UNIVERSITÉ DE MONTRÉAL

ORDONNANCEMENT DE PROJETS INTERNATIONAUX AVEC CONTRAINTES DE MATÉRIEL ET DE RESSOURCES

JEAN-BAPTISTE BOUCHERIT

DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INDUSTRIEL)

DÉCEMBRE 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ORDONNANCEMENT DE PROJETS INTERNATIONAUX AVEC CONTRAINTES DE MATÉRIEL ET DE RESSOURCES

Présenté par : BOUCHERIT Jean-Baptiste

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BOURGAULT Mario, ing., Ph. D., président

M. PELLERIN Robert, ing., Ph. D., membre et directeur de recherche

M. HAJJI Adnène, ing., Ph. D., membre et codirecteur de recherche

M. GAMACHE Michel, ing., Ph. D., membre

DÉDICACE

À ma sœur Alice, la seule à le mériter vraiment.

REMERCIEMENTS

Je voudrais remercier tout d'abord mes directeurs de recherche Messieurs Robert Pellerin et Adnène Hajji, qui m'ont offert un encadrement et un support d'une grande qualité tout au long de ma recherche. Leurs conseils m'ont été d'une aide précieuse et leur disponibilité fort appréciable. Ils ont su me transmettre leur envie et me communiquer leur passion pour leur travail dans ce domaine.

Je ne saurais jamais assez remercier aussi Messieurs Benoît Saenz de Ugarte et François Berthaut, pour l'aide importante qu'ils m'ont offerte. Leur disponibilité et leur expérience me furent extrêmement profitables.

J'adresse ensuite mes remerciements respectueux aux membres du jury, Messieurs Mario Bourgault et Michel Gamache, qui me font l'honneur d'examiner ce travail et d'en être les rapporteurs.

Je remercie également Monsieur Stephen A. Jarislowsky par l'intermédiaire de sa fondation, l'entreprise SNC-Lavalin et le CRSNG, pour l'aide financière dont j'ai pu bénéficier au sein de la Chaire de Recherche Jarislowsky / SNC-Lavalin en Gestion de Projets Internationaux.

Je remercie les membres de l'équipe de Monsieur Pellerin, amis et collègues que j'ai eu l'opportunité de côtoyer, Alban, Andrée-Anne, Coralie, Hélène, Houaida, Javad, Loïc, Lucas et Sandrine, qui par leur sympathie et leur soutien durant les différentes phases de mon projet de recherche, ont permis une collaboration qui me fut des plus agréables.

De la même manière, je remercie ceux qui durant ces deux années m'ont soutenu, motivé, conseillé, et qui de près ou de loin ont joué un rôle dans la bonne réalisation de cette maîtrise. Je pense en particulier à Alexis, Benoît, Camille, Caroline, Clélia, Élodie, Elsa, Fabien, François, Guillaume, Hicham, Hubert, Julien, Juliette, Katrin, Margaux, Mathilde, Matthieu, Mélanie, Méryl, Pierre-Antoine, Romain, Sabine, Sophie et Thomas, et tous ceux que je n'ai pas nommés.

Je remercie Nora B. et La Blogothèque pour la musique en général et tout en particulier.

Enfin, je ne peux finir sans remercier ma famille, mes parents Élisabeth et Michel, mon frère Clément, ma sœur Alice, qui m'ont toujours soutenu dans les projets que j'ai entrepris et sans qui je ne serais pas là où j'en suis aujourd'hui.

RÉSUMÉ

L'évolution des processus d'affaires, rendue nécessaire par une globalisation des marchés toujours plus importante, a encouragé les firmes internationales à se tourner vers le fonctionnement par projet, qui s'est peu à peu généralisé. Mais la gestion de projets demeure une discipline complexe, impliquant de nombreux acteurs, sous-traitants et parties prenantes, et nécessitant le transport, l'approvisionnement et la livraison d'équipement et de matériel. Dans ce contexte, des délais d'approvisionnement et des contraintes de capacité de stockage peuvent amener des retards dans l'avancement des projets et des dépassements de budget. En effet, en dépit des tous les efforts accomplis en recherche pour développer des outils efficaces, la prise en compte des contraintes de livraison et d'approvisionnement lors de la phase de planification de projets demeure pour l'essentiel gérée manuellement par le gestionnaire de projets, selon son intuition et son expérience. Ainsi, les méthodes et algorithmes développés ne sont utilisés en pratique que pour la gestion dite « traditionnelle » de projet, ne considérant qu'un unique projet de faible portée, sans contraintes logistiques.

Reconnaissant cette nouvelle réalité et les besoins qui en découlent, ce mémoire envisage une approche de résolution plus intégrée où l'on considère le problème de planification de projets avec des contraintes liées au stockage et à la livraison de matériel. Pour cela, un générateur aléatoire de contraintes logistiques a été développé, permettant de définir les données du problème relatives aux contraintes logistiques, tout en jouant sur leur poids relatif. La résolution du problème ainsi formulé est effectuée par le biais d'un algorithme génétique optimisé afin de déterminer un échéancier de projet réalisable. L'algorithme, par des opérations de sélection, croisement et mutation sur une population de solutions admissibles, améliore globalement la qualité de celle-ci au fil des itérations et converge peu à peu vers un optimum.

Sur le plan méthodologique, nous avons appliqué une démarche de résolution séquentielle dans un ordre croissant de complexité. L'étude s'est d'abord concentrée sur un projet simple de taille réduite, afin d'effectuer le calibrage des différents paramètres de l'algorithme et de valider la méthode utilisée en testant la convergence des solutions. Des comparaisons ont ensuite été effectuées avec des méthodes heuristiques et exactes, permettant là d'émettre des hypothèses sur leur comportement à plus grande échelle. Enfin la dernière phase de simulation a permis de tester l'approche de résolution pour des projets de plus grande taille afin de se rapprocher d'une

situation réelle, et le nombre d'activités composant les projets a été progressivement augmenté, ce qui a eu pour effet de complexifier fortement le problème. Les résultats montrent qu'il est important de trouver une juste mesure entre performance et qualité, pour assurer une réponse dans un temps court tout en garantissant un faible écart à l'optimum.

Les résultats obtenus lors de cette étude sont très prometteurs et témoignent du bon comportement et de l'efficacité de l'algorithme génétique pour résoudre ce type de problème. Le modèle développé est flexible et robuste, et permet de simuler précisément les problèmes réels. Il a été validé sur des cas théoriques complexes admettant l'ensemble des contraintes. Et enfin, il a permis d'obtenir des solutions réalisables dans un temps raisonnable.

Cette étude ouvre aussi des perspectives de recherche intéressantes pour continuer l'amélioration des outils mis à la disposition des gestionnaires. Les principaux axes de recherches sont l'amélioration de l'algorithme, son hybridation et son intégration avec des systèmes existants de gestion de projets.

ABSTRACT

With globalization of markets, new business models have emerged and international companies started using project management technics. However, the management of international projects remains complex as it involves several sub-contractors and requires the transportation of lots of construction equipment and materials. In that context, long material delivery times and storage capacity constraints may lead to project delays and budget overruns. Indeed, in spite of all research efforts accomplished to develop strong project management tools, project plans taking into account space and equipment availability for the execution of tasks are mostly manually developed on the basis of planner's intuition and experience. Unfortunately, this task is nearly infeasible to perform in the case of large projects, due to the combinational nature of the resource allocation problem. Methods and algorithms are only used for project management in a "traditional" way, with a single small project and without any logistic constraint.

The proposed model addresses this issue by formulating this problem as a scheduling problem with limited resources - resources can be either employees or storage areas - and by defining material delivery constraints. To that purpose, a random logistic constraints generator was developed, in order to create the problem data and to choose its relative weight. The resolution of the formulated problem is performed by a genetic algorithm, which determines a feasible project plan. This algorithm applies many operators on a population of solutions, such as selection crossover and mutation, to improve the population global quality and make the solutions converge towards a local optimum.

