque l'intervalle de chaque gamme est égal à 500 ou 1000 unités.

Du point de vue de la complexité, il est prévisible que le mode de fonctionnement par tailles discrètes implique une augmentation de la difficulté pour les solveurs de *GAMS* tandis qu'au contraire, il facilite la tâche de l'AG. En effet, comme observé lors des calculs en variables mixtes, la variation du nombre de variables continues n'a qu'une influence réduite sur la difficulté du problème pour les méthodes déterministes. En revanche, cette remarque n'est pas vraie pour les variables entières. Le passage d'un mode mixte à un mode entièrement discret équivaut globalement à une augmentation importante de la combinatoire : pour la discrétisation selon cinq gammes de volumes, la combinatoire supplémentaire vaut $5^{NEtapes}$, soit une augmentation conséquente de la complexité.

Par contre, en ce qui concerne l'AG, le seul changement est le pas plus important de discrétisation des variables. Cela signifie qu'il est possible de coder les variables sur un nombre moins élevé de bits. Par exemple, pour la même discrétisation avec cinq gammes de volumes, la précision nécessaire sur la variable réduite correspondante vaut 0,2. Il est donc suffisant de coder une décimale, signifiant quatre bits au moyen de la boîte de poids, soit quatre fois moins que pour le mode de fonctionnement mixte.

2 – Calculs en variables entières

Les résultats correspondant à la forme de discrétisation à « gros grain », i.e. avec cinq gammes de taille au total, sont présentés dans un premier temps. Puis suivent ceux réalisés avec des intervalles de discrétisation plus fins. Le très important volume de calcul correspondant au traitement de ces problèmes a pu être effectué grâce à une collaboration avec l'Institut de Recherche en Informatique de Toulouse, dans le cadre du projet GridMip. GridMip est un cluster comprenant d'ores et déjà une soixantaine de processeurs et dont la vocation est de mettre à disposition, pour divers laboratoires scientifiques, une plate-forme fournissant les moyens technologiques d'encourager l'utilisation de nouvelles techniques de calcul. Cette collaboration permet donc de présenter les résultats qui vont suivre.

2.1 – Discrétisation à « gros grain »

Il est précisé que, par la suite, les solutions obtenues sont comparées à l'optimum déterminé par *SBB* sur les problèmes en variables mixtes. Il est nécessaire de signaler que ces valeurs ne constituent pas un objectif à atteindre (les problèmes en variables discrètes et en variables mixtes sont évidemment des problèmes différents). Les optima obtenus précédemment servent juste de point de référence pour comparer les différents résultats.

2.1.1 – Résultats numériques

Les résultats obtenus pour chaque exemple au moyen des trois méthodes, ainsi que les temps de calcul correspondants, sont présentés dans le tableau 1. La première ligne rappelle la valeur de l'optimum obtenu précédemment sur le problème en variables mixtes. Pour *SBB*, les solutions en italique indiquent que les conditions d'optimalité du résultat n'ont pas été atteintes avec une exécution de 24 [h] (la solution proposée est néanmoins faisable).

Exemple		1	2	3	4	5	6
Optimum mixte		166079	120777	125146	356610	620638	766031
DICOPT++	Résultat [$]	180637	208431	-	-	-	-
	Temps CPU	10 [s]	< 1 [s]	-	-	-	-
SBB	Résultat [$]	180637	142436	141617	376701	666609	828785
	Temps CPU	1 [s]	33 [s]	3,6 [min]	25 [s]	16 [min]	3,7 [h]
AG	Résultat [$]	180637	142436	141617	376701	666562	828785
	Temps CPU	< 1 [s]	< 1 [s]	2 [s]	3 [s]	2,8 [min]	4 [s]

Exemple		7	8	9	10	11
Optimum mixte		957270	1925887	2894504	3865106	4786804
SBB	Résultat [$]	1016777	*2432203*	*3800364*	*5008147*	*6456102*
	Temps CPU	13,3 [h]	*24 [h]*	*24 [h]*	*24 [h]*	*24 [h]*
AG	Résultat [$]	1016777	2065769	3104623	4144808	5290474
	Temps CPU	9 [s]	14 [s]	7 [min]	34,3 [min]	1,6 [h]

Tableau 1 – Résultats des méthodes pour cinq gammes de volume

L'évolution des temps de calcul est tracée sur la figure 1a. Le temps de calcul de l'AG pour la résolution en variables mixtes avec un codage discrétisé croisé est également rappelé. Une légère différence de mode opératoire est mise en valeur ici : en effet, la gestion des contraintes par tournoi unique a été appliquée dès l'exemple 6 et non à partir de l'exemple 7, comme cela avait fait lors de la résolution des problèmes en variables mixtes. Ce changement est dû à l'augmentation accrue du temps de calcul sur les premiers exemples, lorsque la discrétisation est relativement large. Cette dernière a deux effets antagonistes. Le premier est évidemment une diminution de la combinatoire, donc de la complexité du problème. En

revanche, il est aisément compréhensible que la largeur des intervalles entre chaque taille autorisée soit telle que des régions importantes de l'espace faisable initial (en variables mixtes) sont maintenant complètement exclues. Dans ce cas, c'est le second effet qui l'emporte. Cette analyse est facilement confirmée par le fait que, comme cela avait déjà été évoqué dans le chapitre précédent, l'augmentation du temps de calcul est due à la détermination d'une population initiale entièrement faisable. Le surcroît de difficulté lorsque les variables continues sont discrétisées aussi largement est mis en évidence sur la courbe 1b (seulement pour les exemples où la gestion des contraintes est réalisée par élimination).

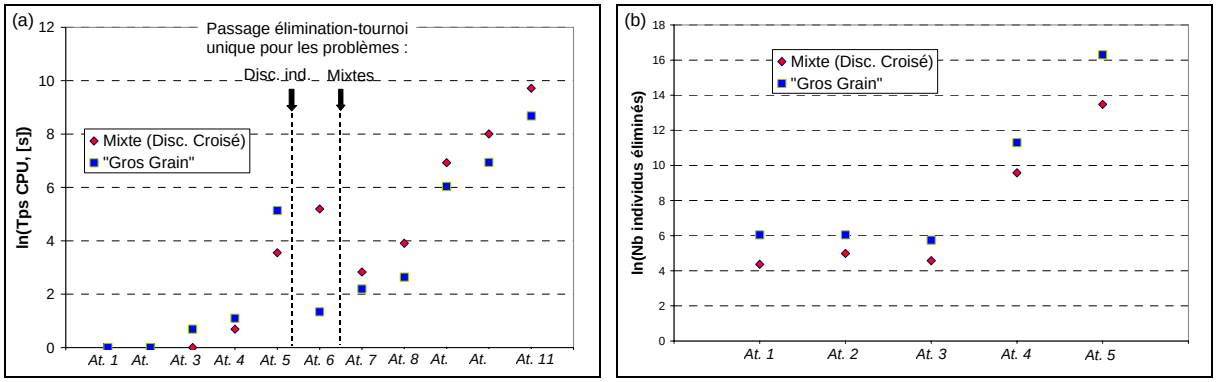


Figure 1 – Temps de calcul (a) et nombre d'individus visités pour créer la population initiale (b), problèmes mixte et à discrétisation « gros grain »

Par contre, sur les exemples pour lesquels la gestion des contraintes est effectuée au moyen d'un tournoi unique, l'obstacle de la faisabilité de la population initiale est évité et c'est alors le premier effet, i.e. la diminution de la combinatoire, qui est prépondérant. L'influence directe de la taille du chromosome sur l'efficacité de la résolution est ainsi facilement visualisable sur la deuxième partie de la courbe 1a.

Vis-à-vis de l'analyse de la qualité des solutions obtenues, l'écart entre l'optimum en variables mixtes et la meilleure solution trouvée avec des variables discrètes est illustré sur le graphique de la figure 2. Pour l'exemple 1, les trois méthodes sont unanimes sur la valeur de l'optimum en variables discrètes. L'écart avec l'optimum trouvé en variables mixtes est relativement important et s'explique aisément en examinant le vecteur de variables à la solution. Pour rappel, l'atelier 1 comporte simplement trois étapes discontinues. La meilleure structure d'atelier (variables discrètes), à savoir un équipement par étage, a été déterminée par les trois méthodes. Mais les variables de taille d'équipements valent (5000,5000,5000) dans le cas traité ici alors qu'elles étaient égales à (2854,4280,5707) pour le problème en variables mixtes. La large discrétisation imposée par le nouveau mode opératoire implique ici une augmentation importante du volume de l'équipement de la première étape discontinue, ce qui explique la valeur élevée du critère. Cette remarque est également applicable aux exemples

suivants, pour lesquels la différence entre optimum discret et optimum mixte ne sera jamais inférieure à 5,5 %.

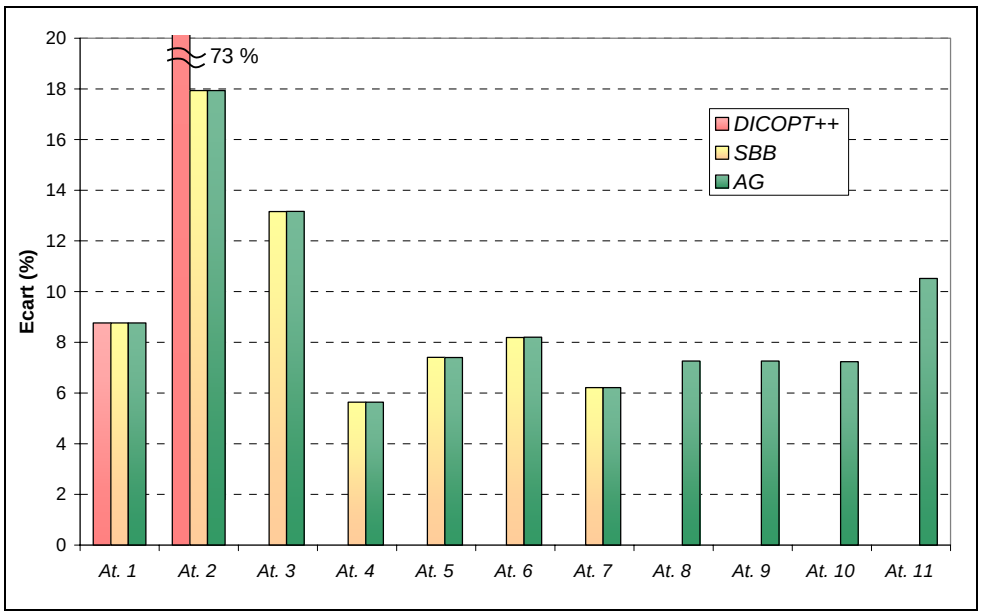


Figure 2 – Ecart à l'optimum mixte pour les trois méthodes

Les trois méthodes sont donc également performantes sur l'exemple 1 en termes de qualité de la solution. Le temps de calcul est néanmoins nettement à l'avantage de *SBB* et de l'AG dans la mesure où *DICOPT++* trouve la solution en 10 fois plus de temps que les deux premières.

Les exemples 2 et 3 sont marqués par la défaillance de *DICOPT++*. En effet, pour l'atelier 2, le résultat trouvé est très loin de l'optimum localisé par *SBB* et par l'algorithme génétique (à 73 % exactement). La structure de l'atelier déterminé correspond pratiquement à celle de l'optimum mixte, mais les volumes des équipements sont largement surdimensionnés. Sur l'exemple 3, *DICOPT++* semble définitivement dépassé puisqu'une exécution de 24 heures ne lui permet pas de trouver une solution faisable. Les combinatoires associées à ces deux exemples (calculées par la formule $3^{NEtapes}.5^{NEtapes}$) valent $1{,}71.10^{8}$ et $2{,}56.10^{9}$ pour les exemples 2 et 3 respectivement. Il est notable qu'elles soient du même ordre de grandeur que celle associée au premier atelier pour lequel *DICOPT++* avait été mis en échec lors des calculs en variables mixtes (l'exemple 6, correspondant à une combinatoire de $1{,}29.10^{8}$). Des tests effectués par la suite sur les exemples suivants ont démontré que *DICOPT++* n'est plus capable de trouver une solution pour les exemples de plus grande taille.

En revanche, le solveur *SBB* et l'AG donnent des résultats plus raisonnables, identiques pour les deux exemples bien que le temps d'une exécution de l'algorithme génétique soit plus faible que celui nécessaire à la convergence de *SBB*. Cependant, l'écart à l'optimum mixte

reste très important (près de 18 % et 13 % respectivement). L'examen des jeux de variables à la solution révèle que pour l'atelier 2, le nombre d'équipements en parallèle est différent de celui de l'optimum mixte. Mais il a été prouvé par ailleurs, en fixant les nombres d'équipements et en résolvant le problème continu associé avec *CONOPT3*, que cette structure d'atelier ne permet pas d'obtenir un optimum de meilleure qualité, quelles que soient les tailles d'équipements. Cet exemple met donc en valeur le fait que le nombre d'équipements optimal n'est pas nécessairement identique lorsqu'on passe du problème mixte au problème purement discret. Concernant l'exemple 3, les nombres d'équipements en parallèle de la solution trouvée sont identiques à ceux de l'optimum mixte. C'est l'ampleur de l'intervalle de discrétisation qui explique dans ce cas l'écart obtenu.

Concernant l'atelier 4, *SBB* et l'AG trouvent également des solutions identiques, à moins de 6 % de l'optimum mixte, ce qui est une bonne performance. Il faut rappeler que l'écueil qui avait été rencontré par l'AG en variables mixtes sur cet exemple était l'existence d'un optimum local, présentant une structure d'atelier différente de celle de l'optimum trouvé au moyen des méthodes déterministes. Ici, la structure d'atelier de l'optimum mixte a bien été identifiée, ce qui met en relief la simplification des problèmes pour l'AG, due à l'augmentation du pas de discrétisation. Le temps de calcul est une fois encore largement à l'avantage de l'AG, bien que les ordres de grandeur des temps restent toujours similaires (inférieurs à la minute tous les deux).

Les solutions trouvées par l'AG et par *SBB* pour l'exemple 5, à 7,4 % de l'optimum mixte, montrent une très faible différence (moins de 0,01 %). Cet écart n'est pas attribuable à des erreurs de précision mais bien à des jeux de variables différents. Plus particulièrement, la structure d'atelier déterminée par *SBB* est différente de celle trouvée lors de la résolution du problème en variables mixtes, tandis que celle de l'AG lui est bien identique. Ceci n'est nullement synonyme de supériorité vu la faible différence entre les deux fonctions objectif, mais c'est tout de même la première fois sur l'ensemble de ces travaux que *SBB* ne détermine pas la meilleure solution. Le temps de calcul de *SBB* augmente singulièrement puisque pour cet exemple, trouver une solution demande plus de 15 minutes. Celui de l'AG s'accroît aussi de manière significative, mais ce comportement a déjà été expliqué précédemment, par le mode de gestion des contraintes par élimination.

Sur les exemples 6 et 7, les deux méthodes encore testées trouvent de nouveau la même solution. Les temps de calcul du solveur *SBB* augmentent alors énormément : 3,7 heures et 13,3 heures de calcul sont nécessaires pour résoudre ces exemple de taille intermédiaire, alors que 3 et 4 respectivement secondes lui suffisaient pour la résolution des mêmes problèmes en variables mixtes. Pour l'AG au contraire, le passage à une gestion des contraintes par tournoi

unique permet de revenir à des temps de calculs largement inférieurs à la minute. L'AG est donc déjà bien plus efficace que la méthode déterministe.

Cette tendance se confirme sur l'exemple 8 : *SBB* est incapable de conclure en un temps de calcul inférieur à 24 heures. Il est cependant possible de récupérer la meilleure solution faisable trouvée pendant l'exécution : celle-ci est égale à 2432203 [$], soit plus de 17 % au-dessus de la meilleure solution déterminée par l'AG. Ce dernier est toujours aussi efficace puisque sa solution reste à une distance raisonnable de l'optimum en variables mixtes. La supériorité de l'AG est donc prouvée dès ce huitième exemple, non seulement en temps de calcul mais aussi concernant la qualité de la solution.

Enfin, pour les exemples 9 à 11, *SBB* n'est plus en mesure de conclure sur l'optimalité des solutions. Une solution faisable, indiquée en italique dans le tableau 1 est cependant toujours récupérable. L'augmentation de la borne inférieure (infaisable) est de plus en plus lente au fur et à mesure que les exemples deviennent plus complexes, et reste très loin d'atteindre la borne supérieure à la fin du temps d'exécution autorisé. Cette borne supérieure est également très éloignée de la solution trouvée par l'AG (à 22,2 %, 20,8 % et 22,0 % pour les exemples 9, 10 et 11 respectivement), dont le comportement reste satisfaisant. L'écart entre la solution trouvée et l'optimum en variables mixtes reste stable. Par ailleurs, comme l'indique la figure 1a, le temps de calcul reste inférieur, mais toujours comparable, à celui obtenu pour les problèmes en variables mixtes. Cette observation achève de mettre en relief l'influence de la longueur du chromosome sur la résolution. Le nombre peu élevé de bits composant le chromosome induit une combinatoire moins élevée et permet d'éviter le piège d'optima locaux comme cela a été démontré pour l'exemple 4.

Concernant l'aspect stochastique de l'AG, il importe d'étudier la répétablité des résultats, caractérisée lors des calculs précédents par les dispersions à 2 % et 5 % des solutions de chaque exécution. Pour l'étude en variables discrètes, il est maintenant possible d'utiliser un critère plus communément adopté pour juger des performances d'un algorithme génétique, à savoir la proportion d'exécutions conduisant à l'obtention de la meilleure solution trouvée. Ce critère n'était pas utilisable en variables mixtes du fait de la taille énorme de l'espace de recherche généré par les variables continues. Par contre, lorsque ces variables continues sont discrétisées selon des intervalles plus larges, il est possible de retomber plusieurs fois sur la même solution.

L'évolution de la proportion des exécutions trouvant une solution identique est illustrée sur la figure 3. Pour les trois premiers exemples, l'AG tombe presque systématiquement sur la meilleure solution trouvée. Pour le quatrième, la proportion chute à 6 % à cause de la

présence de l'optimum local correspondant à une structure différente de l'atelier et bloquant déjà l'AG lors des calculs en variables mixtes. En effet, dès l'exemple 5, qui ne présente pas de difficulté particulière, les performances de l'AG retrouvent leur niveau antérieur : 64 % des exécutions trouvent la meilleure solution. Mais la complexité combinatoire des problèmes se fait sentir dès l'exemple 6, la proportion d'exécutions retombant à 11 %, puis stagnant à 1 % à partir de l'exemple 8.