Concerning the methodology of research, the study focused first on a single basic project, in order to define the genetic algorithm parameters and to validate the method by testing the convergence. The method was then compared to another heuristic and one exact method. And finally, the last part of the study concerned simulations with larger projects - 60, 90 and 120 tasks - in order to validate the algorithm in a highly complex situation.

Results obtained are very promising and demonstrate the effectiveness of genetic algorithms to solve this class of problems. The developed model is quite accurate but remains soft. It returns good solutions in a short CPU time. However, further testing with multiple project data sets is required in order to generalize these results.

The study offers some interesting prospects for the future. Some of them would be the improvement of the genetic algorithm by implementing new mechanisms, the hybridization with another algorithm, and finally the implementation of the developed algorithm within Enterprise Resource Planning systems.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ÉQUATIONS

LISTE DES SIGLES ET ABRÉVIATIONS

λ	Individu / Chromosome / Échéancier
3D	3 dimensions
4D	4 dimensions (3D + temps)
ACO	Ant Colony Optimization
ACOSS	Ant Colony Optimization with Scatter Search
AIS	Artificial Immune Systems
ANOVA	Test statistique d'analyse de la variance, de l'anglais ANalysis Of VAriance
BB	Branch and Bound
BCO	Bee Colony Optimization
C=(C_1,...,C_n)	Ensemble des durées de réalisation des activités
CAO	Conception Assistée par Ordinateur
C_i	Durée de complétion de l'activité i
C_{max}	Durée totale de l'échéancier S
C_p	Taux de croisement
CS	Cuckoo Search
E	Ensemble des contraintes de précédence
E_γ	Nombre d'unités d'espace disponibles dans l'espace d'entreposage γ
e_{im}	Nombre d'unités d'espace occupées par le matériel m pour l'activité i
ERP	Enterprise Resource Planning
$f(\lambda)$	Fonction d'évaluation de la qualité des individus
f_{best}	Meilleure solution au sein de la population
FIFO	First In First Out
G=(V,E)	Réseau orienté des activités avec contraintes de précédence

GA	Genetic Algorithm
GEN	Nombre de générations / Critère de terminaison
GSO	Glow Swarm Optimization
HSOC	Hypothèse Scientifique Originale de Contribution à la recherche
$i \rightarrow j$, (i,j)	Contrainte de précédence de i avec j
JIT	Just In Time
KA	Kangaroo Algorithm - Algorithme kangourou
L_{im}	Nombre de périodes d'entreposage nécessaires avant le début de l'activité i
M	Quantité de matériel m utilisé au total
M_p	Taux de mutation
MS	Monkey Search
n	Nombre d'activités
NFL	Théorème dit du « No Free Lunch »
N_{im}	Quantité de matériel m utilisé par l'activité i
NP-difficile	Se dit d'un problème Non-déterministe Polynômial
p_i	Durée de l'activité i
POP	Taille de la population de chromosomes
Pred(j)	Ensemble des prédécesseurs directs de j
PSO	Particle Swarm Optimization
PSPLIB	Project Scheduling Problem LIBrary
R	Ensemble des ressources renouvelables
RCPSP	Resource-Constrained Project Scheduling Problem
r_{ik}	Quantité de ressources renouvelables k requises lors de l'activité i
R_k	Quantité de ressources renouvelables k disponibles au total

$S=(S_1,...,S_n)$	Ensemble des temps de début des activités
SA	Simulated Annealing
$S_f=S_T \cap S_R \cap S_M$	Ensemble des échéanciers S réalisables
SGS	Schedule Generation Scheme
S_i	Temps de début de l'activité i
S_M	Ensemble des échéanciers S respectant les contraintes matériel
$S_{M\gamma}$	Ensemble du matériel m entreposé à l'espace γ
S_R	Ensemble des échéanciers S respectant les contraintes de ressources
S_T	Ensemble des échéanciers S respectant les contraintes de précédence
$Succ(i)$	Ensemble des successeurs directs de i
TS	Tabu Search
V	Ensemble des activités (tâches)

LISTE DES ANNEXES

INTRODUCTION

La concurrence internationale et la globalisation des marchés sont des facteurs contraignants pour les entreprises de nos jours. Afin de demeurer compétitives dans le contexte économique actuel, elles doivent adopter des méthodes de gestion plus efficientes.

À cet égard, le fonctionnement par projets figure parmi les méthodes de gestion les plus répandues. Cette méthodologie consiste en l'allocation de ressources financières, humaines et matérielles, l'établissement d'un échéancier, et le suivi du déroulement des différentes activités, de manière à atteindre des objectifs précis dans le cadre de contraintes et exigences spécifiées. À de nombreux égards, cette démarche peut apparaître complexe. Cette complexité s'accroît dans un contexte international en raison de l'apparition de nouvelles contraintes. Ces contraintes sont liées principalement à la taille des projets, à la dispersion géographique des ressources sur plusieurs sites, à la décentralisation des processus décisionnels, la délocalisation des activités et à une logistique nécessairement plus importante.

Plusieurs méthodes d'ordonnancement de projets ont été développées. Toutefois, leur efficacité a été démontrée principalement dans le cadre traditionnel de l'exécution de projets uniques, sur sites localisés. Leur utilisation s'avère inadéquate dans bien des cas. Ainsi, près de la moitié des projets internationaux dépassent les délais de complétion ou n'amènent pas les bénéfices escomptés. Il est pourtant nécessaire d'apporter une réponse concrète à la demande grandissante des entreprises en outils fiables, dédiés à la planification et au contrôle de projets internationaux.

Mais malgré l'abondance de la recherche dans ce domaine, les caractéristiques propres au contexte international continuent à être ignorées. D'une part, les approches de résolution exacte proposées dans la littérature sont incapables de traiter des projets comportant des centaines d'activités, comme c'est le cas dans la majorité des projets internationaux. Et bien que les approches heuristiques et métaheuristiques y soient plus adaptées, leur performance reste très variable d'un projet à l'autre et peu de tests ont été menés sur des projets de grande taille. D'autre part, les contraintes logistiques, importantes pour ce type de projet, sont trop fréquemment omises. Ainsi, les modèles actuels ne considèrent que trop peu souvent d'autres ressources que les ressources humaines, quand bien même la gestion des ressources matériel et de leur approvisionnement a un impact majeur sur la planification d'un projet, en raison de la part importante du budget qu'elle occupe, mais aussi en raison des délais d'exécution qu'elle peut

engendrer. Bien que ces pratiques commencent à se répandre et se généraliser en gestion de projet « traditionnelle », elles demeurent difficilement applicables dans un contexte international. Car en effet, la conduite de projets à l'étranger est souvent plus complexe et plus risquée du fait de l'utilisation de nouveaux fournisseurs locaux et du transport de matériel et d'équipements sur de grandes distances. Le contexte international oblige ainsi à coordonner les activités logistiques avec soin en tenant compte des diverses contraintes de livraison et de stockage sur site. Malgré leur importance, ces contraintes ont été peu abordées et traitées de façon indépendante dans la littérature scientifique.

Ainsi, malgré des progrès récents, les méthodes de résolution et les outils de planification de projet demeurent aujourd'hui encore incapables d'intégrer et résoudre l'ensemble des contraintes matériel et de ressources ainsi que les conflits d'utilisation de la capacité de stockage. Dans ce mémoire, nous proposons donc d'inclure le traitement des contraintes matériel, allant de la livraison aux conflits de stockage. Cela nous amène alors à aborder différentes problématiques. Tout d'abord il est nécessaire de s'interroger sur la meilleure façon de modéliser ces nouvelles contraintes, pour en tenir compte au même titre que les contraintes de ressources. Cela nécessite de penser aussi à la modélisation de l'espace de stockage. Et il nous paraît important de chercher à proposer un outil performant, à même de fournir des résultats dans des délais raisonnables, même pour des projets fortement contraints.

Reconnaissant ces besoins, notre objectif de recherche est donc de proposer un modèle d'ordonnancement de projets performant qui tienne compte des contraintes matériel au même titre que les contraintes ressource et soit capable de développer un échéancier dans un délai raisonnable. Pour y arriver, notre démarche sera la suivante : nous effectuerons dans un premier temps une revue exhaustive de l'état et des avancées de la recherche dans les domaines de la planification et de la logistique appliquée à la gestion de projets, mettant en lumière le manque d'études pour notre cas et donc la nécessité d'aboutir à un résultat concret. Nous identifierons et répertorierons ensuite les différentes techniques applicables pour résoudre notre problème et susceptibles de donner des résultats satisfaisants. Nous adapterons l'une d'entre elles afin de résoudre un cas d'étude de base afin de valider notre démarche et notre protocole opératoire. À cette fin, nous analyserons son comportement ainsi que sa performance et conclurons sur la pertinence de son utilisation. Une fois la validité de notre approche établie, il nous restera à l'appliquer à des modèles plus complexes, de par leur taille et leurs contraintes, et à analyser son

comportement et ses performances dans ce contexte plus proche de la réalité du terrain, ce qui nous permettra de discuter de la pertinence générale de notre étude et des opportunités futures de recherche.

CHAPITRE 1 REVUE DE LITTÉRATURE

Au cours des trois dernières décennies, l'intérêt croissant porté à la gestion de projet a permis le développement et la promotion de nombreuses approches de planification et de suivi de projet. Dans le même temps, de nombreuses techniques traitant les problèmes logistiques sont apparues. Mais ces développements ne l'ont toutefois pas été de concert, ni même simplement pensés pour être intégrés les uns aux autres. À travers cette revue de littérature, nous nous attacherons d'abord à positionner notre problème parmi les différents travaux de recherche passés et actuels. Pour cela, nous étudierons les différentes facettes de l'ordonnancement de projet, puis nous verrons les problèmes logistiques en gestion de projet dans leur globalité, afin de pointer, le cas échéant, leurs lacunes dans notre contexte particulier. Nous effectuerons enfin une taxinomie succincte des approches de résolution envisageables dans notre cas, ce qui nous permettra de faire un choix éclairé.

1.1 Problématique liée à l'ordonnancement de projet

Au niveau de la gestion de projets, il est important de bien distinguer ordonnancement et planification. En effet, la planification vise à déterminer les différentes opérations à réaliser et les moyens matériels et humains à y affecter, alors que l'ordonnancement vise à déterminer les différentes dates correspondant aux activités (Baki, 2007). Pour l'ordonnancement, l'attention se porte principalement, et ce depuis de nombreuses années déjà, sur le traitement des conflits de ressources. Par ressources, on entend généralement ressources humaines. De nombreux modèles ont été créés et de nombreuses approches envisagées (Brucker et al., 1998). Celles-ci diffèrent par leur fonction objective (durée, coût, robustesse), la nature des activités et le type de résolution retenu. Nous allons les détailler dans les sous-sections suivantes.

1.1.1 Ordonnancement de projets avec contraintes de ressources

Les problèmes d'ordonnancement de projet avec contraintes de ressources, appelé aussi RCPSP (Resource-Constrained Project Scheduling Problems), formalisés pour la première fois par Pritsker et al. (1969), ont été abondamment étudiés depuis plus de trente ans et de nombreuses recherches ont été menées dans le but d'optimiser la planification d'activités sujettes à des contraintes de précédence et de ressources afin de minimiser la durée totale du projet, et ce, dans

différents contextes. Ce type de problème est dit NP-difficile au sens fort (Blazewicz et al., 1983), du fait de la nature cumulative de la consommation des ressources, qui permet l'exécution d'activités en parallèle. Durant les dernières décennies, le RCPSP est d'ailleurs devenu un problème standard de planification de projets (Hartmann et Briskorn, 2010) et de nombreuses approches ont été développées, ce qui a poussé certains auteurs à les répertorier (Brucker et al., 1999; Demeulemeester et Herroelen, 2002; Hartman et Briskorn, 2010).

1.1.1.1 Programmation linéaire

Les méthodes exactes figurent parmi les premières approches développées. Les travaux ont tout d'abord porté sur la programmation linéaire en nombres entiers (Pritsker et al., 1969). Cette méthode présente l'inconvénient majeur d'être difficilement applicable à des projets de plus de 60 activités (Artigues et Demassey, 2005). En effet, la linéarisation des contraintes de ressource complexifie fortement le problème et la résolution exacte n'est plus accessible qu'aux projets de quelques dizaines d'activités.

L'une des approches les plus utilisées pour résoudre les programmes linéaires en nombres entiers est la méthode de séparation et d'évaluation progressive, ou selon le terme anglo-saxon, branch-and-bound (Brucker et al., 1998; Demeulemeester et Herroelen, 1997; Mingozzi et al., 1998; Sprecher, 2000). Cette méthode consiste à énumérer et parcourir un ensemble de solutions selon une arborescence, définie en fonction des contraintes du projet. L'espace des solutions est réduit et séparé en sous-problèmes ayant chacun leur ensemble de solutions réalisables, grâce à la définition de bornes inférieures (Artigues et Demassey, 2005). Plusieurs travaux ont d'ailleurs été entrepris pour obtenir des bornes inférieures et des règles de dominance adéquates (Klein et Scholl, 1999).

Plus récemment, des résultats intéressants ont été obtenus en utilisant des approches hybrides, combinant branch-and-bound et heuristiques (Cheng et Wu, 2006), mais de manière générale, les méthodes de résolution exacte sont assez peu envisagées pour l'ordonnancement de projet (Herroelen, 2005). La limitation due à la taille et à la complexité des projets représentant un frein important aux recherches. En effet, selon l'étude réalisée par Kolisch et Sprecher (1996), la majorité des projets type de plus de 60 activités demeurent insolubles par une méthode exacte. Ces approches sont désormais principalement utilisées pour obtenir les solutions exactes de problèmes simples, qui serviront alors à tester la qualité d'autres approches (Hartmann, 1999).

1.1.1.2 Heuristiques

La taille des projets est ainsi un facteur clairement limitant. Les recherches se sont ainsi tournées vers des méthodes résolution approchées, dites heuristiques, à même de proposer dans un délai acceptable un échéancier réalisable et ce en utilisant diverses règles de priorité. Les solutions obtenues ne sont pas optimales, mais compte tenu des incertitudes inhérentes aux projets, ces résultats approchés sont suffisants (Hartmann, 1999).

Les méthodes heuristiques utilisent différents schémas de génération d'échéanciers. Les deux principaux sont le schéma de génération sériel et parallèle, qui, à chaque itération, sélectionnent les activités éligibles à la planification. Pour le schéma sériel, l'incrément est l'activité. Le schéma de génération parallèle est lui incrémenté de manière temporelle (Kolisch et Hartmann, 1999). D'autres schémas ont été développés, telle la méthode de séparation des lots (Batch Splitting Mechanism), et les schémas de recyclage matériaux (Recycle Material Scheme) et de recherche stochastique des prédécesseurs (Stochastic Precedence Search Scheme) (Tang et Tang, 2008). Les activités sont ordonnées selon différentes règles heuristiques, pour former l'échéancier. Le choix de ces règles de priorité a une influence déterminante sur le résultat obtenu, d'autant plus que le nombre de combinaisons possibles entre schémas de génération et règles heuristiques est important (Hartmann, 1999; Kolisch et Hartmann, 2006). Des études ont été effectuées sur les performances des progiciels de gestion de projet, qui dans leur ensemble, utilisent des règles de priorité simples, peu adaptées à la grande variété de projets existants. Leur performance est donc d'une grande variabilité selon le modèle étudié (Herroelen, 2005; Guèvremont, 2009).

Kolisch et Hartmann (1999) ont montré que le schéma de génération parallèle est plus adapté pour les projets comportant beaucoup d'activités. Toutefois, aucune des deux méthodes n'est foncièrement meilleure que l'autre et des études ont même montré que l'utilisation des deux méthodes donne des résultats probants (Kolisch et Hartmann, 2006). L'utilisation de règles de priorité dynamiques, évoluant au cours du temps en fonction des allocations passées, offre aussi des résultats intéressants (Pellerin, 1997).