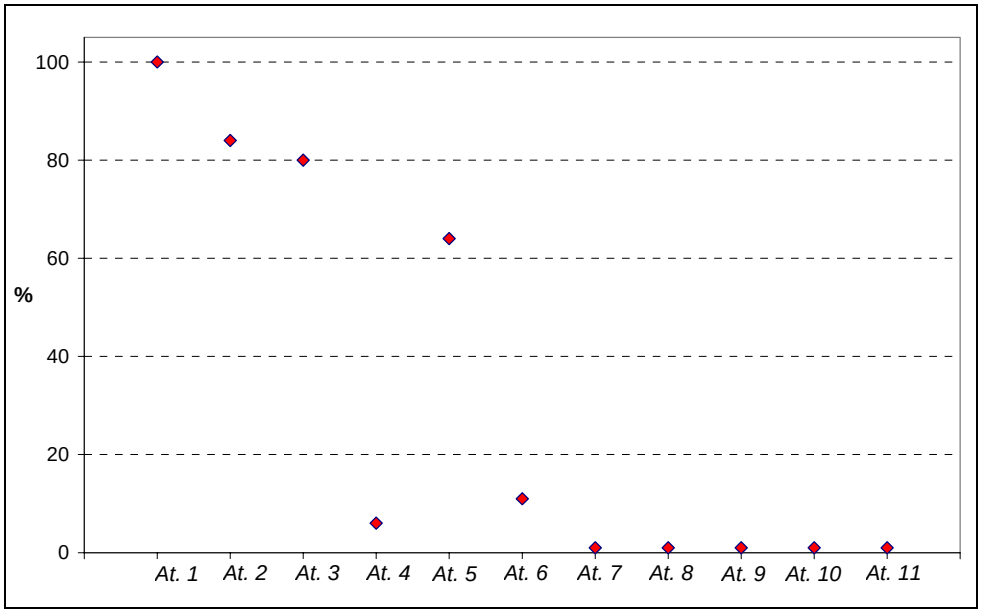


Figure 3 – Proportions d'exécutions de l'AG trouvant la même solution

Ces tendances sont confirmées par l'observation des dispersions à 2 % et 5 % présentées sur la figure 4 et comparées à celles obtenues sur le problème mixte avec le codage discrétisé croisé. Sur les trois premiers exemples, les dispersions obtenues sur le problème en variables discrètes sont meilleures que sur celui en variables mixtes.

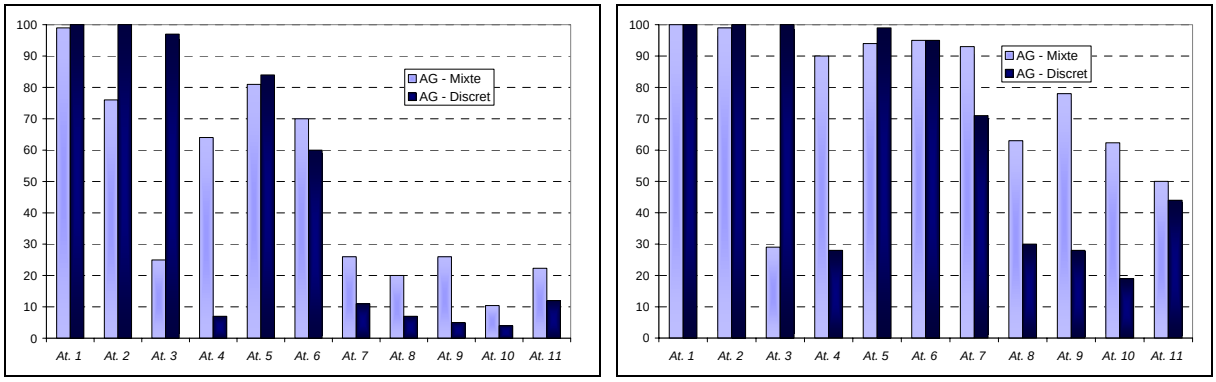


Figure 4 – Dispersions à 2 % et 5 %, problèmes mixte et discret

Pour l'exemple 4, les dispersions chutent seulement pour le problème discret, mais cela est dû à la bonne qualité de la meilleure solution trouvée. En effet, lors de la résolution du

problème en variables mixtes avec un codage discrétisé croisé, l'AG ne parvenait pas à s'extraire de l'optimum local, alors qu'ici, une faible proportion des exécutions réussit à identifier la structure du meilleur optimum. Sur les exemples 5 et 6, les performances de l'AG sur les deux types de problèmes (mixte et discret) sont relativement similaires.

Mais sur les quatre derniers exemples (8 à 11), les dispersions à 2 % et à 5 % pour le problème en variables discrètes sont systématiquement inférieures à celles correspondant au problème mixte. Cette baisse de l'efficacité de l'AG peut être expliquée, comme dans le chapitre précédent, par la difficulté à trouver une première solution faisable pendant la recherche : le large pas de discrétisation rend inaccessible des régions importantes de l'espace faisable comme l'illustre la figure 5. La proportion de solutions faisables dans la population en fin de recherche chute, alors que parallèlement, le nombre d'échecs augmente fortement. Le nombre de solutions faisables corrigé en fonction du taux d'échec souligne la difficulté au niveau de la localisation d'une première solution faisable. En effet, quand ceci est réalisé, la proportion de solutions faisables en fin de recherche se maintient globalement constante, quelle que soit la complexité de l'exemple abordé.

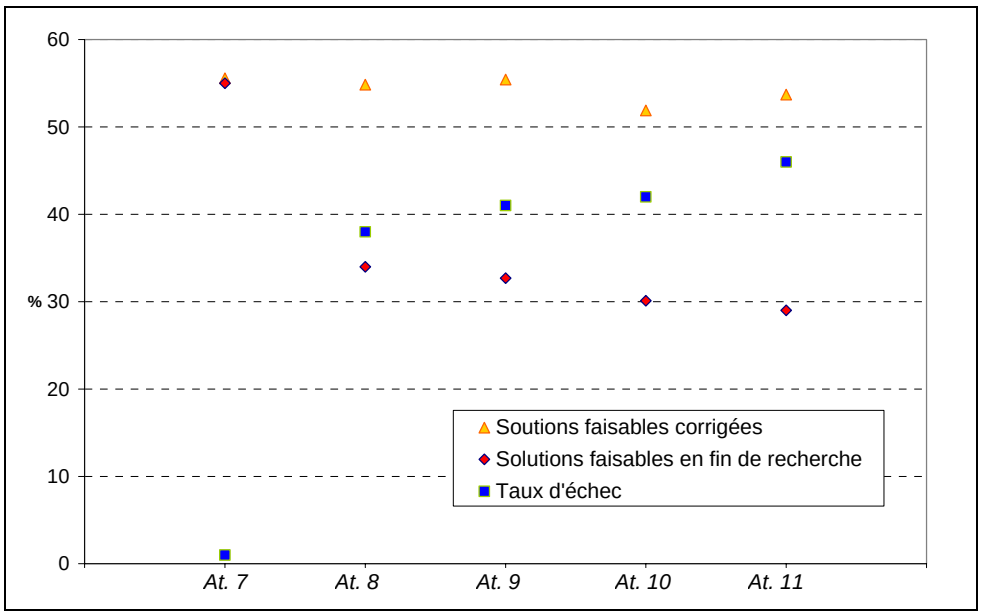


Figure 5 – Solutions faisables et taux d'échec pour les derniers exemples

2.1.2 – Conclusions

Ainsi, la discrétisation selon cinq tailles d'équipements (« gros grain ») a mis en évidence la supériorité de l'algorithme génétique sur les méthodes de Programmation Mathématique. Tout d'abord, *DICOPT++* a montré très rapidement ses limites. Il ne fournit plus aucune solution dès le troisième exemple, ce qui correspond d'ailleurs à une combinatoire concordant avec celle de l'exemple sur lequel il restait également bloqué pour les problèmes en variables

mixtes. Jusqu'à l'exemple 7, *SBB* et l'AG déterminent des solutions de qualité égale, même si elles ne sont pas toujours identiques. En revanche, le temps de calcul est nettement défavorable à *SBB* à cause de la multiplication du nombre de sommets du Branch & Bound, dus aux variables discrètes. A partir de l'exemple 8, *SBB* n'est plus capable de converger mais parvient néanmoins à fournir des solutions faisables sur tous les exemples. La forte combinatoire engendre un temps de calcul rédhibitoire, l'empêchant d'atteindre le critère d'arrêt (égalité des bornes supérieure et inférieure). Par contre, l'AG continue à être efficace : il trouve une solution à une distance toujours raisonnable de l'optimum mixte, en un temps similaire, voire inférieur à celui des calculs menés en variables mixtes. Ses performances en termes de répétabilité baissent quelque peu, mais restent également comparables aux calculs du chapitre précédent. La méthode stochastique est ainsi avantagée par son mode opératoire pour les mêmes raisons que celui-ci l'avait auparavant défavorisée lors du traitement des problèmes en variables mixtes. La conclusion pour ces calculs en variables entières est donc logiquement largement à l'avantage de cette dernière.

2.2 – Vers une discrétisation plus fine

L'étude d'une discrétisation plus fine des tailles d'équipements vise tout d'abord à conforter les tendances observées dans la partie précédente. Dans un second temps, le but est également d'analyser l'influence de la fonction de discrétisation sur le résultat obtenu. En d'autres termes, passant d'un pas de discrétisation à un autre, les éléments du vecteur solution évoluent-ils vers la valeur la plus proche de la nouvelle gamme de tailles ?

2.2.1 – Résultats sur le jeu d'exemples

2.2.1.1 – Résultats de l'AG

Concernant tout d'abord l'algorithme génétique, l'adoption d'un pas de discrétisation de plus en plus petit signifie revenir graduellement vers le mode de codage utilisé pour les problèmes en variables mixtes. Le comportement de l'AG doit donc théoriquement se situer dans une position intermédiaire entre les deux cas envisagés jusqu'alors : résolution de problèmes mixtes avec un codage discrétisé croisé et de problèmes en variables discrétisées selon une gamme à « gros grain ». Le tableau 2 présente la meilleure solution obtenue pour chaque exemple et pour les différents intervalles de discrétisation définis par le mode opératoire retenu. Le mode de gestion des contraintes est cette fois rigoureusement identique à celui employé pour les calculs en variables mixtes, c'est-à-dire élimination jusqu'à l'exemple 6 et tournoi unique à partir de l'exemple 7.

Pas [L]	Exemple 1	Exemple 2	Exemple 3	Exemple 4	Exemple 5	Exemple 6
10	166172	122364	125256	370083	622690	772718
100	166526	122748	125389	371750	623104	773526
500	167941	123352	128001	373082	628823	777108
1000	175336	126327	131858	380099	633134	784609

Pas [L]	Exemple 7	Exemple 8	Exemple 9	Exemple 10	Exemple 11
10	972042	1994946	2995129	4010943	5050716
100	972696	1980207	2995518	4008669	5083426
500	974873	2005608	3016302	4015673	5103457
1000	979555	2020934	3034971	4019652	5100880

Tableau 2 – Solutions obtenues avec l'AG

Les écarts à l'optimum du problème en variables mixtes sont représentés pour mémoire sur la figure 6. Comme prévu, ils montrent que l'AG trouve une solution restant toujours raisonnable, et logiquement, d'une qualité d'autant plus satisfaisante que le pas de discrétisation est petit. Des exceptions sont cependant à noter pour les ateliers 6, 8 et 10. Elles trouvent leur explication dans le caractère stochastique de l'AG et la combinatoire déjà élevée pour les exemples concernés, avec un pas de discrétisation égal à 10.

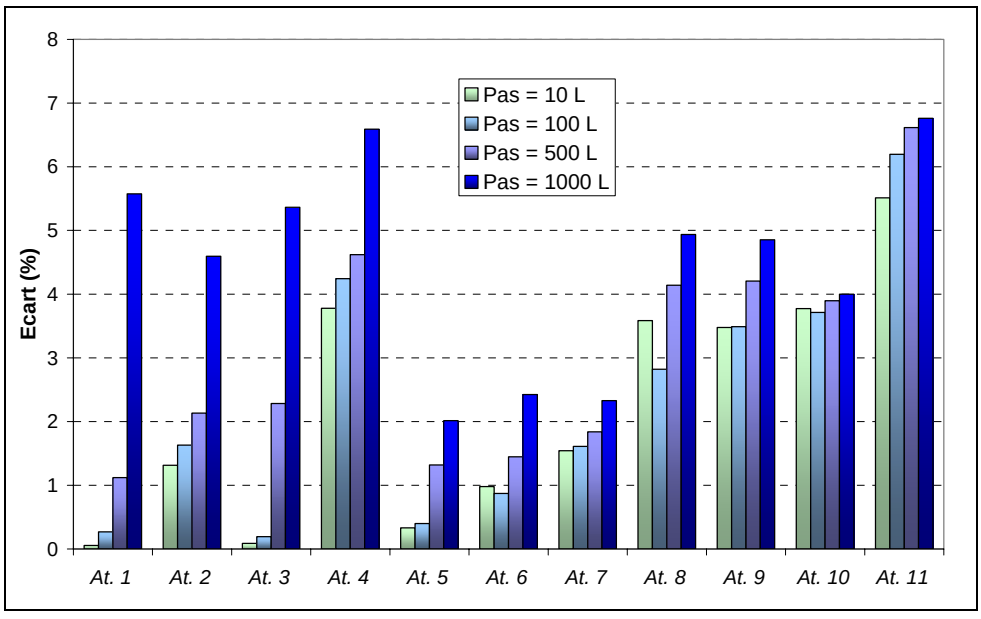


Figure 6 – Ecart à l'optimum mixte

Les dispersions sont également comparables à celles obtenues avec les modes opératoires précédents. Mais, l'aspect stochastique empêche de suivre, pour chaque exemple, une évolution logique et régulière des résultats en passant d'une gamme de tailles à une autre. La figure 7 présente simplement, pour chaque atelier, la moyenne arithmétique des dispersions à 2 % et 5 % sur le nombre de gammes testées.

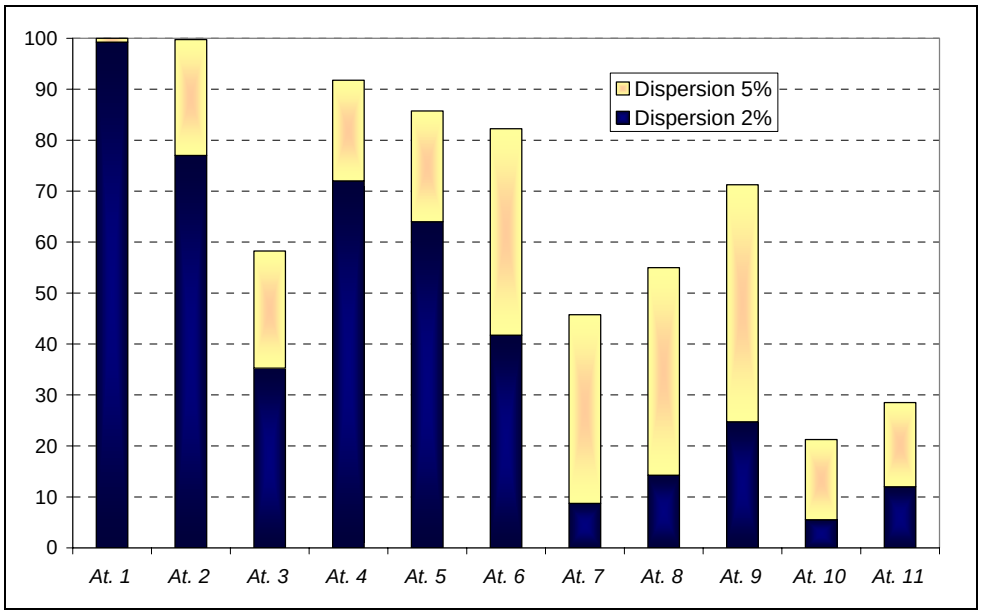


Figure 7 – Moyennes des dispersions pour les quatre pas de discrétisations

Finalement, relativement aux temps de calculs, deux situations sont à envisager. Premièrement, lorsque la gestion des contraintes est effectuée par élimination, les deux effets contradictoires évoqués plus haut sont de nouveau en concurrence : une diminution de la combinatoire du problème, parallèle à l'interdiction de régions faisables importantes lorsque le pas de discrétisation augmente. Il est alors possible de déduire quel effet est prépondérant en observant la difficulté à créer une population initiale faisable. Le nombre d'individus éliminés pour générer la première population est tracé sur la figure 8 pour les six premiers exemples. Il apparaît alors clairement que le premier aspect l'emporte pour des intervalles de discrétisation égaux à 10 [L] et 100 [L] (ou [$L.h^{-1}$]). En effet, dans ces deux cas, une première population faisable est trouvée plus facilement que pour le problème mixte.

Par contre, pour des gammes supérieures ou égales à 500 unités, de trop grandes régions de l'espace faisable sont interdites par la largeur de l'intervalle de discrétisation, et la génération de la population initiale devient plus délicate. Il est à noter que les différences entre les courbes semblent minimes simplement à cause de l'échelle logarithmique retenue. Elles varient en fait d'un facteur pouvant aller jusqu'à 10 pour l'exemple 6. La partie de gauche de la figure 9 montre que les temps de calcul correspondent à cette interprétation, puisqu'ils sont plus courts pour un pas de discrétisation allant jusqu'à 100, et au contraire plus longs pour un pas supérieur ou égal à 500.

La deuxième situation concerne les exemples pour lesquels les contraintes sont gérées par un tournoi unique. Dans ce cas, les temps de calcul sont intermédiaires entre les deux situations extrêmes abordées précédemment : problème mixte et discrétisation à « gros grain ». C'est ici la combinatoire, donc la dimension du chromosome associé qui conditionne

la rapidité de l'exécution. Or, cette dimension est invariante pour les pas de 100, 500 et 1000 unités.

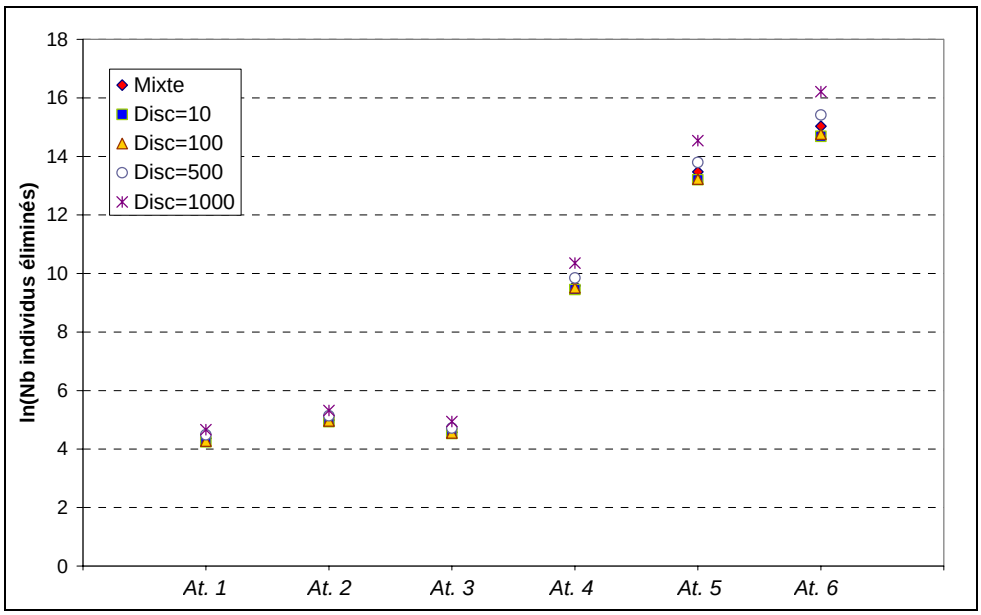


Figure 8 – Nombre d'individus éliminés pour la création de la population initiale

En effet, le nombre de tailles possibles est alors donné par $(V_{min}–V_{max})/Pas+1$, soit respectivement dans chaque cas 99, 21 et 11 : deux décimales, soit huit bits, sont toujours suffisantes pour coder toutes les valeurs discrètes. La figure 9 montre donc des allures confondues dans ces trois cas.

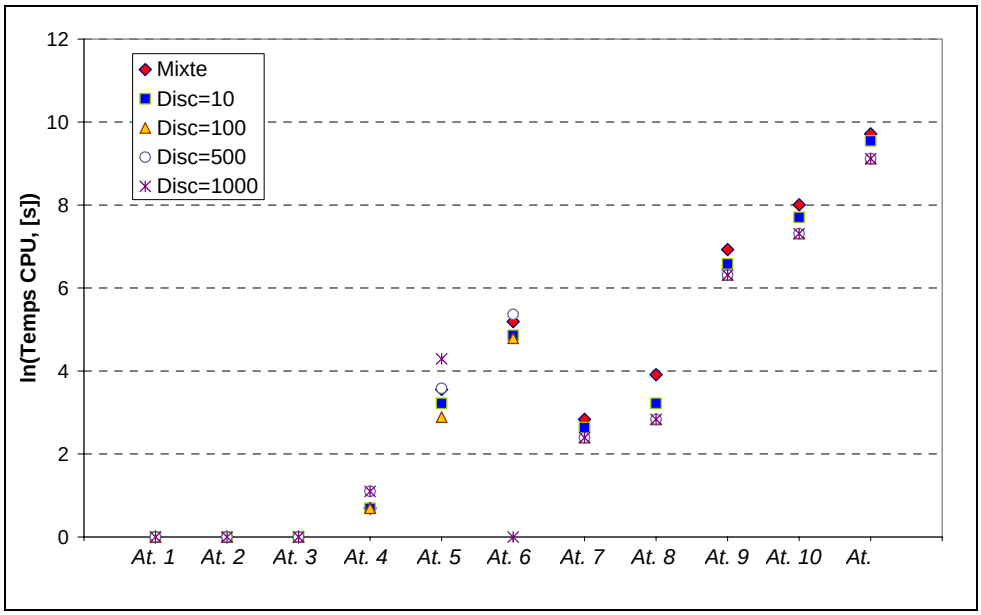


Figure 9 – Temps de calcul pour l'AG

En revanche, pour un pas de 10 [L] ou [$L.h^{-1}$], le nombre de possibilités de 976 impose trois décimales (douze bits) pour pouvoir représenter toutes les valeurs possibles. L'exécution est alors d'autant moins rapide, mais se révèle néanmoins plus efficace que pour les problèmes résolus en variables mixtes.

2.2.1.2 – Résultats des méthodes déterministes

Pour les solveurs de *GAMS*, cette discrétisation plus fine représente un tout autre enjeu. En effet, considérant P valeurs possibles pour la gamme de volumes, alors la combinatoire associée à chaque atelier s'écrit $3^{NEtapes}.P^{NEtapes}=(3.P)^{NEtapes}$. Ainsi, au vu, d'une part, des intervalles de discrétisation proposés et d'autre part, des difficultés déjà rencontrées pour résoudre le jeu d'exemples avec une discrétisation à « gros grain », il est prévisible que *DICOPT++* et même *SBB* atteignent les limites de leur capacité de résolution.

Effectivement, *DICOPT++* n'est capable de résoudre que le premier exemple, pour des pas de discrétisation égaux à 500 [L] et 1000 [L] (voir tableau 3). Lorsque le pas de discrétisation descend à 100 [L], le résultat trouvé est très éloigné de la solution obtenue par l'AG et pour le pas de discrétisation de 10 [L], plus aucune solution n'est trouvée en 24 heures d'exécution. Les combinatoires de ces deux derniers exemples valent $2{,}62.10^{7}$ et $2{,}51.10^{10}$, ce qui correspond une fois encore à l'ordre de grandeur de combinatoire pour lequel *DICOPT++* a été mis en défaut lors des calculs précédents.

	Pas=1000 [L]	Pas=500 [L]	Pas=100 [L]	Pas=10 [L]
Résultat [$]	175336	167941	232824	-
Ecart à l'optimum mixte	5,57 %	1,12 %	40,19 %	-
Temps CPU [s]	15	38	1	-

Tableau 3 – Résultats de *DICOPT++* sur le premier exemple

SBB se montre, lui, plus performant. Pour les exemples les plus simples, il réussit à fournir une solution optimale. Cependant, dès que la complexité augmente un peu, la résolution est souvent réalisée au prix de temps de calcul réellement prohibitifs. Par ailleurs, pour certaines instances d'une complexité modérée (atelier 2 avec un pas de 1000 [L] par exemple), *SBB* parvient à conclure mais ne détermine qu'un optimum local dont la valeur est bien supérieure à celle de la meilleure solution trouvée par l'AG.

Les résultats montrent l'existence d'un seuil à partir duquel *SBB* n'est plus capable de conclure sur l'optimalité du résultat et n'atteint pas le critère d'arrêt pendant une exécution limitée à 24 h : à la fin de la recherche, l'écart entre borne supérieure faisable et borne inférieure infaisable est toujours très important. Le solveur parvient néanmoins à fournir une

solution faisable mais celle-ci est généralement très éloignée du résultat de l'AG. Les calculs avec *SBB* n'ont donc pas été poursuivis. Seuls les résultats pour les exemples 1 à 8, dont certains sont partiels, sont présentés dans le tableau 4. Les cases grisées indiquent les exemples pour lesquels l'optimalité du résultat n'a pas été démontrée, ou ceux qui n'ont pas été résolus pour les raisons citées ci-dessus. L'atelier 8 n'a pas été traité ici puisque dès la discrétisation à « gros grain », aucun résultat optimal n'avait été trouvé. Le résultat en lui-même n'est pas présenté, seul l'écart à la meilleure solution de l'AG (en pourcentages) et le temps d'exécution correspondant apparaissent.

Pas	***At. 1***	***At. 2***	***At. 3***	***At. 4***	***At. 5***	***At. 6***	***At. 7***	***At. 8***
10 [L]	0 % 7,3 [min]	18,3 % 24 [h]						
100 [L]	0 % 33 [s]	0 % 14,2 [h]	13,3 % 24 [h]					
500 [L]	0 % 3 [s]	5,7 % 2,4 [h]	0 % 7,8 [h]	25,2 % 24 [h]	19,4 % 24 [h]			
1000 [L]	0 % 1,5 [s]	9,2 % 15,7[min]	0 % 1,4 [h]	0,04 % 10,1 [h]	0 % 19,8 [h]	5,2 % 24 [h]	32,5 % 24 [h]	

Tableau 4 – Récapitulatif des calculs réalisés avec *SBB*

2.2.2 – Influence du pas de discrétisation

Pour analyser plus en détail les résultats précédents, il a semblé essentiel de quantifier l'influence du pas de discrétisation sur le jeu de variables correspondant à la meilleure solution. Cette étude complémentaire ne porte pas sur tous les exemples mais sur un échantillon de trois d'entre eux, néanmoins représentatif. Il met en évidence deux comportements différents, générés par les divers pas de discrétisation envisagés :

- Un premier cas est caractérisé par l'invariance des nombres d'équipements en parallèle à chaque étape, quel que soit le pas de discrétisation employé. Les ateliers 3 et 5 ont été retenus pour l'illustrer.
- Le deuxième cas concerne les exemples pour lesquels le nombre d'équipements en parallèle varie selon le pas de discrétisation utilisé. Ce comportement est observé sur l'exemple 8.

Pour l'exemple 3, le tableau 5 donne les valeurs des variables à la solution. Le premier commentaire concerne la différence remarquable entre équipements discontinus et semi-continus. En effet, l'écart-type indiqué en dernière ligne du tableau est bien inférieur pour les premiers. Ceci s'explique par les données relatives aux problèmes : les coefficients de coût affectés aux étapes discontinues leur confèrent en effet un poids bien supérieur à celui des étapes semi-continues (cf. chapitre 2). Par conséquent, la taille de ces dernières peut varier

assez librement sans pour autant affecter fondamentalement la valeur de la fonction objectif. Ainsi, en se concentrant uniquement sur les variables des étapes discontinues, il est clair que leur variation est relativement minime, sauf pour la discrétisation à « gros grain », qui ne permet que des sauts d'amplitude très large entre chaque valeur admissible. Ce dernier type de discrétisation est largement responsable de l'augmentation de la valeur de l'écart-type, tant pour les équipements discontinus que pour les équipements semi-continus.

	Eqts discontinus [L]			Eqts semi-continus [$L.h^{-1}$]				
	B1	B2	B3	SC1	SC2	SC3	SC4	SC5
Mixte	3693,1	1754,3	2338,7	4066,0	2986,5	3538,4	3521,9	300,1
Pas=10	3740	176	2350	4040	2000	2500	3520	300
Pas=100	3650	1750	2350	6100	4100	3900	4000	300
Pas=500	3750	1750	2750	2800	2300	3800	4300	300
Pas=1000	3250	2250	3250	3300	1300	300	3300	300
« Gros grain »	5000	2500	2500	5000	300	300	2500	300
Moyenne	3847,2	1960,7	2589,7	4217,7	2164,4	2389,7	3523,6	300,0
Ec.-type	594,8	330,5	359,9	1118,	1318,4	1693,2	620,9	0,0

Tableau 5 – Solution de l'exemple 3 en fonction du pas de discrétisation

Une analyse similaire peut être effectuée sur l'exemple 5. Le déséquilibre entre étapes discontinues et semi-continues se vérifie de nouveau puisque la moyenne des écarts-types vaut respectivement 416,2 et 1294,7. La figure 10 met elle aussi en évidence cette différence de comportement. Les courbes correspondant aux équipements semi-continus sont désordonnées et sans cohérence, tandis que celles correspondant aux équipements discontinus sont pratiquement confondues, sauf, encore une fois, pour la discrétisation à « gros grain ».

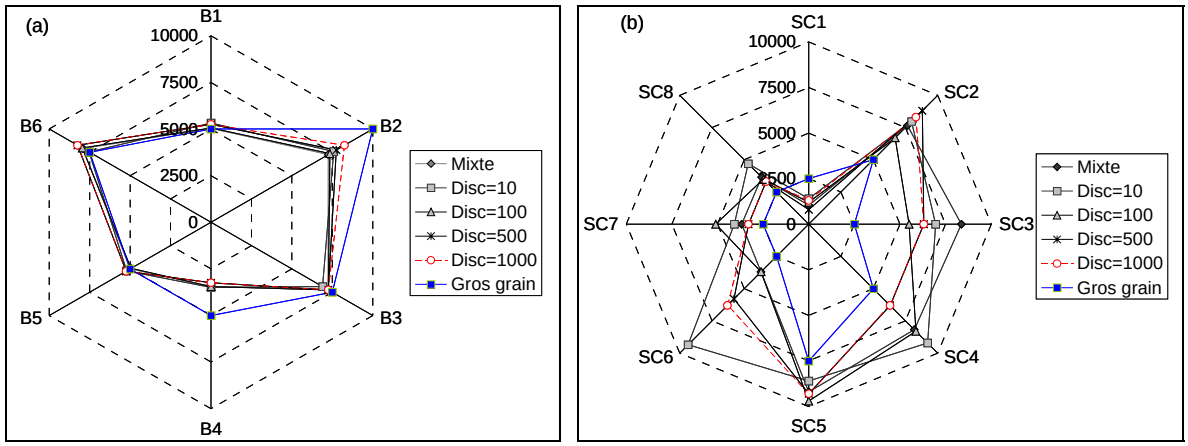


Figure 10 – Evolution des variables à la solution selon le pas de discrétisation pour l'exemple 5

a) Etapes discontinues

b) Etapes semi-continues

Le deuxième type de comportement de la solution selon le pas de discrétisation est révélé par l'exemple 8, qui montre une variation du nombre d'équipements en parallèle à la solution. La figure 11 présente des graphiques similaires aux précédents, illustrant la dispersion des variables à la solution, selon le pas de discrétisation. Seuls les volumes des étapes discontinues sont représentés, car de la même manière que dans les cas évoqués ci-dessus, les étapes semi-continues sont bien moins significatives. Ainsi, la moyenne des écarts-types relatifs à ces dernières vaut 1485,6 tandis qu'elle est égale à 839,5 pour les étapes discontinues. La figure 11 met de nouveau en relief une constance assez nette des tailles des équipements discontinus à l'optimum pour des pas de discrétisation allant jusqu'à 1000 [L]. En effet, dans tous ces cas, les courbes des volumes (figure 11a) sont effectivement très proches les unes des autres et le nombre d'équipements par étape opératoire est invariant (figure 11b). L'écart-type recalculé juste pour ces valeurs faibles de discrétisations se réduit à 377,3 ce qui renforce la preuve de la similitude des solutions.

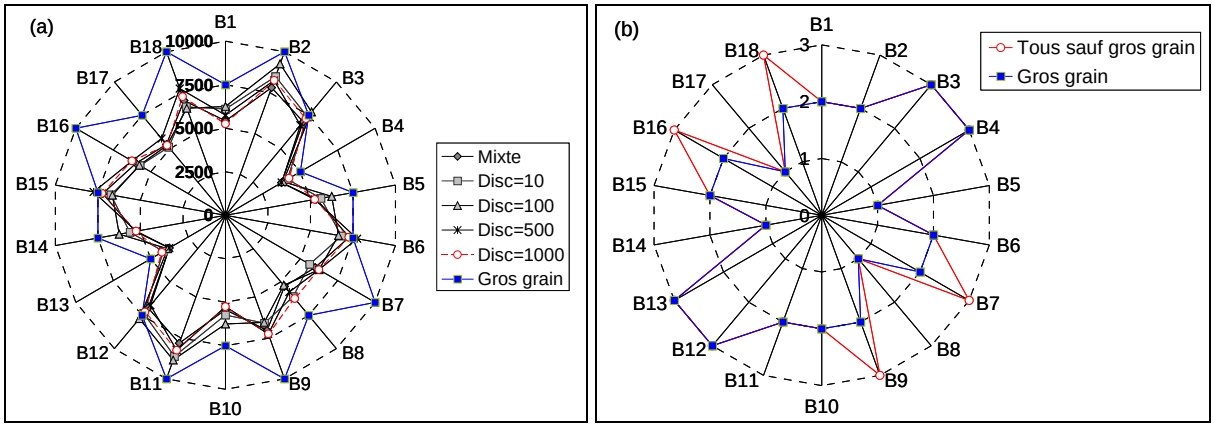


Figure 11 – Evolution des variables à la solution selon le pas de discrétisation pour l'exemple 8 : (a) volume et (b) nombre d'équipements à chaque étape

La courbe des volumes relative à la discrétisation à « gros grain » suit la même tendance, (mais reste décalée à cause de la largeur de l'intervalle de discrétisation), sauf pour les étapes B7, B9, B16 et B18. Pour celles-ci, le nombre d'équipements par étape est égal à 2 au lieu de 3 pour les autres pas. Il a été vérifié par le calcul que cette dernière option, avec des volumes associés égaux à 7500 [L] au lieu de 1000 [L], est une solution faisable pour la discrétisation à « gros grain » : elle se rapprocherait alors des autres courbes (l'écart-type recalculé pour cette configuration vaut 634,1) comme cela est montré par la figure 12. Mais la valeur du critère est alors supérieure à celle de la solution présentée sur la figure 11, ce qui montre bien l'influence d'une discrétisation très large sur la structure même de l'atelier-solution.

Remarque : il est intéressant de noter la symétrie des figures 11 et 12 par rapport au centre des graphes. Pour rappel, l'atelier 8 est construit comme la juxtaposition de deux ateliers 7, comptant neuf étapes discontinues. La symétrie centrale des graphes

s'explique alors facilement par la répétition de deux structures-solutions identiques.

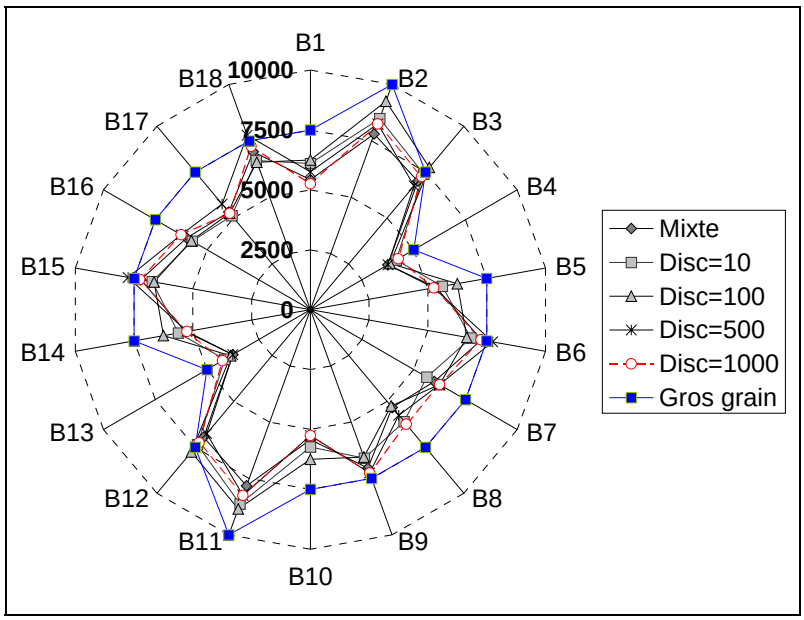


Figure 12 – Volumes pour des nombres d'équipements identiques selon le pas de discrétisation (atelier8)

Finalement, la discrétisation semble n'avoir qu'une influence très faible, voire inexistante, sur les variables à la solution optimale, dans la mesure où le pas de discrétisation varie raisonnablement. Par ailleurs, une deuxième restriction est que les variables aient un poids suffisant dans le critère d'optimisation comme c'est le cas pour les volumes des équipements discontinus. En effet, il est facile d'imaginer que les variables des étapes semi-continues pourraient adopter, avec un même pas de discrétisation, des valeurs très différentes sans occasionner une modification très importante du critère. L'étude de leur dispersion manque donc de réelle pertinence et il est difficile de conclure sur ces dernières.