1.1.1.3 Métaheuristiques

Plus récemment, pour résoudre le problème de la grande variabilité de la qualité des solutions obtenues, de nouvelles approches métaheuristiques ont été développées. Ces approches

comprennent, entre autres, des algorithmes génétiques, des méthodes de type recherche tabou, d'autres de type recuit simulé, de colonies de fourmis, d'optimisation par essaim particulaire, ainsi que des méthodes hybrides. Parmi elles, celles ayant suscité le plus d'intérêt dans la communauté scientifique sont la recherche tabou et les algorithmes génétiques (Kolisch et Hartmann, 2006). Elles sont principalement basées sur des mécanismes de voisinage permettant de faire évoluer la solution. Ces approches heuristiques permettent toujours d'obtenir des solutions réalisables dans un temps acceptable, mais leur variabilité est moindre. Valls et al. (2003) ont d'ailleurs démontré que l'utilisation combinée de ces techniques sur des instances de projets comprenant 120 activités donne de meilleurs résultats que ceux obtenus par les autres approches connues. Debels et Vanhoucke (2005) ont aussi démontré que l'utilisation d'approches métaheuristiques hybrides permet d'obtenir de meilleurs échéanciers que ceux obtenus par le moyen de règles de priorité simple dans le cas de projets comprenant 300 activités et 4 types de ressources, et ce, dans des délais raisonnables. C'est ce développement de nouvelles approches hybrides qui est en plein essor actuellement (Zhang et al., 2005; Tang et Tang, 2008). Ainsi, dernièrement, Chen et al. (2010) proposent leur algorithme ACOSS, combinant une approche de type colonies de fourmis et une méthode dite de recherche dispersée, pour résoudre des problèmes RCPSP comportant jusqu'à 120 activités. Leurs résultats montrent que la voie hybride, en pleine expansion, est une voie très prometteuse pour résoudre les problèmes RCPSP.

1.1.2 Systèmes de planification et de contrôle de projets

Malgré les progrès récents sur le plan scientifique, la plupart des projets continuent à accumuler retards et dépassements de budget (Yourdon, 2003). Cela peut s'expliquer dans de nombreux cas par l'inefficacité des processus de planification et de contrôle de projet (Heroellen, 2005).

Pourtant, la grande majorité des planificateurs de projets utilisent aujourd'hui un système de gestion de projet (Liberatore et Pollack-Johnson, 2003). Ce paradoxe s'explique en partie par le fait que d'une part, les systèmes d'information en gestion de projets sont surtout utilisés pour répondre à des besoins de représentation et de communication et non d'optimisation, et d'autre part, les planificateurs de projets ont souvent une connaissance très limitée du fonctionnement des outils qu'ils utilisent (De Reyck et van de Velde, 1999; Deckers, 2001). Ce dernier problème n'est pas surprenant étant donné que la plupart des systèmes commerciaux de planification de projet reposent sur des outils de résolution propriétaires et non diffusés. De plus, la performance

de ces outils d'ordonnancement, qui reposent essentiellement sur des règles heuristiques simples, varie grandement d'un projet à l'autre et semble diminuer avec un accroissement du nombre d'activités et du nombre de contraintes de ressources (Kolisch, 1999; Guèvremont, 2009; Trautmann et Baumann, 2009).

Les outils de planification de projet souffrent aussi de leur incapacité à traiter les processus connexes à la planification de projet comme la gestion de l'approvisionnement et la gestion du personnel (Dominic, 2008). Étant centré uniquement sur les activités de projets, ils doivent être utilisés avec d'autres systèmes d'information afin d'obtenir un échéancier de projet réalisable tant au niveau de l'exécution, du recrutement que de l'approvisionnement. Certains auteurs ont d'ailleurs soulevé la séparation fréquente en pratique des systèmes d'information utilisés par les différents membres d'une équipe de projet, ce qui amène bien souvent les organisations à mettre en place des processus manuels itératifs d'analyse et de validation des plans de projets et d'approvisionnement (Persona et al., 2004). Il n'est d'ailleurs pas surprenant de voir un nombre grandissant d'entreprises développer leurs propres outils de gestion de projets, plus souples, et auxquels elles peuvent greffer des outils complémentaires, ou d'adopter des outils intégrés de gestion d'entreprise, appelés aussi ERP (Enterprise Resource Planning), afin de mettre en place des processus de planification plus complets et intégrés.

1.1.3 Revue critique

Le cas des problèmes d'ordonnancement de projets internationaux pourrait être vu comme une synthèse des différents problèmes d'ordonnancement existants, auxquels serait ajoutée une composante logistique pour les contraintes matériel. L'ordonnancement de projets internationaux est ainsi complexe pour plusieurs raisons. Tout d'abord, les projets internationaux tendent à couvrir une plus grande portée, à comporter plus d'activités, à s'articuler sur un plus grand nombre de sites, à nécessiter plus de ressources et à comporter plus d'éléments variables que les projets traditionnels (Lientz et Rea, 2003). Cela tend à augmenter fortement l'incertitude relative à ce genre de projets. De plus, la présence de contraintes logistiques importantes engendre des besoins de transport et d'entreposage plus importants (Pellerin et al. 2009). Pourtant, les chercheurs ont ignoré en grande partie ces caractéristiques.

D'une part, les approches de résolution exacte proposées dans la littérature sont incapables de traiter des projets comportant des centaines d'activités, comme c'est le cas dans la majorité des

projets internationaux. Bien que les approches heuristiques et métaheuristiques y soient mieux adaptées, leur performance est très variable d'un projet à l'autre et peu de tests ont été menés sur des projets de très grande taille. D'autre part, les approches et les outils d'ordonnancement de projets ignorent en grande partie les aspects incertains et dynamiques qui caractérisent la majorité des projets internationaux. En plus de l'incertitude liée à l'estimation du niveau de ressources disponibles et l'influence de certaines caractéristiques propres à l'emplacement du site principal de projet, les durées des activités varient souvent en fonction des niveaux de compétences et d'expériences des ressources sélectionnées. Certains chercheurs ont proposé des modèles permettant de mieux appréhender les caractéristiques culturelles et géopolitiques propres aux projets internationaux (He et Hu, 2007; Chaiy et al., 2009), mais ces approches ne sont pas incorporées dans les modèles de planification actuels.

Enfin, l'exclusion des contraintes logistiques représente aussi une limitation très importante des modèles actuels, alors même qu'elles tiennent une part importante dans la productivité et la rentabilité des projets (Polat et Arditi, 2005; Tserng et al., 2006). Là non plus, il n'est pas tenu compte du caractère particulier de la gestion de l'approvisionnement en gestion de projet international. L'utilisation de nouveaux fournisseurs locaux et le transport de matériel et d'équipements sur de grandes distances rend en effet cette contrainte beaucoup plus importante que dans les projets plus conventionnels.

Malgré ces progrès récents, les méthodes de résolution et les outils de planification de projet demeurent aujourd'hui incapables de résoudre l'ensemble des contraintes de ressources, des conflits d'utilisation de la capacité de stockage et des contraintes liées aux particularités de la livraison de matériel en contexte international (Guèvremont, 2009). Il existe de nombreux modèles, ainsi que de nombreuses approches de résolutions différentes. Cependant, aucun d'eux, même parmi les plus aboutis, ne prend en compte la gestion des espaces de stockage et de l'approvisionnement en matériel. Pour rendre cela possible, une approche pourrait être de s'orienter vers des métaheuristiques, dont l'utilisation est désormais largement répandue et qui offrent des résultats prometteurs. Nous allons maintenant nous pencher sur l'intégration des problèmes d'ordonnancement avec contraintes logistiques.