Conclusion

Dans ce chapitre, une variante au modèle initial a été considérée. Les tailles des équipements, jusqu'alors représentées par des variables continues dans le modèle, ont été discrétisées. Cette démarche a été adoptée principalement pour satisfaire un souci de réalisme industriel, selon une gamme de tailles à « gros grain ». Puis, des discrétisations plus fines ont été prises en compte pour comparer les performances des trois méthodes d'optimisation utilisées jusqu'à présent, sur des problèmes complexes en variables purement entières.

Les résultats obtenus sont bien en accord avec les tendances prévues par l'analyse préalable du comportement des deux types de techniques d'optimisation, selon leur mode de

représentation des variables. Les méthodes de Programmation Mathématique sont lourdement pénalisées par l'introduction d'une combinatoire supplémentaire, générée par les volumes discrets de chaque étape opératoire. Au contraire, l'efficacité de l'AG est favorisée par un mode de représentation identique à celui employé pour le traitement des problèmes en variables mixtes et impliquant des chromosomes de taille réduite.

Ainsi, l'AG peut résoudre tous les problèmes envisagés ici, quel que soit le pas de discrétisation employé, avec une efficacité logiquement similaire (voire meilleure en termes de temps de calcul) à celle observée dans le chapitre précédent. Concernant les méthodes déterministes, *DICOPT++* est inefficace très rapidement. Il est remarquable que son point de rupture corresponde à une combinatoire proche de celle rencontrée pour les problèmes en variables mixtes (entre 10^8 et 10^9). *SBB* parvient à résoudre une partie des problèmes. Dans la majorité des cas, il fournit une solution faisable mais très éloignée de la meilleure solution de l'AG lorsque la combinatoire du problème (due indifféremment au pas de discrétisation ou à la complexité de l'atelier) augmente. Le temps de calcul pour prouver l'optimalité de la solution augmente très rapidement avec la finesse du pas et réduit les chances de conclure dans une grande partie des cas.

Finalement l'utilisation de variables discrètes pour représenter les volumes des équipements est une voie pertinente pour traiter le problème, tant du point de vue industriel (équipements disponibles selon une gamme définie, traduisant la flexibilité de l'atelier) que du point de vue numérique. Les performances obtenues par un AG sont du même niveau que celles obtenues en variables mixtes.

CHAPITRE 5

VALIDATION : ATELIER DE PRODUCTION DE PROTEINES

Ce dernier chapitre a pour objectif de vérifier la validité des résultats obtenus jusqu'à présent sur un problème concret modélisé de manière similaire aux exemples abordés précédemment. Il s'agit d'une application dans le domaine des bioprocédés, traitant de la conception d'un atelier multiproduit de production de protéines. Il a été traité dans plusieurs études successives par le même groupe d'auteurs [MON00]. Le principal intérêt du formalisme proposé réside dans l'intégration de modèles de performance associés à chaque étape opératoire de l'atelier [PIN01] [ASE00]. Ils impliquent la prise en compte des conditions opératoires de l'atelier. Certains paramètres clés sont alors considérés comme des variables du problème d'optimisation et interagissent avec les variables classiques de conception d'ateliers par le biais des modèles de performance évoqués. Pinto et al. [PIN01] résolvent ce problème avec le module *DICOPT++* de *GAMS*.

Cet exemple a également fourni un support de validation aux travaux de thèse présentés dans [DIE04], mais avec un formalisme différent, basé sur l'utilisation d'un simulateur à évènements discrets (SED). Ce dernier autorise en effet une approche séquentielle qui conduit finalement à contourner la contrainte du respect d'un horizon de temps fixe pour la production, en la remplaçant par une simulation impliquant des notions d'ordonnancement et de planification de la production. L'optimisation est alors nécessairement réalisée par une méthode stochastique, un algorithme génétique.

Cette démarche n'a pas été retenue dans nos travaux car elle n'est pas applicable aux méthodes de Programmation Mathématique employées. C'est donc l'approche « orientée équation » de [PIN01] qui est utilisée dans ce chapitre. Celui-ci se divise en deux parties. Le modèle et l'exemple étudié sont présentés brièvement dans un premier temps. Puis, deux séries de calculs (avec des tailles d'équipements continues puis discrètes) sont effectuées dans la seconde partie, dans l'optique de vérifier les conclusions des chapitres précédents.

1 – Présentation du nouveau problème

L'atelier de production de protéines utilisé comme exemple de validation peut simplement être considéré comme un problème similaire au précédent, auquel sont ajoutées des équations traduisant les modèles de performance des équipements. Cependant, la compréhension, même sommaire, de la physique du système étudié aide indéniablement à formuler le problème de manière pertinente pour une résolution plus efficace. Ainsi, le procédé de synthèse de protéines est présenté d'abord de manière générale, puis le nouveau modèle employé est décrit dans un second temps.

1.1 - Procédé de synthèse de protéines

1.1.1 – Généralités

L'exemple traité [MON00] est un atelier multiproduit conduisant à la fabrication de quatre produits. Parmi eux, deux sont des protéines thérapeutiques, l'insuline humaine et la vaccine pour l'hépatite B, tandis que les deux autres sont dotées de propriétés enzymatiques dont l'une est de qualité alimentaire, la chymosine, et l'autre utilisée comme détergent, la protéase cryophylique. L'atelier comporte huit étapes. Le procédé débute par une culture cellulaire ou fermentation, suivie de sept étapes de séparation : microfiltration, homogénéisation, ultrafiltration, extraction liquide-liquide, et chromatographie ([ASE00], [MON00]). Montagna et al. [MON00] ont « standardisé » le procédé dans le but d'obtenir un atelier générique. La figure 1 représente le schéma fonctionnel de l'atelier multiproduit pour la synthèse des quatre protéines dans le fermenteur par culture cellulaire, puis leur séparation des produits secondaires suivant un procédé spécifique.

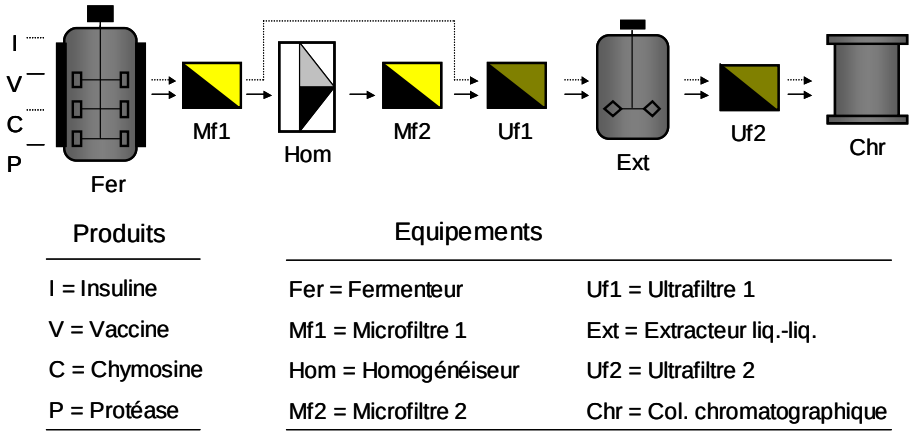


Figure 1 – Schéma fonctionnel du procédé de synthèse de protéines

La vaccine (*V*) et la protéase (*P*) sont des protéines intracellulaires, d'où une première étape de microfiltration mise en oeuvre pour concentrer la solution, laquelle est ensuite dirigée vers l'homogénéiseur pour la destruction des parois des cellules et la libération des protéines d'intérêt. La seconde microfiltration a pour rôle d'éliminer les débris de cellules de la suspension. La première étape d'ultrafiltration permet de concentrer la solution et ainsi de minimiser la taille de l'extracteur liquide-liquide. Dans l'étape d'extraction, le contrôle de la concentration du sel (NaCl) permet de transférer les protéines, dans un premier temps, de la phase aqueuse vers la phase polyéthylène-glycol (PEG), puis de les faire retourner vers une phase aqueuse de phosphate. Une seconde étape d'ultrafiltration concentre la solution avant la séparation par chromatographie.

L'insuline (*I*) et la chymosine (*C*) sont des produits extracellulaires. Ils percolent avec le perméat qui traverse la membrane de filtration du premier microfiltre. Pour réduire la quantité de produit contenue dans le rétentat, de l'eau est ajoutée à la suspension de cellules. Il n'est pas nécessaire de faire passer les produits extracellulaires à travers l'homogénéiseur et le deuxième microfiltre. L'ultrafiltration sert cependant pour concentrer la solution diluée avant l'extraction. Les étapes d'extraction, ultrafiltration et chromatographie sont communes à tous les produits.

1.1.2 – Modèles de performance

La motivation particulière de l'étude de [PIN01] concerne le développement de modèles simples d'opérations unitaires en vue de leur intégration au modèle initial de [MON00]. Le but est de mettre en évidence l'influence des conditions opératoires lors de la conception de l'atelier (taille et nombre d'équipements). Les paragraphes suivants proposent une description fonctionnelle des étapes de traitement sans rentrer dans le détail du modèle résultant. Seule la notion de rendement (η_j) de chaque étape est introduite à ce niveau de la description.

- ***Fermenteur***

  La première opération de l'atelier où les protéines sont élaborées est la culture cellulaire. Toutes les protéines sont produites par une levure, la *saccharomyces cerevisiae*. La réaction mise en jeu est la suivante :

  $$C_6H_{22}N_{11} + NH_3 + O_2 \rightarrow levure + CO_2 + H_2O$$

  Les réactifs nécessaires à la réaction, en plus d'une quantité initiale de cellules, sont des sucres, de l'azote ammoniacal et de l'oxygène. Les produits de la réaction sont les cellules avec les protéines d'intérêt, du dioxyde de carbone et de l'eau. Le rendement de cette étape est nécessairement égal à 1.

- ***Premier microfiltre***

  Cette opération implique trois équipements : un bac pour l'alimentation et destiné à recevoir le rétentat, le microfiltre proprement dit (membrane de microfiltration), et un bac pour le perméat (utilisé seulement par les produits extracellulaires : insuline et chymosine). Dans le cas de protéines intracellulaires, le but de cette unité est de concentrer la solution pour réduire la taille des équipements en aval (homogénéiseur, second microfiltre et premier ultrafiltre). Dans le cas des produits extracellulaires, les protéines sont séparées des cellules. La solution est d'abord concentrée jusqu'à sa

valeur maximale. Puis, de l'eau est ajoutée pour laver la solution et obtenir une récupération plus importante de produits.

Le rendement du premier microfiltre, défini comme la quantité du produit *i* en sortie par rapport à la quantité en entrée, est $\eta_{i,Mf1}=1$ pour les produits intracellulaires (vaccine et protéase), car il n'y a pas de perte de produits à travers la membrane. Dans le cas de produits extracellulaires (insuline et chymosine), seule une portion des protéines est récupérée après la filtration. Par conséquent, le rendement, inférieur à l'unité, doit être calculé.

- ***Homogénéiseur***

  La vaccine et la protéase nécessitent cette opération pour la rupture de la paroi des cellules, afin de libérer les protéines intracellulaires. Les passages successifs à travers l'homogénéiseur conduisent à une destruction totale des cellules : la fraction de protéines libérées est donc égale à 1. Le rendement est ainsi fonction du nombre de passages dans l'appareil.

- ***Second microfiltre***

  Les débris des cellules sont séparés des protéines, vaccine et protéase, à cette étape. Les opérations de filtration et diafiltration y sont également menées. La filtration est limitée à une réduction de 50 % de la taille initiale du lot pour éviter certains problèmes opérationnels dus à la concentration en matière solide. Comme pour le premier microfiltre, seule une portion des protéines est récupérée après la filtration, et par conséquent le rendement, inférieur à 1, doit être calculé.

- ***Premier ultrafiltre***

  Il s'agit de concentrer la solution à une concentration limite totale des protéines (estimée à 50 [kg/m^3]) pour réduire autant que possible la taille des équipements en aval, tout en évitant le risque de précipitation des protéines dans l'extracteur liquide-liquide (présence de NaCl). Le rendement de l'ultrafiltration est de 1, il n'y pas de perte de produits à cette étape, ce qui suppose que les pores de la membrane sont assez petits pour retenir la totalité des protéines.

- ***Extracteur liquide-liquide***

  Les travaux précédents [MON00] considèrent qu'à cette étape, les protéines autres que celles d'intérêt sont entraînées par le solvant. Le but est donc la séparation des protéines de la solution-mère contenant des matières premières non consommées et des impuretés. Le rendement de l'opération dépend du rapport de phases, polyéthylène-glycol (PEG) par rapport au phosphate.

- ***Second ultrafiltre***

  Le but de cette opération est, comme pour le premier ultrafiltre, d'augmenter la concentration totale des protéines à 50 [kg/m^3] pour réduire la taille de la colonne chromatographique. Le rendement est de nouveau égal à 1.

- ***Séparation chromatographique***

  On effectue par chromatographie la séparation finale de la protéine d'intérêt des autres protéines. Une utilisation de 50 % de la capacité a été supposée, conduisant à un rendement de $\eta_{i,Chr} = 0{,}95$.

De la description des étapes peut être déduite la liste des variables de procédé qui sont prises en compte par la suite dans le modèle. Elles sont au nombre de dix-huit :

- Concentration imposée pour chaque produit en sortie du fermenteur $X_{i,Fer}$, $i=\{I,C,V,P\}$.
- Concentration imposée pour chaque produit en sortie du premier microfiltre $X_{i,Mf1}$, $i=\{I,C,V,P\}$.
- Eau ajoutée dans le premier microfiltre, relativement à chaque produit extracellulaire $W_{i,Mf1}$, $i=\{I,C\}$.
- Eau ajoutée dans le deuxième microfiltre, relativement à chaque produit intracellulaire $W_{i,Mf2}$, $i=\{V,P\}$.
- Nombre de passages dans l'homogénéiseur, pour chaque produit intracellulaire NP_i, $i=\{V,P\}$.
- Rapport des phases dans l'extracteur liquide-liquide pour chaque produit R_i, $i=\{I,C,V,P\}$.

Ces variables représentent les conditions opératoires sur lesquelles il est possible de jouer pour améliorer les performances du procédé et qui interviennent dans la phase de conception. Leur contribution au modèle final est présentée par la suite.

1.2 – Formulation du modèle

Ce nouveau modèle est très proche de la formulation retenue dans le chapitre 2. Les différences notables sont à la fois d'ordre structurel (ce sont les plus minimes) et spécifiques aux modèles d'opérations unitaires. Tout d'abord, d'un point de vue structurel, le modèle prend uniquement en compte des étapes discontinues. Elles sont caractérisées comme précédemment par leur volume V_j, le nombre d'équipements *hors-phase* m_j, mais également par des équipements *en-phase* n_j. Ces équipements permettent d'éclater un lot en plusieurs lots plus petits au sein d'une même étape.

La variable n_j intervient dans l'équation (14) du modèle initial, qui s'écrit maintenant :

$$\forall i \in \{1,...,I\} \; ; \; B_i = \min_{j \in J} \left[\frac{V_j n_j}{S_{ij}} \right] \tag{1}$$

Elle apparaît aussi logiquement dans la formulation de la nouvelle fonction objectif :

$$Min\ Coût = \sum_{j=1}^{J} n_j m_j a_j V_j^{\alpha_j} \tag{2}$$

Par ailleurs, certains équipements sont composés de plusieurs appareils : c'est le cas pour (i) les microfiltres, comprenant deux bacs, l'un pour le rétentat et l'autre pour récupérer le perméat, ainsi que la membrane de filtration à proprement parler ; (ii) la configuration est identique pour les ultrafiltres, mis à part qu'ils ne comportent pas de bac de rétentat ; (iii) l'homogénéiseur compte un bac et l'équipement d'homogénéisation (caractérisé par une capacité Cap_{Hom}) ; (iv) enfin l'étape de chromatographie compte un bac ainsi que la colonne, où la protéine d'intérêt est séparée des autres. Les membranes des microfiltres et ultrafiltres ainsi que la capacité de l'homogénéiseur peuvent être assimilées en fait à des équipements semi-continus, imbriqués au sein d'appareils discontinus. Le tableau 1 récapitule les variables de conception pour le nouveau modèle. Il faut bien entendu y ajouter les nombres d'équipements en phase (n_j) et hors phase (m_j) pour chaque étape opératoire.

Noms	Description
V_{Fer}	Volume du fermenteur [m^3]
$V_{Mf1,ret}$	Volume du rétentat du premier microfiltre [m^3]
$V_{Mf1,per}$	Volume du perméat du premier microfiltre [m^3]
$S_{Mf1,fil}$	Surface de filtration du premier microfiltre [m^2]
V_{Hom}	Volume du bac de l'homogénéiseur [m^3]
Cap_{Hom}	Capacité de l'homogénéiseur [m^3/h]
$V_{Mf2,ret}$	Volume du rétentat du deuxième microfiltre [m^3]
$V_{Mf2,per}$	Volume du perméat du deuxième microfiltre [m^3]
$S_{Mf2,fil}$	Surface de filtration du deuxième microfiltre [m^2]
V_{Uf1}	Volume du rétentat du premier ultrafiltre [m^3]
$S_{Uf1,fil}$	Surface de filtration du premier ultrafiltre [m^2]
V_{Ext}	Volume de l'extracteur liquide-liquide [m^3]
V_{Uf2}	Volume du rétentat du deuxième ultrafiltre [m^3]
$S_{Uf2,fil}$	Surface de filtration du deuxième ultrafiltre [m^2]
V_{Chr}	Volume du bac de la colonne chromatographique [m^3]
$V_{Chr,col}$	Volume de la colonne chromatographique [m^3]

Tableau 1 – Récapitulatif des variables de conception

Ainsi, le vecteur global des variables d'optimisation s'écrit $X = [X_{conc}, X_{proc}]^T$, où X_{conc} est le vecteur des variables de conception et $X_{proc} = [X_{i,Fer}, X_{i,Mf1}, W_{i,Mf1}, W_{i,Mf2}, NP_i, R_i]^T$ le

vecteur des variables de procédé explicitées précédemment. Les modifications dues aux modèles de performances, fonctions des variables de procédé, consistent en :

- ✓ L'introduction d'une variable, interne au modèle, représentant le rendement η_j de chaque étape *j*, calculée selon l'équation générale :

$$\eta_j = f(X_{proc}), \qquad j=\{Fer,\ldots,Chr\} \tag{3}$$

- ✓ Le calcul des facteurs de taille, non plus fixes comme auparavant, mais dépendant cette fois des variables opératoires et des rendements :

$$S_{ij} = f(\eta_j, X_{proc}) , \qquad i=\{I,C,V,P\}, j=\{Fer,\ldots,Chr\} \tag{4}$$

- ✓ Enfin le calcul des temps opératoires est fonction non seulement des tailles de lots, mais aussi des variables opératoires, des rendements, et parfois des caractéristiques des équipements semi-continus de l'atelier (aires de filtration des filtres, capacité de l'homogénéiseur). L'équation (5) l'exprime sous une forme générale :

$$T_{ij} = f(\eta_j, X_{proc}, B_i, S_{Mf1,fil}, S_{Mf2,fil}, S_{Uf1,fil}, S_{Uf2,fil}, Cap_{Hom}) \tag{5}$$
$$i=\{I,C,V,P\}, j=\{Fer,\ldots,Chr\}$$

La forme développée des équations (3), (4) et (5) est disponible dans l'annexe 1 de ce chapitre. Les données de l'exemple (coefficients de coût, demandes, et bornes des variables) sont présentées en annexe 2.

Remarque : il est à noter qu'aucun bac de stockage n'a été considéré dans ce modèle. Leur prise en compte est pourtant proposée dans [MON00] et [PIN01]. Ni leur existence ni leur localisation ne sont fixées, mais liées à des variables binaires qui font partie des variables d'optimisation. L'étude présentée se restreint au cas (envisagé également dans [PIN01]) d'ateliers sans bacs de stockage intermédiaire.

2 – Résultats numériques

2.1 – Mode opératoire

Concernant la programmation dans l'environnement *GAMS*, le modèle explicité plus haut ne présente pas de difficulté particulière. La structure des programmes implémentés

auparavant est conservée, seules les équations relatives aux modèles de performance (calcul des rendements, facteurs de taille et temps opératoires) doivent y être intégrées. Les bornes sur ces variables internes au modèle sont ajustées de manière conséquente.

Relativement à l'AG, les conclusions tirées des études réalisées lors des chapitres précédents sont mises en application. Lorsque le problème est traité avec des variables de tailles d'équipements continues, ces dernières sont codées sur des loci à valeur réelle, tandis que les nombres d'équipements m_j ou n_j sont représentés sur un seul locus, adoptant la valeur de la variable codée. Par contre, lorsque les variables de tailles d'équipements sont discrètes, le codage par boîte de poids est employé tel qu'il avait utilisé auparavant. La discrétisation est effectuée en adoptant trois valeurs possibles de taille par gamme. Comme précédemment, chaque taille est représentée par une variable réduite : ainsi, une décimale, correspondant à quatre bits, est suffisante pour décrire les trois valeurs possibles. Cohabitent alors dans le chromosome trois types de codage : (i) des bits codant par la méthode de la boîte de poids les variables de taille d'équipement discrétisées ; (ii) des loci à valeur réelle codant les variables de procédé (toujours continues) ; (iii) des loci à valeur entière codant le nombre d'équipements (en-phase et hors-phase) par étape. Le croisement SBX peut être appliqué sans problème à la globalité du chromosome (des tests préalables ont vérifié son bon fonctionnement). Des procédures de mutation, déjà décrites auparavant, sont adaptées à chaque zone du chromosome.

Concernant les paramètres de fonctionnement et le mode de gestion des contraintes, ils ont tout d'abord été choisis par analogie avec des exemples de taille similaire parmi ceux traités dans les chapitres précédents. L'atelier compte 34 variables continues bornées (18 variables de procédé et 16 tailles caractéristiques) et 16 variables entières (pour huit étapes opératoires). Cela correspond à une taille de problème située entre celles des exemples 7 et 8.

Mais il est apparu rapidement que les taille de population et nombre de générations maximal correspondant à ces deux instances (200 et 500 individus évoluant sur 1000 et 500 générations pour les exemples 7 et 8 respectivement) ne sont pas suffisants pour trouver des solutions satisfaisantes. Les courbes d'évolution de la moyenne des critères et du meilleur individu au fil des générations indiquent que l'évolution n'est pas terminée. Cette difficulté peut être expliquée par les fortes interactions entre les variables. En effet, dans le modèle de [PIN01], ces dernières sont imbriquées de manière complexe, les variables de procédé intervenant dans le calcul des variables intermédiaires servant à la conception (tailles de lot, temps de traitement). Sur la fin de la recherche, l'AG éprouve ainsi plus de difficultés à tendre vers la solution et nécessite davantage de temps pour effectuer une minimisation efficace.

En conséquence, les paramètres retenus sont une taille de population de 500 individus et un nombre de générations de 1500. Par ailleurs, pour les raisons évoquées ci-dessus, la priorité est donnée à la diversification. Les taux de survie et de mutation sont donc pris tous deux égaux à 0,4. De même, le mode de gestion des contraintes s'est orienté vers une méthode de tournoi unique, avec une probabilité S_r valant 0,95.

Remarque : il a été cependant remarqué que la détermination aléatoire de solutions faisables n'est pas une tâche si ardue. En effet, il a été vérifié que le nombre supplémentaire d'évaluations du critère, induit par la création d'une population initiale faisable, n'est pas considérable par rapport au nombre global d'évaluations. Ainsi, la méthode d'élimination pourrait également être retenue. Mais il a semblé plus pertinent d'adopter tout de même la méthode de tournoi unique de manière à diversifier au mieux la population. Par ailleurs, cette technique permet, au contraire de l'élimination, de tirer parti d'informations fournies par des solutions légèrement infaisables.

2.2 – Calculs avec des tailles continues d'équipements

Dans cette partie, les volumes des équipements discontinus et tailles des appareils semi-continus (aire de la membranes des filtres et capacité de l'homogénéiseur) sont bornées. Les bornes choisies figurent dans l'annexe 2, qui regroupe les données des problèmes. Le nombre maximal d'équipements par étape est de 6 appareils hors-phase et 6 appareils en-phase (soit $M_{max} = N_{max} = 6$). Les solutions proposées dans [PIN01] ont été retrouvées au moyen des solveurs *DICOPT++* et *SBB*. L'AG parvient à déterminer une valeur relativement proche, à 0,85 % de l'optimum de *DICOPT++* et *SBB*. Les solutions données par les méthodes déterministes et par l'AG sont présentées respectivement dans les tableaux 2 et 3.

Les variables de conception indiquées sur les tableaux 2a et 3a sont globalement équivalentes, sauf pour la dernière étape, i.e. la chromatographie : trois équipements en-phase sont déterminés par l'AG au lieu de deux pour l'optimum des méthodes déterministes. C'est cette étape opératoire qui explique à elle seule la différence entre l'optimum de *SBB* et *DICOPT++* et le résultat de l'AG. Les variables de procédé montrent des différences elles-aussi minimes entre les deux solutions. Elles restent toutefois difficiles à interpréter, vu la complexité du modèle.

Relativement aux temps de calcul, les deux méthodes mathématiques convergent en trois secondes approximativement, tandis qu'une exécution de l'AG dure 42 secondes. Par ailleurs, les dispersions à 2 % et à 5 % des exécutions de l'AG valent respectivement 12 % et 61 %.

Cela signifie que l'AG est donc moins performant sur cet exemple : il ne garantit une solution très proche de l'optimum des méthodes déterministes qu'avec une faible probabilité, et avec un temps de calcul bien supérieur.

(a)

			Investissement = 1501434 [$]		
			Tailles d'éqts	Nb d'éqts hors-phase	Nb d'éqts en-phase
Fer	V_{Fer}	[m^3]	5,561	5	1
Mf1	$V_{Mf1,ret}$	[m^3]	5,561	1	1
	$V_{Mf1,pe}$	[m^3]	6,418		
	$A_{Mf1,fil}$	[m^2]	10,594		
Hom	V_{Hom}	[m^3]	0,859	1	1
	Cap_{Hom}	[$m^3.h^{-1}$]	0,64		
Mf2	$V_{Mf2,ret}$	[m^3]	0,859	1	1
	$V_{Mf2,pe}$	[m^3]	1,991		
	$A_{Mf2,fil}$	[m^2]	8,308		
Uf1	V_{Uf1}	[m^3]	6,418	1	1
	$A_{Uf1,fil}$	[m^2]	69,849		
Ext	V_{Ext}	[m^3]	1,43	1	1
Uf2	V_{Uf2}	[m^3]	0,986	1	1
	$A_{Uf2,fil}$	[m^2]	7,701		
Chr	V_{Chr}	[m^3]	0,187	1	2
	$V_{Chr,col}$	[m^3]	0,461		

(b)

		Insuline	Chymosine	Vaccine	Protéase
$X_{i,Fer}$	[$kg.m^{-3}$]	46,425	39,129	38,634	31,369
$X_{i,Mf1}$	[$kg.m^{-3}$]	250	246,45	250	250
$W_{i,Mf1}$	[$m^3.m^{-3}$]	0,34	0,1	-	-
$W_{i,Mf2}$	[$m^3.m^{-3}$]	-	-	1,817	1,748
NP_i	[]	-	-	2,278	2,305
R_i	[$m^3.m^{-3}$]	0,636	0,633	0,451	0,637

Tableau 2 – Solution des solveurs *SBB* et *DICOPT++*
(a) Variables de conception
(b) Variables de procédé

Les solveurs de *GAMS* sont ici nettement plus efficaces. Cela est peu étonnant dans la mesure où il a été déduit des chapitres précédents que les performances de l'AG sont essentiellement pénalisées par la présence de variables continues. Or, pour cet exemple en variables mixtes, le nombre et le rôle de ces dernières est prépondérant, ce qui défavorise l'AG.

2.3 – Calculs avec des tailles discrètes d'équipements

Les gammes de taille pour les différents équipements, qui sont maintenant des variables discrètes, sont disponibles en annexe 2. Cette fois, *DICOPT++* n'a pas été capable de fournir

un résultat, ce qui est relativement conforme aux tendances constatées dans le chapitre précédent. La combinatoire associée aux non-linéarités, encore plus sévères pour ce modèle, empêche le module représentant la méthodes des Approximations Externes de trouver une quelconque solution faisable. Par contre, *SBB* et l'AG trouvent des solutions (présentées dans les tableaux 4 et 5 respectivement) relativement différentes de la solution en tailles continues du tableau 2.

(a)			***Investissement = 1514216 [$]***		
			Tailles d'éqts	Nb d'éqts hors-phase	Nb d'éqts en-phase
Fer	V_{Fer}	[m^3]	5,735	5	1
Mf1	$V_{Mf1,ret}$	[m^3]	5,728	1	1
	$V_{Mf1,pe}$	[m^3]	5,910		
	$A_{Mf1,fil}$	[m^2]	10,081		
Hom	V_{Hom}	[m^3]	0,712	1	1
	Cap_{Hom}	[$m^3.h^{-1}$]	0,711		
Mf2	$V_{Mf2,ret}$	[m^3]	0,712	1	1
	$V_{Mf2,pe}$	[m^3]	1,751		
	$A_{Mf2,fil}$	[m^2]	9,480		
Uf1	V_{Uf1}	[m^3]	5,906	1	1
	$A_{Uf1,fil}$	[m^2]	66,156		
Ext	V_{Ext}	[m^3]	1,388	1	1
Uf2	V_{Uf2}	[m^3]	0,822	1	1
	$A_{Uf2,fil}$	[m^2]	7,125		
Chr	V_{Chr}	[m^3]	0,127	1	3
	$V_{Chr,col}$	[m^3]	0,310		

(b)		Insuline	Chymosine	Vaccine	Protéase
$X_{i,Fer}$	[$kg.m^{-3}$]	45,333	40,310	31,172	30,765
$X_{i,Mf1}$	[$kg.m^{-3}$]	249,99	248,97	249,98	249,99
$W_{i,Mf1}$	[$m^3.m^{-3}$]	0,213	0,1	-	-
$W_{i,Mf2}$	[$m^3.m^{-3}$]	-	-	1,817	1,748
NP_i	[]	-	-	2,350	2,515
R_i	[$m^3.m^{-3}$]	0,655	0,651	0,689	0,639

Tableau 3 – Solution de l'AG
a) Variables de conception
b) Variables de procédé

Il apparaît que l'AG trouve une solution plus éloignée (5,7 %) de celle de *SBB* que lorsque les variables de taille étaient continues. Il faut cependant nuancer ce commentaire négatif en rappelant que le problème comporte encore des variables mixtes (variables de conception discrètes et variables de procédé continues). Ainsi, une efficacité comparable à celle obtenue dans les problèmes en variables purement entières du chapitre précédent ne peut logiquement pas être atteinte.

(a)			***Investissement = 1583252 [$]***		
			Tailles d'éqts	Nb d'éqts hors-phase	Nb d'éqts en-phase
Fer	V_{Fer}	[m^3]	6	5	1
Mf1	$V_{Mf1,ret}$	[m^3]	6	1	1
	$V_{Mf1,pe}$	[m^3]	6		
	$A_{Mf1,fil}$	[m^2]	10		
Hom	V_{Hom}	[m^3]	1	2	1
	Cap_{Hom}	[$m^3.h^{-1}$]	0,25		
Mf2	$V_{Mf2,ret}$	[m^3]	1	1	1
	$V_{Mf2,pe}$	[m^3]	3		
	$A_{Mf2,fil}$	[m^2]	10		
Uf1	V_{Uf1}	[m^3]	6	1	1
	$A_{Uf1,fil}$	[m^2]	50		
Ext	V_{Ext}	[m^3]	1	1	2
Uf2	V_{Uf2}	[m^3]	1	1	1
	$A_{Uf2,fil}$	[m^2]	10		
Chr	V_{Chr}	[m^3]	1	1	2
	$V_{Chr,col}$	[m^3]	0,5		

(b)		Insuline	Chymosine	Vaccine	Protéase
$X_{i,Fer}$	[$kg.m^{-3}$]	51,301	46,744	36,517	32,014
$X_{i,Mf1}$	[$kg.m^{-3}$]	250	150	250	234,79
$W_{i,Mf1}$	[$m^3.m^{-3}$]	0,16	0,1	-	-
$W_{i,Mf2}$	[$m^3.m^{-3}$]	-	-	2,085	1,894
NP_i	[]	-	-	2,093	2,066
R_i	[$m^3.m^{-3}$]	0,617	0,661	0,460	0,637

Tableau 4 – Solution de *SBB*
a) Variables de conception
b) Variables de procédé

Ce comportement peut être expliqué par le nouveau mode opératoire : l'influence des conditions opératoires sur les variables de conception est cette fois-ci biaisée par la discrétisation de ces dernières. Ainsi, quand les tailles étaient des variables continues, une variation, même minime, des conditions opératoires entraînait une modification immédiate des variables de conception. En revanche, quand les tailles sont discrètes, l'influence des conditions opératoires est moins immédiate puisque les volumes d'équipements peuvent rester dans la même gamme de taille. Les méthodes de Programmation Mathématique ne sont au contraire pas pénalisées par cet aspect car elles fonctionnent en prenant des informations du second ordre (dérivées secondes), qui traduisent bien l'interaction exacte entre les différentes variables.

Ainsi, les valeurs des variables de procédé trouvées par l'AG ne sont pas celles de l'optimum déterminé par la méthode de Branch & Bound. La répercussion sur les variables de conception se fait essentiellement sentir au niveau du premier ultrafiltre, qui compte deux

équipements hors-phase dans la solution de l'AG et un seul dans celle de *SBB*. Cette différence explique à elle seule l'écart entre les deux coûts d'investissements résultants.

(a)

			Investissement = 1673624 [$]		
			Tailles d'éqts	Nb d'éqts hors-phase	Nb d'éqts en-phase
Fer	V_{Fer}	[m^3]	6	5	1
Mf1	$V_{Mf1,ret}$	[m^3]	6	1	1
	$V_{Mf1,pe}$	[m^3]	6		
	$A_{Mf1,fil}$	[m^2]	10		
Hom	V_{Hom}	[m^3]	1	1	1
	Cap_{Hom}	[$m^3.h^{-1}$]	0,5		
Mf2	$V_{Mf2,ret}$	[m^3]	1	1	1
	$V_{Mf2,pe}$	[m^3]	3		
	$A_{Mf2,fil}$	[m^2]	10		
Uf1	V_{Uf1}	[m^3]	6	2	1
	$A_{Uf1,fil}$	[m^2]	50		
Ext	V_{Ext}	[m^3]	1	1	2
Uf2	V_{Uf2}	[m^3]	1	1	1
	$A_{Uf2,fil}$	[m^2]	10		
Chr	V_{Chr}	[m^3]	1	1	2
	$V_{Chr,col}$	[m^3]	0,5		

(b)		Insuline	Chymosine	Vaccine	Protéase
$X_{i,Fer}$	[$kg.m^{-3}$]	44,199	41,035	30,002	32,794
$X_{i,Mf1}$	[$kg.m^{-3}$]	227,27	249,99	209,03	249,39
$W_{i,Mf1}$	[$m^3.m^{-3}$]	0,145	0,137	-	-
$W_{i,Mf2}$	[$m^3.m^{-3}$]	-	-	1,00	1,959
NP_i	[]	-	-	1,709	1,662
R_i	[$m^3.m^{-3}$]	0,528	0,470	0,612	0,504

Tableau 5 – Solution de l'AG
a) Variables de conception
b) Variables de procédé

Enfin, relativement aux temps de calcul, celui l'AG n'a pratiquement pas varié d'un mode opératoire à l'autre : le temps d'une exécution passe simplement de 42 à 53 secondes. Ceci s'explique par l'augmentation de la longueur du chromosom,e générée par le codage différent des variables de conception. Les dispersions à 2 % et 5 % valent respectivement 8 % et 47 %, ce qui révèle une répétabilité des résultats assez médiocre. Par contre, le temps de résolution nécessaire à *SBB* augmente sensiblement pour atteindre 5,3 [min]. Ce temps reste cependant d'autant plus raisonnable qu'il permet d'atteindre une solution optimale.