1.2 Problématique liée à la logistique

Une attention croissante est portée aux problèmes logistiques en gestion de projet aujourd'hui, particulièrement dans le domaine de la construction. En effet, de nombreuses études tendent à montrer que ces problèmes jouent un rôle important dans le budget, la productivité et la rentabilité des projets (Howell and Ballard 1997; Agapiou et al, 1998; Polat et Arditi, 2005; Tserng et al., 2006). Ces problèmes logistiques sont principalement dus à un mauvais ordonnancement des différents espaces sur site, principalement au niveau des zones d'entreposage, ou à un processus d'approvisionnement peu en phase avec les réalités du terrain (Jang, 2004; Guèvremont, 2009). Nous allons les détailler dans les sous-sections suivantes.

1.2.1 Problème de gestion de l'espace de construction

1.2.1.1 Problème d'aménagement du site

Lorsque l'on considère les problèmes d'ordonnancement d'espace sur site de construction, l'étude la plus générale est d'abord celle de l'aménagement du site en lui-même (Zouein et al., 2002; Jang, 2004). De nombreux auteurs se sont penchés sur la question et ont proposé différentes approches pour la résoudre. Tommelein et al. (1992) ont été parmi les premiers à exploiter le potentiel de la recherche opérationnelle avec leur modèle « SightPlan » basé sur des techniques d'intelligence artificielle proposant un aménagement des différents espaces du site de construction. Des systèmes d'information géographiques ont été développés (Cheng et O'Connor, 1996; Karan et Ardeshir, 2008), capables d'organiser et de présenter les données référencées, tandis que Retik et Shapira (1999) ont mis en pratique des techniques de réalité virtuelle, afin de simuler et vérifier la qualité d'un aménagement. Plus récemment, les recherches se sont portées sur des approches heuristiques, et en leur sein, particulièrement sur des algorithmes génétiques (Elbeltagi et al., 2001; Mawdesley et al., 2002; Tam et al., 2002; Zouein et al., 2002; Jang, 2004), ou des algorithmes dits de colonies de fourmis (Ning et al., 2010). Enfin, les chercheurs se penchent maintenant sur les approches CAO hybrides (Osman et al., 2003), associant la puissance de la Conception Assistée par Ordinateur à la robustesse des métaheuristiques. Toutefois, ces approches ne considèrent pas le côté ordonnancement de projets et sont limitées à la partie logistique du problème nous concernant.

1.2.1.2 Problème de conflits d'espace

Riley (1994) identifie douze types d'espaces différents, entrant en jeu dans les différentes activités des projets de construction. Parmi ces espaces, certains, tel l'espace de stockage, nous concernent particulièrement, lors de l'ordonnancement desdits projets. Le problème récurrent associé à la gestion de ces projets est le conflit de temps et d'espace, qui peut engendrer d'importants retards dans la planification (Haque et Rahman, 2009). Ainsi, afin de le traiter, diverses stratégies ont été élaborées. Trois branches principales se sont détachées : la visualisation statique associée au processus de planification, la simulation 4D et l'optimisation heuristique.

La première approche privilégiée a été celle consistant à créer des représentations visuelles statiques des zones d'activité associées à chaque tâche du projet et à analyser chacune des situations (Cheng et O'Connor, 1996; Jacobus, 1997). Pareillement, Riley et Sanvido (1997) ont proposé une méthodologie d'ordonnancement de l'espace de construction avec des échéanciers détaillés. Aussi, et afin de faciliter l'approche proactive de traitement des conflits pouvant se produire durant l'exécution de projets, Akinci et al. (2002a) ont développé une classification des conflits de temps et d'espace.

Basées sur la simulation des situations de conflits, de nombreuses méthodes dites 4D ont été développées. Ces méthodes font appel à des représentations visuelles en trois dimensions (3D) créées à l'aide de logiciels tel « AutoCAD » (Guo, 2002) et couplées à une lecture temporelle des différentes situations de manutention liées aux activités du projet de construction (Akinci et al., 2002b; Bjornfot et Jongeling, 2007; Haque et Rahman, 2009). L'utilisation de la simulation permet aux planificateurs d'expérimenter et d'évaluer différents scénarios durant les phases d'ordonnancement (Haque et Rahman, 2009). Ainsi, Akinci et al. (2002b) ont développé le système 4D WorkPlanner Time-Space Conflict Analyser (4D TSConAn) capable de détecter et d'analyser les conflits de temps et d'espace. Plus récemment, mais toujours dans un registre similaire, des outils d'aide à la décision ont été créés, tel VIRCON, élaboré par Winch et North (2006) afin d'analyser les zones de construction et repérer les espaces disponibles en vue de l'affectation des différentes activités. Cet outil permet aussi une interopérabilité forte entre les trois systèmes MS Project, MS Access et AutoCAD.

Finalement, des approches heuristiques ont été développées, permettant de retarder la date de début d'une activité ou de faire varier son niveau de ressources, afin de modifier l'occupation des zones de stockage lors des conflits, le tout en minimisant l'augmentation de la durée du projet (Zouein et Tommelein, 2001). Et plus récemment, plusieurs travaux ont eu recours aux métaheuristiques et plus particulièrement les algorithmes génétiques. Ainsi, Dawood et Sriprasert (2006) ont développé une approche utilisant un algorithme génétique pour résoudre certains problèmes d'optimisation multicritères. Cette méthode minimise la durée du projet ainsi que la fonction de coût, tout en s'assurant qu'il n'y a pas de conflit de temps et d'espace dû à une surcharge des zones d'activité. Enfin, d'autres recherches se sont penchées sur des systèmes hybrides 4D, intégrant des algorithmes métaheuristiques pour rechercher la meilleure stratégie de planification et optimiser les conflits d'espace entre activités (Mallasi, 2009).

1.2.2 Problème de gestion de l'approvisionnement

Plusieurs travaux de recherche ont abordé la question des meilleures pratiques de gestion du matériel, qui mènent à la réussite ou à l'échec de projets de construction (Kini, 1999; Koushki et Kartam, 2004; Laedre et al., 2006). Ces travaux ont clairement identifié la gestion des approvisionnements comme une composante cruciale à prendre en considération lors de la planification des projets de construction (Ng et al., 2009). Pour cela, diverses techniques ont été développées ou adaptées, que nous allons passer en revue dans cette section.

L'approche dite Juste-à-temps est une voie de recherche intéressante. De nombreuses études ont en effet montré que l'application d'un tel concept à la gestion de l'approvisionnement en matériel lors des projets de construction permet un gain de productivité pouvant aller jusqu'à 10% (Bertelsen et Nielsen, 1995; Pheng et Chuan, 2001; Polat et al., 2007). Ce système de gestion permet aussi, en diminuant les stocks, de réduire l'espace d'entreposage et de niveler l'utilisation des ressources. Néanmoins il engendre des coûts plus grands, notamment au niveau de l'inventaire, du fait de l'incertitude de l'avancement des projets et des prix plus élevés des achats à court terme (Polat et al., 2007).

Pour parer au principal inconvénient de l'approche Juste-à-temps, une solution est de provoquer un surplus volontaire de stock, appelé stock tampon, afin de se protéger des fluctuations d'utilisation du matériel, qui ont des effets nuisibles sur la bonne réalisation des projets (Ballard et Howell, 1998; Ng et al., 2009). La question est alors de savoir quelle est la taille optimale du

stock tampon à définir. À cette fin, Horman et Thomas (2005) ont analysé la relation entre l'inventaire du matériel et la performance des travailleurs en construction et ont déterminé un intervalle intéressant pour optimiser la construction et la performance des travailleurs, tout en assurant un tampon suffisant pour absorber les aléas du projet. Au delà de cet intervalle, le stock tampon ne joue plus son rôle, et peut même avoir un effet négatif, en créant des conflits d'espace (Horman et Thomas, 2005).

Des recherches se sont penchées sur la définition d'approches proactives pour résoudre ces problèmes de gestion de l'approvisionnement. Caron et al. (1998) ont présenté un modèle simplifié qui aide à la planification de la livraison des matériaux pour les projets de construction et qui peut être utilisé tôt dans la phase de la planification du projet. Plus récemment, Ala-Risku et Kärkkäinen (2006) s'intéressent à l'approche proactive de la gestion de l'approvisionnement, en proposant une méthode basée sur le suivi des flux de matériel pour rendre la gestion de l'inventaire plus transparente et ainsi permettre de mieux prévoir les capacités des stocks.