Conclusion

Dans ce chapitre, un exemple de conception d'ateliers discontinus quelque peu différent des précédents a été abordé. L'intégration de modèles de performance simplifiés introduit de nouvelles variables continues représentant certaines conditions opératoires du procédé. La résolution est effectuée avec les trois méthodes d'optimisation, en considérant les tailles d'équipements continues d'abord puis discrètes dans un deuxième temps.

Les résultats pour les variables continues sont en accord avec les calculs effectués jusqu'alors : si la supériorité des méthodes déterministes est indéniable, l'AG fournit également des solutions proches de l'optimum déterminé par ces dernières. Cependant, des valeurs élevées de la taille de population et du nombre de générations maximal doivent être retenues pour conduire à des solutions convenables. En effet, les variables continues de procédé, en interagissant fortement avec les variables de conception du modèle initial, génèrent une difficulté de résolution supplémentaire. Cette interaction n'est pas préjudiciable aux méthodes de Programmation Mathématique dans la mesure où elles utilisent des informations du second ordre pour guider leur recherche. Par contre, l'AG se base sur l'influence directe des paramètres, et le temps nécessaire pour constater une stabilisation du meilleur critère trouvé est plus long.

Ces tendances sont confirmées lors des calculs pour lesquels les tailles d'équipements sont considérées comme des variables discrètes. Tout d'abord, *DICOPT++* est mis en échec par la combinatoire et ne trouve aucune solution faisable, confirmant ainsi les résultats du chapitre précédent. Au contraire, *SBB* trouve un optimum en un temps toujours raisonnable. L'AG, enfin, donne une solution de qualité médiocre : les variables de procédé combinées à une discrétisation assez large de l'espace de recherche pénalisent fortement ses performances. Ce dernier exemple met donc en lumière le fait que les conclusions tirées dans le cadre de la formulation des chapitres précédents sont applicables mais dans une certaine mesure, et certainement pas extrapolables sans nuance pour des formulations différentes.

CHAPITRE 5
ANNEXE 1

MODELES DE PERFORMANCE

<table>
<tr><th colspan="2">Fermentation :</th></tr>
<tr><td>Protéines intracellulaires</td><td>Protéines extracellulaires</td></tr>
<tr><td>$S_{i,fer}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,fer}.\eta_{i,mf2}.\eta_{i,hom}.\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,fer}[h]=4+3,8.ln\left[\frac{0,35.X_{i,fer}}{\left(1-\frac{X_{i,fer}}{55}\right)}\right]$
$\eta_{i,fer}=1$</td><td>$S_{i,fer}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,fer}[h]=4+3,8.ln\left[\frac{0,35.X_{i,fer}}{\left(1-\frac{X_{i,fer}}{55}\right)}\right]$
$\eta_{i,fer}=1$</td></tr>
<tr><td colspan="2">$K_C^1=20,8\ K_V^1=31,2\ K_I^1=62,5\ K_P^1=15,6$</td></tr>
<tr><th colspan="2">Première Microfiltration :</th></tr>
<tr><td>Protéines intracellulaires</td><td>Protéines extracellulaires</td></tr>
<tr><td>$S_{i,mf1,ret}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,fer}.\eta_{i,mf2}.\eta_{i,hom}.\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,mf1}[h]=1,25+\left[\frac{K_i^2\left(1-\frac{X_{i,fer}}{X_{i,mf1}}\right)}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}\right].\frac{B_i}{A_{mf1}}$
$\eta_{i,mf1}=1$</td><td>$S_{i,mf1,ret}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}$
$S_{i,mf1,per}\left[\frac{m^3}{kg}\right]=\frac{K_i^1\left(1-\frac{X_{i,fer}}{X_{i,mf1}}+W_{i,mf1}\right)}{X_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,mf1}[h]=1,75+\left[\frac{K_i^2\left(1-\frac{X_{i,fer}}{X_{i,mf1}}+W_{i,mf1}\right)}{X_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}\right].\frac{B_i}{A_{mf1}}$
$\eta_{i,mf1}=1-\frac{X_{i,fer}}{X_{i,mf1}}exp\left(-\frac{W_{i,mf1}.X_{i,mf1}}{X_{i,fer}}\right)$</td></tr>
<tr><td colspan="2">$K_P^2=62,5\ K_V^2=125\ K_C^2=83,5\ K_I^2=250$</td></tr>
<tr><th colspan="2">Homogénéisation :</th></tr>
<tr><td>Protéines intracellulaires</td><td>Protéines extracellulaires</td></tr>
<tr><td>$S_{i,hom}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,hom}[h]=1,25+\left[\frac{K_i^3.NP_i}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}\right].\frac{B_i}{Cap_{hom}}$
$\eta_{i,hom}=\left[1-exp(-1,5.NP_i)\right].exp(-0,03.NP_i)$</td><td></td></tr>
<tr><td colspan="2">$K_P^3=12,5\ K_V^3=25$</td></tr>
</table>

Deuxième Microfiltration :

Protéines intracellulaires	Protéines extracellulaires
$S_{i,mf2,ret}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}$ $S_{i,mf2,per}\left[\frac{m^3}{kg}\right]=\frac{K_i^1(0,5+W_{i,mf2})}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}$ $T_{i,mf2}[h]=1,75+\left[\frac{K_i^4(0,5+W_{i,mf2})}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}\right].\frac{B_i}{A_{mf2}}$ $\eta_{i,mf2}=1-0,5.exp(-2.W_{i,mf2})$	

$K_P^4=125\ K_V^4=250$

Première Ultrafiltration :

Protéines intracellulaires	Protéines extracellulaires
$S_{i,uf1}\left[\frac{m^3}{kg}\right]=\frac{K_i^1}{X_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}$ $T_{i,uf1}[h]=1+\frac{K_i^5(0,5+W_{i,mf2})}{C_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}.\eta_{i,ext}.\eta_{i,chr}}.\left[1-\frac{0,24.C_{i,mf1}.\eta_{i,hom}.\eta_{i,mf2}}{50(0,5+W_{i,mf2})exp(-0,03.NP_i)}\right].\frac{B_i}{A_{uf1}}$ $\eta_{i,uf1}=1$	$S_{i,uf1}\left[\frac{m^3}{kg}\right]=\frac{K_i^1\left(1-\frac{X_{i,fer}}{X_{i,mf1}}+W_{i,mf1}\right)}{X_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}$ $T_{i,uf1}[h]=1+\frac{K_i^5\left(1-\frac{C_{i,fer}}{C_{i,mf1}}+W_{i,mf1}\right)}{C_{i,fer}.\eta_{i,mf1}.\eta_{i,ext}.\eta_{i,chr}}.\left[1-\frac{0,12.X_{i,fer}.\eta_{i,mf1}}{50.\left(1-\frac{C_{i,fer}}{C_{i,mf1}}+W_{i,mf1}\right)}\right].\frac{B_i}{A_{uf1}}$ $\eta_{i,uf1}=1$

$K_I^5=2500\ K_V^5=1250\ K_C^5=835\ K_P^5=625$

Extracteur Liquide-Liquide

Protéines intracellulaires	Protéines extracellulaires
$S_{i,ext}\left[\frac{m^3}{kg}\right]=\frac{K_i^6(1+R_i)}{exp(0,03.NP_i).\eta_{i,ext}.\eta_{i,chr}}$ $T_{i,ext}[h]=1,8$ $\eta_{i,ext}=\frac{K_i^7R_i}{(1+K_i^7R_i)\left(1+R_i.10^{K_i^8\left(\frac{7.R_i}{R_i+1}-5\right)}\right)}$	$S_{i,ext}\left[\frac{m^3}{kg}\right]=\frac{K_i^6(1+R_i)}{\eta_{i,ext}.\eta_{i,chr}}$ $T_{i,ext}[h]=1,8$ $\eta_{i,ext}=\frac{K_i^7R_i}{(1+K_i^7R_i)\left(1+R_i.10^{K_i^8\left(\frac{7.R_i}{R_i+1}-5\right)}\right)}$

$K_P^6=0,075\ K_V^6=0,15\ K_C^6=0,05\ K_I^6=0,15$

$K_P^7=25,1\ K_V^7=39,8\ K_C^7=50,1\ K_I^7=31,6$

$K_P^8=0,7\ K_V^8=0,8\ K_C^8=0,85\ K_I^8=0,75$

Deuxième Ultrafiltration :

Protéines intracellulaires	Protéines extracellulaires
$S_{i,uf2}\left[\frac{m^3}{kg}\right]=\frac{K_i^6}{exp(0,03.NP_i).\eta_{i,ext}.\eta_{i,chr}}$	$S_{i,uf2}\left[\frac{m^3}{kg}\right]=\frac{K_i^9}{\eta_{i,ext}.\eta_{i,chr}}$
$T_{i,uf2}[h]=0,3+\left[\frac{K_i^9-\eta_{i,ext}-\frac{K_i^{10}.R_i}{(R_i+1)^2}}{\eta_{i,ext}.\eta_{i,chr}.exp(-0,03.NP_i)}\right].\frac{B_i}{A_{uf2}}$	$T_{i,uf2}[h]=0,3+\left[\frac{K_i^9-\eta_{i,ext}\frac{K_i^{10}.R_i}{(R_i+1)^2}}{\eta_{i,ext}.\eta_{i,chr}}\right].\frac{B_i}{A_{uf2}}$
$\eta_{i,uf2}=1$	$\eta_{i,uf2}=1$

$$K_V^9=6\ K_C^9=2\ K_I^9=6\ K_P^9=3$$
$$K_P^{10}=2\ K_V^{10}=5\ K_C^{10}=1\ K_I^{10}=5$$

Séparation Chromatographique

Protéines intracellulaires	Protéines extracellulaires
$S_{i,chr}\left[\frac{m^3}{kg}\right]=\frac{0,025.\left[\eta_{i,ext}+\frac{K_i^{10}.R_i}{(R_i+1)^2}\right]}{\eta_{i,ext}.\eta_{i,chr}}$	$S_{i,chr}\left[\frac{m^3}{kg}\right]=\frac{0,025.\left[\eta_{i,ext}+\frac{K_i^{10}.R_i}{(R_i+1)^2}\right]}{\eta_{i,ext}.\eta_{i,chr}}$
$S_{i,chr,col}\left[\frac{m^3}{kg}\right]=\frac{0,1}{\eta_{i,chr}}$	$S_{i,chr,col}\left[\frac{m^3}{kg}\right]=\frac{0,1}{\eta_{i,chr}}$
$T_{i,chr}[h]=0,375+\left[\frac{0,0025\left[\eta_{i,ext}+\frac{K_i^{10}.R_i}{(R_i+1)^2}\right]}{\eta_{i,ext}.\eta_{i,chr}}\right]\frac{B_i}{V_{chr}}$	$T_{i,chr}[h]=0,375+\left[\frac{0,0025\left[\eta_{i,ext}+\frac{K_i^{10}.R_i}{(R_i+1)^2}\right]}{\eta_{i,ext}.\eta_{i,chr}.exp(-0,03.NP_i)}\right]\frac{B_i}{V_{chr}}$
$\eta_{i,chr}=0,95$	$\eta_{i,chr}=0,95$

CHAPITRE 5
ANNEXE 2

DONNEES DU PROBLEME

- ***Coefficients de coût***

Equipement	Taille	Coût
Fermenteur	$V\ [m^3]$	$63400.V^{0,6}$
Micro- et ultrafiltres	$V_{rétentat}\ [m^3]$	$5750.V^{0,6}$
	$V_{perméat}\ [m^3]$	$5750.V^{0,6}$
	$A_{filtre}\ [m^2]$	$2900.A^{0,85}$
Homogénéisateur	$V_{bac}\ [m^3]$	$5750.V^{0,6}$
	$Capacité\ [m^3/h]$	$12100.Cap_{Hom}^{0,75}$
Extracteur	$V_{extracteur}\ [m^3]$	$23100.V^{0,65}$
Chromatographie	$V_{bac}\ [m^3]$	$5750.V^{0,6}$
	$V_{colonne}\ [m^3]$	$360000.V^{0,995}$
Stockage	$V_{bac}\ [m^3]$	$5750.V^{0,6}$

- ***Productions souhaitées***

Produit	Production [kg/an]
(i) Insuline	1500
Vaccin	1000
Chymosine	3000
Protéase	6000

- ***Bornes des variables opératoires***

Variable	Borne inf.	Borne sup.
$X_{i,Fer}$ [kg.m^{-3}]	35	55
$X_{v,Fer}$ [kg.m^{-3}]	35	55
$X_{c,Fer}$ [kg.m^{-3}]	35	55
$X_{p,Fer}$ [kg.m^{-3}]	35	55
$X_{i,Mf1}$ [kg.m^{-3}]	150	250
$X_{v,Mf1}$ [kg.m^{-3}]	150	250
$X_{c,Mf1}$ [kg.m^{-3}]	150	250
$X_{p,Mf1}$ [kg.m^{-3}]	150	250
$W_{i,Mf1}$ [m^3.m^{-3}]	0,5	3,0
$W_{c,Mf1}$ [m^3.m^{-3}]	0,5	3,0
$NP_{v,Hom}$	1,0	3,0
$NP_{p,Hom}$	1,0	3,0
$W_{v,Mf2}$ [m^3.m^{-3}]	1,0	3,0
$W_{p,Mf2}$ [m^3.m^{-3}]	1,0	3,0
$R_{i,Ext}$ [m^3.m^{-3}]	0,05	1,5
$R_{v,Ext}$ [m^3.m^{-3}]	0,05	1,5
$R_{c,Ext}$ [m^3.m^{-3}]	0,05	1,5
$R_{p,Ext}$ [m^3.m^{-3}]	0,05	1,5

- ***Variables de conception***

Les bornes supérieure et inférieure apparaissent dans le tableau suivant pour les variables de conception, et plus particulièrement les tailles d'équipements (les nombres d'équipements en parallèle, en-phase et hors-phase, sont limités à 6, quelle que soit l'étape opératoire considérée). Pour les calculs en tailles discrètes, trois valeurs sont admissibles pour chaque étape : les valeurs inférieure et supérieure sont respectivement fixées aux bornes qui étaient utilisées pour les calculs en tailles d'équipements continues. La taille intermédiaire est donnée dans le tableau suivant

Equipement	Borne sup.	Taille intermédiaire	Borne inf.
V_{Fer} [m^3]	6	3	1
$V_{Mf1,ret}$ [m^3]	6	3	1
$A_{Mf1,fil}$ [m^2]	10	5	1
$V_{Mf1,per}$ [m^3]	6	3	1
V_{Hom} [m^3]	6	3	1
Cap_{Hom} [m^3/h]	0,5	0,25	0,1
$V_{Mf2,ret}$ [m^3]	6	3	1
$A_{Mf2,fil}$ [m^2]	10	5	1
$V_{Mf2,per}$ [m^3]	6	3	1
V_{Uf1} [m^3]	6	3	1
$A_{Uf1,fil}$ [m^2]	200	50	10
V_{Ext} [m^3]	6	3	1
V_{Uf2} [m^3]	6	3	1
$A_{Uf2,fil}$ [m^2]	20	10	1
V_{Chr} [m^3]	6	3	1
$V_{Chr,col}$ [m^3]	0,5	0,25	0,001
$V_{Chr,bac}$ [m^3]	6	3	1

CONCLUSIONS ET PERSPECTIVES

Ces travaux de doctorat avaient pour objectif majeur de proposer une méthodologie de résolution pour les problèmes d'optimisation en variables mixtes rencontrés en Génie des Procédés. Au moment de conclure ce mémoire, des éléments de réponse peuvent être apportés et des idées directrices émises pour guider vers le choix d'une technique d'optimisation pertinente, lors du traitement d'un cas nouveau.

Rappelons que l'étude exposée dans ces travaux a retenu comme support le problème de conception d'ateliers discontinus, de type MINLP, et présentant un fort aspect combinatoire. La méthodologie adoptée présente cependant l'intérêt d'être applicable dans le cadre plus général de conception en Génie des Procédés (réseaux d'échangeurs de chaleur, de réacteurs, de matière…). Nous avons ainsi évalué l'efficacité de méthodes issues des deux grandes classes, i.e. déterministe et stochastique, sur une formulation nécessairement analytique. Un jeu d'exemples a été construit pour prendre en compte la complexité de l'instance abordée et pousser à leur extrême les performances des méthodes considérées.

Concernant les méthodes déterministes, le choix de l'environnement *GAMS*, classiquement utilisé en Génie des Procédés, a conduit à implanter l'utilisation de ce logiciel dans l'équipe. Les algorithmes de Programmation Mathématique mis en œuvre au sein des solveurs *DICOPT++* et *SBB* représentent respectivement la méthode des Approximations Externes et celle de Branch & Bound. La seconde classe de méthodes est représentée par un Algorithme Génétique, ayant déjà fait ses preuves pour la résolution de ce type de problèmes [DIE04] [DED01] [BER99].

L'objectif du problème de conception d'un atelier discontinu est de déterminer la configuration minimisant un critère économique basé sur le coût d'investissement pour l'ensemble des équipements, tout en respectant une contrainte de production en un temps imparti. Ce formalisme sous-tend un second axe de recherche, à savoir la détermination des procédures internes de la méthode stochastique, qui lui permettront d'appréhender de manière efficace la nature mixte des variables et la présence d'une contrainte. Ces enjeux pour le bon fonctionnement de toute métaheuristique constituent un thème récurrent de recherche, tant pour des applications formulées analytiquement comme c'est le cas ici, que pour des cas typiques où l'évaluation du critère se fait de manière numérique, au moyen d'un simulateur.

La modification d'un AG déjà existant a été effectuée pour implémenter, de manière entièrement générique, la possibilité de choisir le mode de gestion des contraintes ou la forme de codage, et de fournir simplement les données propres à chaque instance. Ces travaux,

menés dans l'optique d'une éventuelle réutilisation, ont constitué une réalisation notable de l'étude (voir figure 1).