Finalement, des méthodes heuristiques sont désormais envisagées pour optimiser l'approvisionnement en matériel lors des projets de construction. Georgy et Basily (2008) utilisent ainsi un algorithme génétique pour fournir des échéanciers d'approvisionnement matériel admissibles et les intègrent dans l'échéancier de construction.

Ainsi, l'application du concept de Juste-à-temps et des nouvelles méthodes heuristiques hybrides sont des solutions envisageables et encourageantes. Cependant, leur bonification s'impose pour permettre une meilleure gestion des aléas du projet. De plus, les approches développées demeurent cantonnées à la logistique et ne servent qu'à optimiser la gestion du matériel en fonction des activités, et non l'inverse. Pour pallier ces limitations, l'approche métaheuristique semble bien être l'approche la plus intéressante dans l'optique de fournir des solutions réalisables dans un temps raisonnable.

1.3 Choix de l'algorithme de résolution

L'espace de recherche au sein des métaheuristiques est particulièrement vaste, tant leur champ d'application est grand. En recherche RCPSP, de nombreuses approches métaheuristiques sont utilisées et développées, mais pour des usages et des caractéristiques propres. Toutefois, depuis les premières recherches sur les approches d'optimisation, certains algorithmes se sont

démarqués. Afin de pouvoir procéder à un choix éclairé, nous allons effectuer une revue détaillée des différents algorithmes métaheuristiques utilisés en gestion de projet.

1.3.1 Algorithmes cités dans la littérature

1.3.1.1 Algorithme génétique (Genetic Algorithm)

L'algorithme génétique est un des premiers algorithmes à avoir été développé puis étudié. C'est une méthode de recherche métaheuristique présentée par Holland (1975), qui trouve son nom dans le fait qu'elle reproduit les différents processus d'évolution et de sélection naturelle. Cet algorithme évolutionniste balaie une large portion de l'espace des solutions et fournit une solution approchée de bonne qualité en un temps court, en optimisant au fil des itérations une population de solutions.

De nombreuses analogies avec la biologie et la génétique sont effectuées. Ainsi l'algorithme opère sur une population de chromosomes, entité à la base de tout organisme vivant, chacun représentant ici une solution au problème considéré. Un chromosome est composé de gènes, chaque gène représentant une donnée de la solution. Différentes opérations sont effectuées sur cette population de chromosome. Les trois principales sont la sélection, le croisement et la mutation. D'autres phénomènes peuvent être engagés, telle la prédation ou la mémoire de l'espèce (Saenz de Ugarte, 2009), qui permettent d'améliorer le temps de complétion en éliminant les redondances.

```
1   Initialiser paramètres
2   Genérer population
3   k := 0
4   tant que k < critère de terminaison faire
5       évaluation()
6       sélection()
7       recombinaison()
8       k := k + 1
9   fin
10  Evaluer population
11  Retourner meilleure solution
```

Figure 1.1 : Pseudo-code pour l'algorithme génétique

Afin de résoudre un problème, il est nécessaire de créer une représentation sous forme génétique du domaine de solution et de coder la fonction d'évaluation des différentes solutions, qui sont les deux bases de fonctionnement de l'algorithme. Et il est important d'accorder une attention toute particulière à la fonction d'évaluation, qui doit être pensée le plus simplement possible, car elle est appelée de nombreuses fois durant le traitement. Une fois ceci effectué et les paramètres initialisés, la population initiale de solutions est générée (figure 1.1). Chaque solution est ensuite évaluée, et elles sont sélectionnées et appariées pour produire par croisement et reproduction une nouvelle génération de solutions, celles-ci pouvant subir une mutation. Chaque nouvelle solution est évaluée et la population mise à jour. Le processus reprend alors avec un nouvel appariement des chromosomes de la population. L'algorithme se termine quand l'un des critères d'arrêt est atteint. Il retourne alors la meilleure solution parmi la population.

1.3.1.2 Recuit simulé (Simulated Annealing)

Un autre type d'algorithme métaheuristique qui s'est popularisé au cours des dernières décennies est l'algorithme dit de recuit simulé (Kirkpatrick et al., 1983). Dans ce cas, l'analogie est faite avec le procédé métallurgique de recuit. C'est un algorithme de descente aléatoire qui opère sur une solution, cherchant à l'améliorer par des opérations locales et pouvant effectuer des sauts aléatoires dans l'espace des solutions.

L'algorithme reproduit la méthode du traitement thermique consistant à effectuer des cycles de chauffage, maintien en température et refroidissement, sur une pièce métallique, afin de modifier sa structure cristalline pour lui faire atteindre un état d'équilibre stable, de plus basse énergie thermodynamique. La probabilité qu'un système thermodynamique passe d'un niveau d'énergie E_1 à un niveau E_2 est donnée par

$$P = e^{-\frac{E_2 - E_1}{kT}} \qquad (1)$$

où k est la constante de Boltzmann (1.38×10^{-23}) et T la température du système, qui varie au cours du temps. Pour simplifier le problème, k est pris égal à 1. La fonction objectif à minimiser est l'énergie E des états s du problème considéré. Le paramètre T décroit en fonction du nombre d'itérations.

Le pseudo-code de l'algorithme est présenté à la figure 1.2. Une première solution s_0, issue de l'espace des solutions, est nécessaire pour démarrer l'algorithme. À cet état s_0 correspond un niveau d'énergie E(s_0). Tant que le critère d'arrêt n'est pas atteint, une nouvelle solution s_{new} est générée dans le voisinage de s puis évaluée. Elle remplace la solution actuelle s si son niveau d'énergie est inférieur ou selon la probabilité P. Le paramètre T dépend du nombre i d'itérations, et décroit selon une loi de décroissance. La fonction random() retourne une valeur aléatoire entre 0 et 1. À l'arrêt des itérations, la meilleure s en date est retournée.

```
1   Initialiser paramètres
2   s := s0 ; e := E(s0)
3   i := 0
4   tant que i < critère de terminaison faire
5        Générer snew := voisin(s)
6        enew := E(snew)
7        si enew < e ou P(enew, e, T(i)) > random() alors
8             s := snew ; e := enew
9        fin
10       i := i + 1
11  fin
12  Retourner s
```

Figure 1.2 : Pseudo-code pour l'algorithme de recuit simulé

1.3.1.3 Recherche tabou (Tabu Search)

Glover (1986) a proposé une nouvelle approche métaheuristique, basée là encore sur un algorithme de descente, qui effectue une recherche locale au voisinage d'une solution pour tendre vers un extrémum. Mais contrairement à l'algorithme de recuit simulé, l'algorithme de recherche tabou possède une mémoire, la liste tabou, qui lui permet d'éviter l'attraction des extrémums locaux, en l'empêchant de retourner à une solution déjà envisagée.

Le fonctionnement de l'algorithme est schématisé à la figure 1.3. Au lancement de l'algorithme, la liste *tabou* est vide. Une première solution s_0 est choisie aléatoirement dans l'espace des solutions. Ensuite, l'ensemble V_s des solutions au voisinage de s_0 est généré et chaque solution s_v est évaluée. Si la meilleure d'entre elles est aussi meilleure que la meilleure solution à l'heure actuelle s_{best}, alors elle devient la solution s de référence et la meilleure solution actuelle est

actualisée. Sinon, l'espace V_s est restreint aux solutions s_v n'étant pas mentionnée dans la liste *tabou*, et la meilleure solution de cet espace est prise comme nouvel espace de référence. Finalement, et comme à la fin de chaque itération, la liste *tabou* est actualisée. L'algorithme s'arrête lorsque le critère de terminaison est atteint, et la meilleure solution s_{best} est retournée.

```
1   s := s0
2   sbest := s
3   tabou := ø
4   tant que critère de terminaison n'est pas atteint faire
5        Générer Vs solutions sv := voisin(s)
6        si best(sv) meilleur que sbest alors
7             s := best(sv)
8             sbest := s
9        sinon
10            s := best(sv) tel que sv ∈ Vs\tabou
11       fin
12       Ajouter s à tabou
13  fin
14  Retourner sbest
```

Figure 1.3 : Pseudo-code pour l'algorithme de recherche tabou

Le fait de stocker les transformations dans une liste tabou pourrait alourdir très fortement le traitement, c'est pourquoi, généralement, les transformations stockées dans la liste le restent durant un certain nombre d'itérations puis sont retirées de la liste, selon un principe « Premier entré, premier sorti » (First In First Out).

1.3.1.4 Algorithme de colonie de fourmis (Ant Colony Optimization)

L'optimisation par colonie de fourmis développée par Dorigo et al. (1992) ouvre la voie à un autre type d'approches métaheuristiques, dites multi-agents. Cette approche reproduit le comportement et les interactions liant les individus au sein d'une population. Dans ce cas, le comportement imité est celui des fourmis parcourant des chemins aléatoires à la recherche de nourriture et qui les marquent avec des phéromones fugaces. L'intérêt de cet algorithme est son aspect « constructif ». Il n'est pas nécessaire que la meilleure solution soit effectivement parcourue pour être obtenue, elle peut être construite à partir des meilleures solutions voisines.

La base du fonctionnement de cet algorithme repose sur les phéromones que déposent les fourmis lors de leurs trajets. Ces hormones volatiles sont détectées par les autres fourmis et leur forte concentration poussera les individus à utiliser majoritairement un chemin plutôt qu'un autre. Avec le temps, la concentration en phéromones diminue, ce qui tend à faire disparaitre les chemins qui sont peu empruntés. Cette propriété est importante puisqu'elle intègre un caractère dynamique dans le processus de recherche d'un chemin.

```
1   Initialiser paramètres
2   Générer population Q
3   Générer matrice des phéromones T := (τij)
4   tant que critère d'arrêt non atteint faire
5       Construire Q solutions sq := f(T)
6       Évaluer les solutions sq
7       Mettre à jour T
8   fin
9   Construire la smax := f(T)
10  Retourner smax
```

Figure 1.4 : Pseudo-code pour l'algorithme de colonie de fourmis

De manière générale, l'algorithme se décompose en plusieurs phases (figure 1.4). La population Q est tout d'abord générée, ainsi que la matrice des phéromones $T := (\tau_{ij})$, telle que :

$$\tau_{ij} = \begin{cases} H \ \forall i \neq j \\ 0 \ sinon \end{cases} \quad (2)$$

Chaque « fourmi » va ensuite construire une solution s_q, en progressant de nœud en nœud selon une probabilité :

$$P_{i \to j}(T) = \frac{(\tau_{ij})^{\alpha}}{\sum_{k \neq i}(\tau_{ik})^{\alpha}} \quad (3)$$

Puis chaque solution est évaluée. Grâce à ces évaluations, la matrice T va être remise à jour, les meilleures solutions amenant une plus forte « concentration » en phéromones, et les chemins les moins fréquentés voyant leur trace de phéromones s'effacer un peu plus à chaque itération. Lorsque le critère d'arrêt est atteint, l'algorithme stoppe ses itérations et la solution maximisant la concentration en phéromones est construite et retournée.

1.3.1.5 Optimisation par essaim particulaire (Particle Swarm Optimization)

Le dernier type d'algorithme utilisé pour traiter des problèmes RCPSP est l'algorithme d'optimisation par essaim particulaire, développé par Kennedy et Eberhart (1995). Comme pour l'algorithme de colonie de fourmis, celui-ci s'inspire des interactions entre individus au sein d'une population. L'algorithme reproduit ici les déplacements au sein de nuées d'oiseaux, bancs de poissons et autres essaims d'insectes, et permet de faire converger les solutions vers des optimums locaux. Bien que cette approche ait été plutôt développée pour les milieux continus, elle a aussi été appliquée aux milieux discrets.

Les solutions sont symbolisées par la position des individus dans l'espace. Dans notre cas, il y a autant de dimensions que d'activités. Les équations du mouvement sont données par :

$$V_i(k+1) = X_i(k+1) - X_i(k) \quad (4)$$

$$V_i(k+1) = \omega . V_i(k) + c_1 . \left(p_{ibest} - X_i(k)\right) + c_2 . \left(g_{best} - X_i(k)\right) \quad (5)$$

où $X_i(k)$ représente la position de la particule i à l'itération k, $V_i(k)$ sa vitesse, p_{ibest} sa meilleure position, g_{best} la meilleure position parmi l'ensemble des particules, ω l'inertie d'une particule, et c_1 et c_2 des coefficients des pondération. Ainsi, à chaque itération, les particules, symbolisant chacune un individu, se déplacent vers le barycentre des trois points pondérés $X_i(k\text{-}1)$, p_{ibest} et g_{best}.

Au lancement, la population P est générée (aléatoirement ou non) dans l'espace de recherche du problème (figure 1.5). La meilleure solution locale pour chaque particule p_{ibest} ainsi que la meilleure solution globale g_{best} sont initialisées. Lors des itérations, les nouvelles positions sont calculées à partir des équations 4 et 5, puis p_{ibest} et g_{best} sont mises à jour le cas échéant. Enfin, lorsque le critère de terminaison k_{max} est atteint, l'algorithme retourne g_{best}.

```
1   Initialiser paramètres
2   Générer population P
3   Initialiser pour chaque pi, pibest := xi(∅)
4   gbest := best(pibest)
5   tant que k < kmax faire
6        pour chaque pi faire
7             Calculer xi(k+1)
8             Mettre à jour pibest si nécessaire
9        fin
10       Mettre à jour gbest si nécessaire
11       k := k+1
12  fin
13  Retourner gbest
```

Figure 1.5 : Pseudo-code pour l'algorithme d'essaim particulaire

1.3.2 Autres algorithmes

De nouveaux algorithmes font leur apparition tous les ans et certains n'ont pas encore été appliqués à des problèmes d'ordonnancement ou de RCPSP. Néanmoins, ils présentent des intérêts et il est envisageable de les voir jouer un rôle non négligeable dans les recherches futures sur les RCPSP et leurs dérivés.

Parmi eux, notons tout d'abord l'algorithme kangourou (Kangaroo Algorithm), développé par Pollard (1978), qui agit de façon similaire à la recherche tabou en utilisant une procédure de descente par mutation de solution, dans son voisinage, et effectuant des sauts aléatoires pour éviter l'attraction des extrémums locaux lorsque la solution ne progresse plus. Présentés par Farmer et al. (1986), les systèmes immunitaires artificiels (Artificial Immune Systems) sont déjà utilisés pour résoudre des problèmes RCPSP (Agarwal et al., 2007). Ils reproduisent les différents mécanismes de mémoire, apprentissage et sélection, inhérents au fonctionnement des systèmes immunitaires naturels. Les algorithmes d'essaims de vers luisants (Glowworm Swarm Optimization) et de colonies d'abeilles (Bee Colony Optimization) sont deux autres méthodes métaheuristiques d'optimisation par essaim, proches des algorithmes d'optimisation par essaim particulaire et de colonies de fourmis (Krishnanand et Ghose, 2005; Chong et al., 2006) qui simulent le comportement et les interactions entre individus au sein d'une population dense. Plus récemment, Mucherino et Seref (2007) ont présenté leur méthode, appelée Monkey Search, basée

sur le comportement des singes qui parcourent les arbres à la recherche de nourriture et se rappellent des branches de qualité. Enfin, l'algorithme métaheuristique le plus récent pouvant être appliqué à notre problématique est la méthode de recherche coucou (Cuckoo Search), présentée par Yang et Deb (2009), et calquée sur le comportement des femelles coucous lors de la période de couvaison, qui déposent leurs œufs dans les nids d'autres oiseaux.