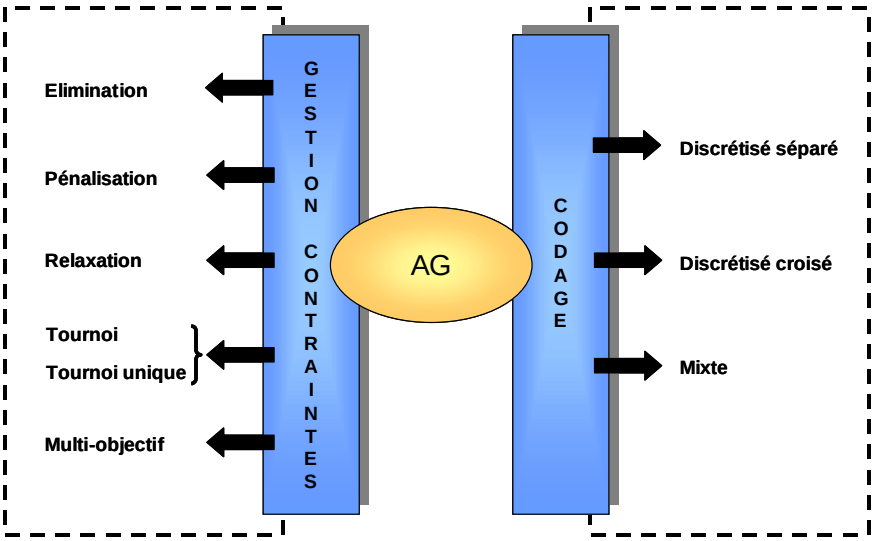


Figure 1 – Enjeux de l'Algorithme Génétique

Les résultats numériques obtenus sur le traitement du jeu d'exemples ont confirmé qu'un point clé pour le choix d'une méthode de résolution réside dans la formulation du modèle. Si celui-ci est mis en œuvre au sein d'un simulateur, seule une méthode stochastique est viable. Mais dans le cas abordé ici d'un problème en variables mixtes formulé de manière analytique, la classe déterministe est définitivement la plus puissante.

En effet, le module *SBB* fait preuve d'une efficacité remarquable, quelle que soit la taille de l'exemple traité. Des solutions optimales sont fournies en des temps de calcul augmentant certes de manière exponentielle, mais restant acceptables si l'on tient compte de la qualité des résultats.

En revanche, la conclusion est à nuancer pour la méthode des Approximations Externes. Tout d'abord, elle se laisse piéger sur un optimum local pour un exemple apparemment simple. Par ailleurs, dès que les tailles de problèmes deviennent plus importantes, l'accumulation de programmes continus infaisables conduit à l'échec du solveur *DICOPT++*.

Les calculs effectués avec l'AG conduisent tout d'abord à considérer un critère de répétabilité original qui, prenant en compte sa nature stochastique, permet l'évaluation de la dispersion des résultats des différentes exécutions autour de la meilleure solution trouvée.

Dans un premier temps, l'AG montre des performances globales satisfaisantes, bien qu'il reste lui aussi piégé dans des optima locaux et que son temps de calcul semble restrictif assez

rapidement. Mais, la comparaison avec les solutions optimales fournies par *SBB* permet d'améliorer son efficacité en déterminant les procédures appropriées de codage et de gestion des contraintes. Il apparaît ainsi que la méthode d'élimination des individus infaisables retenue dès le début des calculs promet une bonne qualité des résultats sous réserve d'une complexité raisonnable du problème. Dans le cas contraire, les temps de calculs prohibitifs orientent alors vers un mode de gestion par tournoi unique, basé sur des règles de dominance inspirées de l'optimisation multiobjectif. Il a été cependant démontré que, pour des problèmes de taille vraiment critique, cette méthode est quelque peu pénalisée par la difficulté à localiser une première solution faisable. Des méthodes alternatives, telles que celle consistant à effectuer une réflexion de la valeur de la fonction objectif par rapport à la frontière de faisabilité [TSU01], mériteraient d'être envisagées pour améliorer les performances de l'AG.

Puis, des considérations logiques sur le mode de représentation des variables continues ont amené à modifier leur codage. Celui-ci, basé initialement sur une discrétisation, s'oriente vers l'utilisation de chromosomes mixtes, mélangeant loci à valeurs entière et réelle et signifiant la mise en œuvre d'opérateurs génétiques adaptés. Les rectifications apportées permettent non seulement d'améliorer le temps de résolution de l'AG mais surtout la qualité des solutions obtenues en termes de répétabilité. Il n'en demeure pas moins que l'écart à la solution optimale déterminée par *SBB* reste significativement élevé pour les instances de plus grande taille.

Cependant, des éléments permettent de nuancer la supériorité a priori sans appel de la méthode de Branch & Bound. Tout d'abord, un effort important de formulation est nécessaire au bon fonctionnement des méthodes de Programmation Mathématique, induisant un temps de développement difficile à quantifier, mais probablement assez important. L'étude réalisée pour convexifier le modèle étudié, dans la mesure du possible, en apporte la preuve. Au contraire, la méthode stochastique est généralement aussi simple à comprendre qu'à implémenter.

Par ailleurs, des calculs supplémentaires ont été effectués en considérant les mêmes problèmes, avec des variables purement entières cette fois. L'AG est alors capable de fournir des résultats avec une efficacité équivalente à celle observée lors des calculs en variables mixtes. Par contre, les deux méthodes déterministes sont mises en échec par la combinatoire générée par les variables discrètes.

La méthode stochastique montre alors à quel point elle est adaptée pour répondre à des problèmes d'ingénierie concrets et réalistes. Elle ne peut, certes, vérifier l'optimalité du résultat. Cependant, elle est facilement adaptable à une large gamme d'applications et capable

d'atteindre un « objectif », i.e. de fournir un ensemble de solutions faisables, de bonne qualité, à un praticien libre alors de choisir quelle solution lui semble la plus intéressante. Cette réflexion peut être par exemple illustrée par les travaux de thèse de A. Aguilar Lasserre, orientés vers la représentation de l'imprécision sur la demande grâce à des concepts propres à l'arithmétique floue [AGU05].

Finalement, la démarche est validée sur un exemple de conception d'un atelier de synthèse de protéines, formulé d'une manière similaire à la précédente. Cet exemple constitue un pas supplémentaire vers la finesse de la représentation, par l'intégration de la connaissance du procédé. Cette approche prometteuse est réalisée au moyen de variables continues additionnelles, traduisant des conditions opératoires essentielles de l'atelier. Les résultats numériques obtenus au moyen des trois techniques affirment la supériorité des méthodes mathématiques (et plus particulièrement de *SBB*), qui trouvent une solution optimale en des temps plus que raisonnables. L'AG fait preuve d'une efficacité satisfaisante, tout en restant inférieur.

Ainsi, dans les limites de la formulation envisagée, la supériorité des méthodes de Programmation Mathématique semble établie. Il n'est évidemment pas possible de généraliser ces conclusions à n'importe quelle application, mais elles semblent vérifiables pour cette classe de problèmes, englobant non seulement la conception d'ateliers discontinus mais aussi celle de réseaux de distribution de gaz (thèse en cours de Firooz Tabkhi), de réseaux d'échangeurs de matière [SHA02], ou de réseaux de réacteurs, permettant l'identification de modèles [HOC05].

Différentes perspectives peuvent être également tracées.

- ***Hybridation***

Tout d'abord, l'étude a permis de mettre en évidence que l'écueil principal rencontré par la méthode stochastique est dû à la présence de variables continues, alors qu'au contraire, c'est la combinatoire générée par les variables entières qui pénalise les performances des méthodes déterministes. Ces remarques confortent ainsi l'approche déjà développée consistant à hybrider une métaheuristique et une méthode déterministe pour tirer parti de leur puissance respective. La première traiterait un problème maître portant sur les variables discrètes, tandis que la seconde résoudrait les problèmes esclaves continus. D'autres modes de mise en œuvre de l'hybridation ont déjà été proposés, simplement en juxtaposant ou au contraire en imbriquant plus intimement des méthodes des deux classes. Toutefois, plusieurs études [MON04] ont montré que, pour les méthodes hybrides, la gestion des problèmes

continus (initialisation des variables, variation du nombre de contraintes au cours des itérations) générés à partir des résultats des métaheuristiques posait de très sérieux problèmes et pouvait rendre cette approche difficile à mettre en œuvre.

- ***Gestion des contraintes***

Par ailleurs, il semble que l'étude sur la gestion des contraintes au sein de l'AG gagnerait à être approfondie, tout d'abord dans le sens évoqué précédemment : la difficulté à trouver une première solution faisable pour des problèmes sévèrement contraints doit motiver des recherches tendant à améliorer les performances de la méthode de tournoi unique employée ici. Par ailleurs, même s'il est théoriquement abordable de la même manière, le cas d'un problème comportant un plus grand nombre de contraintes mériterait d'être testé.

- ***Parallélisation / Grilles de calcul***

Enfin, il a été souligné qu'un obstacle à la performance des méthodes, notamment déterministes, est le temps de calcul, prohibitif sur les exemples de grande taille. Ainsi, l'augmentation de la capacité des calculateurs permet de considérer des problèmes qui auraient été inabordables quelques décennies ou même quelques années auparavant. Outre l'amélioration des performances des processeurs en puissance pure, leur combinaison a pu apporter des solutions pour contourner des problèmes de temps de calcul trop élevés. Des techniques de parallélisation sont en cours de développement [MIG03]. Elles permettent de mener de front une même recherche, chaque processeur s'informant des progrès effectués par les autres (collaboration de méthodes de recherche locale, initialisées en différents points de l'espace de recherche, par exemple) ; ou bien alors de découpler un problème en sous-problèmes impliquant un constant va-et-vient entre les processeurs ; ou bien enfin, dans le cas particulier où la recherche se base sur une population d'individus (algorithmes évolutifs, typiquement), de découper la population de solutions en sous-populations, traitées chacune par un processeur, et de mettre en commun l'évolution observée à chaque itération. Les projets de gridification [LAF02] [GRO05] sont également d'actualité, proposant du temps de calcul sur des clusters régionaux ou nationaux, dans l'objectif d'offrir de nouvelles options de calcul scientifique. Nous avons vu l'intérêt de cette approche à travers notre contribution au projet réseau Grid'5000 dans le cadre du BQR inter-établissements INP-INSA. Le prototype développé concernant l'algorithme génétique a pu être « gridifié » (i.e. porté sur la grille de calcul Grid'5000, exécuté à distance, les transferts de données et les aspects de visualisation étant transparents à l'utilisateur). L'exécution parallèle des différents essais pour un même jeu de paramètres de l'AG a ainsi conduit à une réduction drastique des temps de calcul (le rapport est supérieur à 20).

Mais outre ces améliorations et perspectives possibles, ce mémoire peut être conclu sur l'espoir d'avoir apporté des éléments de réponse pour aborder de manière rigoureuse la phase de résolution d'un problème d'optimisation non-linéaire en variables mixtes et contribué à l'amélioration de la connaissance des problèmes numériques en Génie des Procédés.

NOMENCLATURE

$A_{Mf1,fil}$: surface de la membrane de filtration pour le premier microfiltre [m^2].

$A_{Mf2,fil}$: surface de la membrane de filtration pour le deuxième microfiltre [m^2].

$A_{Uf1,fil}$: surface de la membrane de filtration pour le premier utrafiltre [m^2].

$A_{Uf2,fil}$: surface de la membrane de filtration pour le deuxième utrafiltre [m^2].

a_j : coefficient de coût pour l'étape discontinue *j*.

b_k : coefficient de coût pour l'étape semi-continue *k*.

B_{is} : taille caractéristique d'un lot de produit *i* dans le sous-procédé *s* [kg].

c_s : coefficient de coût pour les stockages intermédiaires.

Cap_{Hom} : capacité de l'homogénéiseur.

Coût : fonction objectif à minimiser, i.e. coût d'investissement pour les équipements de l'atelier.

D_{ik} : capacité de traitement du produit *i* dans l'étape semi-continue *k* [$L.kg^{-1}.h^{-1}$].

d_{ij} : coefficient puissance pour le calcul du temps opératoire du produit *i* dans l'étape discontinue *j*.

f(x,y) : forme générale de la fonction objectif du problème d'optimisation.

f_L : borne inférieure de la fonction *f* pour le problème d'optimisation sans contraintes.

f_U : borne supérieure de la fonction *f* pour le problème d'optimisation sans contraintes.

g_{ij} : coefficient pour le calcul du temps opératoire du produit *i* dans l'étape discontinue *j*.

g(x,y) : contraintes inégalité du problème d'optimisation.

H : horizon de temps [h].

H_i : temps de production du produit *i* [h].

h(x,y) : contraintes égalité du problème d'optimisation.

i : indice pour les produits.

I : nombre total de produits.

j : indice pour les étapes discontinues.

J : ensemble des étapes discontinues.

J_s : ensemble des étapes discontinues dans le sous-procédé *s*.

k : indice pour les étapes semi-continues.

K : ensemble des étapes semi-continues.

K_t : ensemble des étapes semi-continues dans le sous-train *t*.

K_s : ensemble des étapes semi-continues dans le sous-procédé *s*.

m_j : nombre d'unités identiques en parallèle dans l'étape discontinue *j*.

n_k : nombre d'unités identiques en parallèle dans l'étape semi-continue *k*.

N_{max} : nombre maximal d'équipements en parallèle dans un étage (*B* ou *SC*).

N_{min} : nombre minimal d'équipements en parallèle dans un étage (*B* ou *SC*).

NP_i : nombre de passages dans l'homogénéiseur pour les produits intracellulaires.

p_{ij} : temps opératoire du produit *i* dans l'étape discontinue *j* [h].

p_{ij}^{0} : constante pour le calcul du temps opératoire du produit *i* dans l'étape discontinue *j*.

$Prod_i$: productivité en produit *i* sur le procédé global.

$Prodloc_{is}$: productivité locale en produit *i* dans le sous-procédé *s*.

Q_i : demande en produit *i* [kg].

R_k : taux opératoire pour l'unité semi-continue *k* [h].

R_i : rapport de phase dans l'extracteur liquide-liquide pour le produit *i*.

R_{max} : taille maximale acceptable pour l'unité semi-continue *k* [$L.h^{-1}$].

R_{min} : taille minimale acceptable pour l'unité semi-continue *k* [$L.h^{-1}$].

S : nombre total de sous-procédés.

S_{ij} : facteur de taille pour le produit *i* dans l'étape discontinue *j* [$L.kg^{-1}$].

S_{is} : facteur de taille pour le produit *i* dans le stockage intermédiaire *k* [$L.kg^{-1}$].

t : indice de sous-train semi-continu.

T : ensemble des sous-trains semi-continus.

T_{ij} : temps de cycle du produit *i* dans l'étape discontinue *j*.

T_{is}^{L} : temps de cycle limitant du produit *i* dans le sous-procédé *s*.

V_j : taille de l'étape discontinue *j* [L].

V_{max} : taille maximale acceptable pour l'étape discontinue *j* [L].

V_{min} : taille minimale acceptable pour l'étape discontinue *j* [L].

V_s : taille du stockage intermédiaire [L].

$W_{i,Mf1}$: eau ajoutée dans le premier microfiltre pour les produits extracellulaires [$m^3.m^{-3}$].

$W_{i,Mf2}$: eau ajoutée dans le deuxième microfiltre pour les produits intracellulaires [$m^3.m^{-3}$].

x : vecteur des variables continues.

X : ensemble fermé de définition de *x*.

$X_{i,Fer}$: concentration en produit *i* dans le fermenteur [$kg.m^{-3}$].

$X_{i,Mf1}$: concentration en produit *i* dans le premier microfiltre [$kg.m^{-3}$].

X_{conc} : variables de conception de l'atelier.

X_{proc} : variables des conditions opératoires du procédé.

y : vecteur des variables discrètes.

Y : ensemble fermé de définition de *y*.

Z : critère à minimiser.

Z_L : borne inférieure (infaisable) du critère du problème d'optimisation.

Z_U : borne supérieure (faisable) du critère du problème d'optimisation.

α_j : exposant de coût pour l'étape discontinue *j*.

β_k : exposant de coût pour l'étape semi-continue *k*.

γ_s : exposant de coût pour le stockage intermédiaire.

η_j : rendement de l'étape opératoire *j*.

θ_{ik} : temps opératoire du produit *i* dans l'étape semi-continue *k* [h].

Θ_{it} : temps opératoire du produit *i* dans le sous-train semi-continu *t* [h].

REFERENCES BIBLIOGRAPHIQUES

[ABA69] Abadie J., Carpentier J., *Generalization of the Wolfe reduced gradient method to the case of* non-*linear constraints*. Optimization, R. Fletcher (éd.), Academic Press, New York, 1969, 37-47.

[ADJ00] Adjiman C.S., Androulakis I.P., Floudas C.A., *Global optimization of mixed-integer nonlinear problems*. AIChE Journal, 2000, 46, 1769-1797.

[AGU05] Aguilar Lasserre A., Azzaro-Pantel C., Domenech S., Pibouleau L., *Modélisation des imprécisions de la demande en conception optimale multicritère d'ateliers discontinus*. Communication : SFGP, Toulouse (France), 20-22 septembre 2005.

[AND01] André J., Siarry P., Dognon P., *An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization*. Advances in Engineering Software, 2001, 32 ,49-60.

[ARU05] Arumugam M.S., Rao M.V.C., Palaniappan R., *New hybrid genetic operators for real coded genetic algorithm to compute optimal control of a class of hybrid systems*. Applied Soft Computing, 2005, 6, 38-52.

[ASE00] Asenjo J.A., Montagna J.M., Vecchietti A.R., Iribarren O.A., Pinto J.M., Strategies for the simultaneous optimization of the structure and the process variables of a protein production plant. Computers and Chemical Engineering, 2000, 24, 2277-2290.

[AZZ05] Azzaro-Pantel C., Ordonnancement et conception d'ateliers discontinus. Techniques de l'Ingénieur, AG 3010, 1-12.

[BÄC97] Bäck T., Hammel U., Schwefel H.P., Evolutionary computation : comments on the history and current state. IEEE Transactions on Evolutionary Computation, 1997, 1, 3-17.

[BAU97] Baudet P., Ordonnancement à court-terme d'un atelier discontinu de chimie : cas du fonctonnement job-shop. Thèse de doctorat, INP ENSIGC, Toulouse (France), 1997.

[BEA93] Beasley D., Bull D.R., Martin R.R., An overview of genetic algorithms : Part 2, research topics. University Computing, 1993, 15, 170-181.

[BER99] Bernal Haro L., Conception d'ateliers discontinus multiobjectifs de chimie fine par un algorithme génétique. Thèse de doctorat, INP ENSIGC, Toulouse (France), 1999.

[BIE04] Biegler L.T., Grossmann I.E., Retrospective on optimization. Computers and Chemical Engineering, 2004, 28, 1169-1192.

[BIX00] Bixby R.E., Fenelon M, Gu Z., Rothberg E., Wunderling R., MIP : theory and practice – Closing the gap. System Modelling and Optimization : Methods, Theory and Applications. Vol. 174 of IFIP (International Federation for Informatic Processing). Kluwer Academic Publishers, Boston, 2000, 19-49.

[BRO98] Brooke A., Kendrick D., Meeraus A., Raman, R., GAMS User's Guide. GAMS Development Corporation, 1998.

[BUS03] Bussieck M.R., Pruessner A., Mixed-integer nonlinear programming. SIAG/OPT Newsletter, Views & News, 2003, 14.

[CHO06] Chootinan P., Chen A., Constraint handling in genetic algorithmsising a gradient-based repair method. Computers and Operations Research, 2006, 8, 2263-2281.

[COE02a] Coello Coello C.A., Theoretical and numerical constraint-handling techniques used with evolutionary algorithms : a survey of the state of the art. Computers Methods Applied in Mechanical Engineering, 2002, 191, 1245-1287.

[COE02b] Coello Coello C.A., Use of a self-adaptive penalty approach for engineering optimization problems. Computers in Industry, 2002, 41, 113-127.

[COE02c] Coello Coello C.A., Mezura Montes E., Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Advanced Engineering Informatics, 2002, 16, 193-203.

[COL02] Collette Y., Siarry P., Optimisation multiobjectif. Eyrolles, ISBN : 2-212-11168-1, 2002.

[COS01] Costa L., Oliveira P., Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems. Computers and Chemical Engineering, 2001, 25, 257-266.

[DEB95] Deb K., Agrawal R.B., Simulated binary crossover for continuous search space. Complex Systems, 1995, 9, 115-148.

[DEB96] Deb K., Goyal M., A combined adaptive search (GeneAS) for engineering design. Computer Science and Informatics, 1996, 26, 30-45.

[DEB00] Deb K., An efficient constraint handling method for genetic algorithms. Computers Methods Applied in Mechanical Engineering, 2000, 186, 311-338.

[DED01] Dedieu S., Algorithme génétique multicritère : conception et remodelage d'ateliers de chimie fine. Thèse de doctorat, INP ENSIGC, Toulouse (France), 2001.

[DIE04] Dietz A., Optimisation multicritère pour la conception d'ateliers discontinus multiproduits : aspects économique et environnemental. Thèse de doctorat, INP ENSIACET, Toulouse (France), 2004.

[DUR86] Duran M.A., Grossmann I.E., An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming, 1986, 36, 307-339.

[DRU85] Drud A., A GRG code for large sparse dynamic nonlinear optimization problems. Mathematical Programming, 1985, 31, 153-191.

[ELL02] Elliott L., Ingham D.B., Kyne A.G., Mera N.S., Pourkashanian M., Wilson C.W., A real coded genetic algorithm for the optimisation of reaction rate parameters for chemical kinetic modelling in a perfectly stirred reactor. Communnication : Genetic and Evolutionary Computation Conference, Late Breaking Papers, GECCO, New York (Etats unis), 2002.

[ESP89] Espuña A., Lázaro M., Martínez J.M., Puigjaner L., An efficient and simplified solution to the predesign problem of multiproduct plants. Computers and Chemical Engineering, 1989, 13, 163-174.

[EPP97] Epperly T.G.W., Ierapetritou M.G., Pistikopoulos E.N., On the global and effficient solution of stochastic batch plant design problems. Computers and Chemical Engineering, 1997, 21, 1411-1431.

[FLO95] Floudas C.A., Non-linear and mixed-integer optimization, Fundamentals and application. Oxford University Press, 1995.

[FLO05] Floudas C.A., Akrotirianakis I.G., Caratzoulas S., Meyer C.A., Kallrath J., Global optimization in the 21st century : advances and challenges. Computers and Chemical Engineering, 2005, 29, 1185-1202.

[GEO72] Geoffrion A.M., Generalized benders decomposition. Journal of Optimization Theory and Applications, 1972, 10, 237-260.

[GOM58] Gomory R.E., Outline of an algorithm for integer solutions to linear programs. Bulletin of American Mathematics Society, 1958, 64, 275-299.

[GOL89] Golberg, D.E., Genetic algorithms in search, optimization and machine learning. Addison-Wesley Publishing Company Inc., MA, 1989.

[GRO79] Grossmann I.E, Sargent R.W.H., Optimum design of multipurpose chemical plants. Industrial Engineering and Chemical Process Design and Development, 1979, 18, 343-348.

[GRO02] Grossmann I.E., Review of nonlinear mixed-integer and disjunctive programming techniques. Optimization and engineering, 2002, 3, 227-252.

[GRO05] Grossmann I.E., Enterprise-wide optimization : a new frontier in process systems engineering. AIChE Journal, 2005, 51, 1846-1857.

[GUP85] Gupta O.K., Ravindran V., Branch and bound experiments in convex nonlinear integer programming. Management Science, 1985, 31 ,1533-1546.

[HAD97] Hadj-Alouane A.B., Bean J.C., A genetic algorithm for the multiple-choice integer program. Operation Research, 1997, 45, 92-101.

[HAO99] Hao J.-K., Galinier P., Habib M., Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contrainte. Revue d'Intelligence Artificielle, 1999, 11-48.

[HED06] Hedar A.R., Fukushima M., Tabu serarch directed by direct search methods for non-linear global optimization. European Journal of European Research, 2006, 170, 329-349.

[HOC05] Hocine S., Pibouleau L., Azzaro-Panterl C., Domenech S., Modelling a fluidized bed for water treatment via GAMS package. Communication : ICheaP-7, 7th Italian Conference on Chemical and Process Engineering, Giardini Naxos (Italie), 15-18 mai 2005.

[HOL75] Holland J., Adaptation in natural and artificial systems, MIT Press, Cambridge, Massachussets, 1975.

[HOM94] Homaifar A., Lai S.H.Y., Qi X., Constrained optimization via genetic algorithms. Simulation, 1994, 62, 242-254.

[JAIN05] Jain A., Srinivasalu S., Determination of an optimal unit pulse response function using real-coded genetic algorithms. Journal of Hydrology, 2005, 303, 199-214.

[JOH00] Johnson E.L., Nemhauser G.L., Savelsbergh M.W.P., Progress in linear programming-based algorithms for integer programming : an exposition. Journal on Computing, 2000, 12, 2-22.

[KIR82] Kirkpatrick S., Gelatt Jr. C.D., Vecchi M.P., Optimization by simulated annealing. IBM Research Report RC9355, 1982.

[KNO82] Knopf F.C., Okos M.R., Reklaitis G.V., Optimal design of batch/semicontinuous processes. Industrial Engineering and Chemical Process Design and Development, 1982, 21, 79-86.

[KOC87] Kocis G.R., Grossmann I.E., Relaxation strategy for the structural optimisation process flowsheets. Industrial Engineering and Chemistry Research, 1987, 26, 1869-1880.

[KOC88] Kocis G.R., Grossmann I.E., Global optimisation of nonconvex mixed-integer non linear programming (MINLP) problems in process synthesis. Industrial Engineering and Chemistry Research, 1988, 27, 1407-1421.

[KOC89] Kocis G.R., Grossmann I.E., Computational experience with DICOPT solving MINLP problems in process systems engineering. Computers and Chemical Engineering, 1989, 13, 307-315.

[LAF02] Laforenza D., Grid programming : some indications where we are headed. Parallel Computing, 2002, 28, 1733-1752.

[LEE03] Lee S., Grossmann I.E., Global optimization of nonlinear generalized disjunctive programming with bilinear equality constraints : applications to process networks. Computers and Chemical Engineering, 2003, 27, 1557-1575.

[LET03] Letchford A.N., Lodi A., An augment-and-branch-and-cut framework for mixed 0-1 programming. Lecture Notes in Computer Science, Combinatorial Optimization, 2003, 2570, 119-139.

[LEY99] Leyffer S., User manual for MINLP_BB. University of Dundee, Numerical Analysis Report NA/XXX, 1999.

[MAH90] Mah R.S.H., Process structures and information flows. Series in Chemical Engineering, Butterworths, Boston (MA), 1990.

[MET53] Metropolis N., Rosenbluth A., Rosenbluth R., Teller A., Teller E., Equation of state calculations by fast computing machines. Journal of Chemical Physics, 1953, 21, 1087-1092.

[MIC94a] Michalewicz Z., Attia N.F., Evolutionary optimization of constrained problems. Communication : Third Annual Conference of Evolutionary Programming, A.V. Sebald, World Scientific, Singapore, 1994, 98-108.

[MIC94b] Michalewicz Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, seconde édition, 1994.

[MIC95] Michalewicz Z., Genetic algorithms, numerical optimization, and constraints. Communication : Sixth International Conference on Genetic Algorithms, L. Eshelman (Ed.), San Mateo, Morgan Kauffmann Publishers, 1995, 151-158.

[MIC96a] Michalewicz Z., Dasgupta D., Le Riche R.G., Schoenauer M., Evolutionary algorithms for constrained engineering problems. Computers and Industrial Engineering, 1996, 30, 851-870.

[MIC96b] Michalewicz Z., Schoenauer M., Evolutionary algorithms for constrained parameters optimization problems. Evolutionary Computation, 1996, 4, 1-32.

[MIG03] Migdalas A., Toraldo G., Kumar V., Nonlinear optimization and parallel computing. Parallel Computing, 2003, 29, 375-391.

[MOD89] Modi A.K., Karimi I.A., Design of multiproduct batch processes with finite intermediate storage. Computers and Chemical Engineering, 1989, 13, 127-139.

[MON00] Montagna J.M., Vecchietti A.R., Iribarren O.A., Pinto J.M., Asenjo J.A., Optimal design of protein production plants with time and size factor process models. Biotechnology Programming, 2000, 16, 228-237.

[MON03] Montagna J.M., Vecchietti A.R., Retrofit of multiproduct batch plants through generalized disjunctive programming. Mathematical and Computer Modelling, 2003, 38, 465-479.

[MON04] Montastruc L., Azzaro-Pantel C., Pibouleau L., Domenech S., A systemic approach for pellet reactor modelling : application to water treatment. AIChE Journal, 2004, 50, 2513-2524.

[PAD91] Padberg M., Rinaldi G., A branch-and-cut algorithm for the resolution of large scale symmetric travelling salesman problems. SIAM Review, 1991, 33, 60-84.

[PAR96] Pareto V., Cours d'économie, Rouge, Lausanne, Suisse,1896.

[PAT91] Patel A.N., Mah R.S.H., Karimi I. A., Preliminary design of multiproduct non-continuous plants using simulated annealing. Computers and Chemical Engineering, 1991, 15, 451-469.

[PIB99] Pibouleau L., Floquet P., Domenech S., Azzaro-Pantel A., A survey of optimization tools through ESCAPE symposia. Computers and Chemical Engineering Supplement, 1999, S495-S498.

[PIN01] Pinto J.M., Montagna J.M., Vechietti A.R., Iribarren O.A., Asenjo J.A., Process performance models in the optimization of multiproduct protein production plants. Biotechnology and bioengineering, 2001, 74, 461-465.

[PON05] Ponsich A., Azzaro-Pantel C., Pibouleau L., Domenech S., About the relevance of mathematical programming and stochastic optimisation methods : application to optimal batch plant design problems. Communication : ESCAPE15, Barcelona (Spain), 31 mai-1er juin 2005, 58-72.

[RAG05] Raghuwanshi M.M., Kakde O.G., Survey on multiobjective evolutionary and real coded genetic algorithms. Communication : 7th International Conference on Adaptative and Natural Computing Algorithms, Coimbra (Portugal), 21-23 Mars 2005.

[RAM94] Raman R., Grossmann I.E., Modelling and computational techniques for logic based integer programming. Computers and Chemical Engineering, 1994, 18, 563-579.

[RAV98] Davemark D.E., Rippin D.W.T., Optimal design of a multi-product batch plant. Computers and Chemical Engineering, 1998, 22, 177-183.

[RIC89] Richardson J.T., Palmer M.R., Liepins G., Hilliard M., Some guidelines for genetic algorithms with penalty functions. Communnication : Third International Conference on Genetic Algorithms, D. Schaffer (Ed.), George Mason University, Morgan Kauffmann Publishers, 1989, 191-197.

[ROB72] Robinson J.D., Loonkar Y.R., Minimizing capital investment for multiproduct batch plants. Process Technology Intelligence, 1972, 17, 861-863.

[ROS60] Rosen J., The gradient projection method for nonlinear programming : I. Linear constaints. Journal of the Society for Industrial and Applied Mathematics, 1960, 8, 181-217.

[RYO95] Ryoo H.S., Sahinidis N.V., Global optimization of nonconvex NLPs and MINLPs with applications in process design. Computers and Chemical Engineering, 1995, 19, 551-566.

[SHA02] Shafiei S., Davin A., Domenech S., Azzaro-Pantel C., Application of genetic algorithm to the mass exchange networks. Communication : 52nd Canadian Conference of Chemical Engineering (3rd Symposium on Process Integration), Vancouver (Canada), 20-23 octobre 2002.

[SIL03] Silva C.M., Biscaia Jr. E.C., Genetic algorithm development for multi-objective optimization of batch free-radical polymerization reactors. Computers and Chemical Engineering, 2003, 27, 1329-1344.

[SMI99] Smith E.M.B., Pantelides C.C., A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Computers and Chemical Engineering, 1999, 23, 457-478.

[TEH03] Teh Y.S., Rangaiah G.P., Tabu search for global optimization of continuous functions with application to phase equilibrium calculations. Computers and Chemical Engineering, 2003, 27, 1665-1679.

[TRI05] Triki E., Collette Y., Siarry P., A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. European Journal of Operational Research, 2005, 166, 77-92.

[TSU01] Tsutsui S., Goldberg D E., Search space boundary extension method in real-coded genetic algorithms. Information Sciences, 2001, 133, 229-247.

[VEC99] Vecchietti, Grossmann I.E., LOGMIP : a disjunctive 0-1 non-linear optimizer for process system models. Computers and Chemical Engineering, 1999, 23, 555-565.

[VIS90] Viswanathan J., Grossmann I.E., A combined penalty function and outer-approximation method for MINLP optimisation. Computers and Chemical Engineering, 1990, 14, 769-782.

[WAN96] Wang C., Quan H., Xu X., Optimal design of multiproduct batch chemical process using genetic algorithms. Industrial and Engineering Chemistry Research, 1996, 35, 3560-3566.

[WAN99] Wang C., Quan H., Xu X., Optimal design of multiproduct batch chemical process using tabu search. Computers and Chemical Engineering, 1999, 23, 427-437.

[WAN02] Wang C., Xin Z., Ants foraging mechanisms in the design of batch chemical process. Computers and Chemical Engineering, 2002, 41, 6678-6686.

[WES95] Westerlünd T., Petterson F., A cutting plane method for solving convex MINLP problems. Computers and Chemical Engineering, 1995, 19, S131-S136.

[WES03] Westerlünd T., Lundqvist K., Alpha-ECP, Version 5.04. An interactive MINLP-solver based on the extended cutting plane method. Report 01-178-A, Process Design Laboratory, Abo Akademi University, Abo (Finlande), 2003.

[WHI93] Whitley D.D., A genetic algorithm tutorials. Technical Report CS-93-103, Colorado State University, Department of Computer Science, 1993.

[WOL97] Wolpert D.H., Macready W.G., No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1997, 1, 67-82.

[WU04] Wu W.H., Lin C.Y., The second-generation of self organizing adaptive penalty strategy for constrained genetic search. Advances in Engineering Software, 2004, 35, 815-825.

[YEH87] Yeh N.C., Reklaitis G.V., Synthesis and sizing of batch/semicontinuous processes : single product plant. Computers and Chemical Engineering, 1987, 11, 639-654.

[YUA88] Yuan X., Zhang S., Pibouleau L., Domenech S., Une méthode d'optimisation non-linéaire en variables mixtes pour la conception de procédés. Recherche Opérationnelle/Operations Research, 1988, 22, 331-346.

TITRE : Stratégies d'optimisation mixte en Génie des Procédés – Application à la conception d'ateliers discontinus

RESUME

La conception d'ateliers discontinus implique généralement la résolution de problèmes d'optimisation non-linéaire en variables mixtes. L'objectif de ce travail est de proposer une méthodologie adaptée pour leur traitement en évaluant les performances de deux méthodes déterministes de l'environnement *GAMS* et un algorithme génétique (AG), sur un jeu d'exemples de complexité croissante.

Avec la formulation de Programmation Mathématique retenue, les résultats numériques vérifient l'efficacité de la méthode de Branch & Bound. Les solutions optimales fournissent une référence pour fixer des procédures appropriées de codage et de gestion des contraintes au sein de l'AG. Ainsi, les performances de ce dernier sont très satisfaisantes et valables pour le cas récurrent de problèmes où le critère est calculé par un simulateur.

Puis le modèle initial est modifié pour traiter les mêmes problèmes en variables purement discrètes. L'AG reste aussi efficace alors que les méthodes déterministes sont dépassées par la combinatoire des problèmes. La stratégie est finalement validée sur un exemple de bioprocédés formulé de manière similaire.

MOTS-CLES : Optimisation mixte non-linéaire (MINLP), Programmation Mathématique, Métaheuristiques, Algorithme Génétique, Conception d'ateliers discontinus.

TITLE : Mixed integer non-linear optimisation strategies for Process Engineering – Application to batch plant design

ABSTRACT

Optimal batch plant design area typically implies the solution of mixed integer non-linear programming (MINLP) problems. The aim of this work is to provide some guidelines for the efficient treatment of these problems. Efficiencies of two deterministic methods from the *GAMS* environment and of a genetic algorithm (GA) are evaluated on a set of increasing complexity examples.

In the framework of the chosen Mathematical Programming formalism, computational results prove the efficiency of a Branch & Bound method. Optimal results provide good references in order to determine appropriate encoding and constraint handling procedures within the GA. Thus, performances of this one are really satisfactory and are still true for the classical case of problems for which the criterion is evaluated through the use of a simulator.

The initial model is then modified in order to treat the same example set with pure discrete variables. The GA keeps being efficient. Besides, the Mathetical Programming methods are overwhelmed by the combinatory effect. The strategy is finally validated on a similarly formulated biotechnology example.

KEYWORDS : Mixed integer non-linear programming optimisation (MINLP), Mathematical Programming, Metaheuristics, Genetic Algorithm, Batch plant design.