1.3.3 Choix de la métaheuristique

Nous avons passé en revue les différentes métaheuristiques pouvant être utilisées pour notre problème particulier de RCPSP hybride. Certaines d'entre elles se démarquent par leur fréquente utilisation dans les recherches en ce domaine. Parmi elles, l'algorithme génétique semble offrir les meilleures garanties, tant du point de vue de ses caractéristiques, de son adaptabilité, que de ses performances (Azadivar et Tompkins, 1999; Chaudhry et Luo, 2005). En effet, la taille de la population et son renouvellement partiel à chaque itération lui assurent une meilleure résistance à l'attraction des extrémums locaux. Les différents opérateurs pouvant être appliqués permettent à la fois de couvrir le voisinage et de parcourir plus largement l'espace des solutions. Il possède une mémoire, lui assurant de ne pas faire de redondance dans ses recherches. Enfin, de nombreuses autres méthodes peuvent lui être greffées pour optimiser encore son fonctionnement.

Il est toutefois intéressant de rappeler qu'aucune métaheuristique n'est foncièrement meilleure qu'une autre, car en effet si l'une d'elle peut être particulièrement adaptée à une situation, elle ne peut prétendre l'être pour tous les problèmes d'optimisation. Le plus important reste donc de bien modéliser le problème et d'utiliser l'algorithme qui s'y prête le plus. Dans le cas des RCPSP, l'algorithme génétique décompose ses chromosomes en gènes, ce qui convient bien à la décomposition des échéanciers en activités. Nous supposons donc que l'algorithme donnera des résultats acceptables pour un temps donné, tout en permettant la mise en place d'un modèle simplifié grâce aux similitudes relevées.

1.4 Synthèse

Au cours de cette revue de littérature, nous avons abordé les différentes problématiques relatives à notre sujet. Ainsi, nous avons pu démontrer la nécessité de tenir compte des contraintes logistiques dans le cas de la gestion de projets de construction internationaux, du fait des impacts importants de ces contraintes sur la rentabilité et la productivité d'un projet. Toutefois, nous

avons aussi montré que malgré tous les efforts de recherche effectués, les outils de planification et d'ordonnancement de projets de construction internationaux n'offraient toujours pas la possibilité de résoudre les conflits engendrés par ces contraintes logistiques.

Il apparaît nettement ici que les recherches ne sont pas allées dans le sens de la convergence entre les problématiques de logistique et de planification. Du côté logistique, en effet, et malgré l'abondance de références ayant trait au traitement des problèmes logistiques en gestion de projets de construction internationaux, cette recherche a toujours été séparée de la gestion de projet « traditionnelle », et n'offre pas de solution pour intégrer le traitement logistique au sein d'un échéancier. C'est d'ailleurs plutôt l'inverse qui se produit, et les outils ne sont pas développés à l'intention des planificateurs de projet, mais des gestionnaires de production, en leur permettant uniquement de modifier des allocations logistiques, mais sans pouvoir adopter une approche réactive de réordonnancement du planning en fonction des contraintes logistiques. Du côté ordonnancement, les recherches continuent de se focaliser principalement sur les contraintes de ressource, et rien de réellement tangible n'a été étudié pour traiter les problèmes de planification avec contraintes logistiques. De plus, les systèmes d'information et de gestion de projet semblent être inefficaces. En plus de ne pas être utilisés à bon escient pour répondre aux besoins en ordonnancement et optimisation, le fait qu'ils soient pour la plupart des logiciels propriétaires empêche les utilisateurs d'en avoir une connaissance approfondie. Et ces logiciels n'intègrent toujours pas les divers processus liés aux contraintes logistiques.

Ainsi, ce chapitre montre bien la nécessité de parvenir au développement d'un modèle d'ordonnancement de projet avec traitement de contraintes matériel et de ressources, et justifie à notre sens les développements que nous avons effectués. Les projets que nous serons amenés à traiter ont un haut degré de complexité, du fait de leur taille et des contraintes à prendre en compte. Au vu de cette complexité, l'approche de résolution exacte ne semble pas être la plus pertinente. Nous nous sommes donc orientés vers une approche heuristique. Mais, plusieurs travaux ayant pointé l'importante variabilité de la qualité des solutions ainsi obtenues, nous avons préféré aller plus loin et étudier une approche métaheuristique. Parmi ces approches, et après avoir effectué une brève revue des solutions existantes, nous avons opté pour l'approche génétique, qui nous semblait offrir les meilleures garanties quant aux performances en terme de qualité et de vitesse de calcul.

1.5 Objectifs de recherche

Nous avons tout d'abord réfléchi aux questions de recherche que notre revue mettait en lumière. Ainsi, nous avons pu déterminer deux questions qui nous semblaient importantes :

- Comment modéliser les nouvelles contraintes logistiques de livraison et de stockage matériel, pour se rapprocher des situations réelles ?
- Comment traiter simultanément l'ensemble des contraintes ressources et matériel, pour offrir une réponse dans un délai raisonnable ?

L'objectif général de ce mémoire est donc de développer un outil performant et robuste capable de solutionner les modèles complexes de gestion de projets internationaux dans des délais raisonnables. À partir de celui-ci, nous pouvons formuler les trois objectifs spécifiques suivants, qui précisent les axes de notre recherche :

- Simplicité et précision de l'implémentation : Définir un modèle simple et proche des problèmes réels.
- Validation du modèle : Tester le modèle sur des cas théoriques complexes admettant l'ensemble des contraintes.
- Performance du modèle : Obtenir des solutions optimisées en un temps raisonnable.

Les hypothèses scientifiques originales de notre contribution à la recherche (HSOC) sont :

- HSOC n°1 : Le traitement combiné des deux types de contraintes dans la gestion d'un projet international assure un ordonnancement plus réaliste et ainsi un contrôle amélioré des aléas pouvant survenir au cours d'un projet.
- HSOC n°2 : L'utilisation d'une approche métaheuristique utilisant un algorithme génétique permet d'obtenir le meilleur compromis entre la qualité de la solution et la vitesse de résolution.

L'originalité de ces hypothèses découle du fait qu'à notre connaissance, aucune approche de modélisation combinant les deux types de contraintes n'a pas été envisagée à ce jour, les chercheurs se contentant d'effectuer des simulations ne tenant compte que d'une contrainte à la fois. De plus, l'approche génétique n'est utilisée pour l'instant que pour les RCPSP.

Sur le plan expérimental, nous proposons de prendre en compte des contraintes de stockage et de délais de livraison du matériel sur site. Nous ferons cela pour des projets allant de 30 à 120 activités afin de tester notre approche au plus près des conditions réelles. Finalement, l'approche proposée sera comparée tout d'abord à une approche heuristique, puis exacte, et enfin métaheuristique

1.6 Conclusion

Après avoir passé en revue l'état de la recherche sur les modèles ayant trait à la gestion des problèmes logistiques et matériel en gestion de projet, dans le prochain chapitre nous allons présenter le modèle développé, formulé de manière à ce que les ressources matériel soient traitées au même titre que les autres ressources.

CHAPITRE 2 PRÉSENTATION DU MODÈLE

Reconnaissant la complexité de la gestion des projets internationaux et la nécessité d'avoir des outils robustes pour traiter ce type de problème, nous avons orienté nos recherches vers les approches métaheuristiques, les plus à même à notre sens pour offrir des résultats se rapprochant de nos attentes. Parmi elles, nous avons choisi l'approche génétique, qui s'est démarquée par ses performances et son potentiel d'adaptabilité. Parallèlement à cela, nous avons réfléchi au développement d'un modèle de RCPSP hybride, qui intègrerait les contraintes logistiques au même titre que les contraintes de ressources. Le présent chapitre a ainsi pour but de présenter et exposer clairement le modèle développé ainsi que l'approche de résolution retenue.

2.1 Formulation du problème

2.1.1 Modèle RCPSP

Le modèle de base est donc un modèle d'ordonnancement de projet avec contraintes de ressources.

2.1.1.1 Notation

Dans ce modèle, dont les notations sont reprises au tableau 2.1, le projet est défini par un ensemble V d'activités et un ensemble E de contraintes de précédence. Il peut être représenté par un graphique orienté $G = (V, E)$ décrit par un réseau d'activités sur les nœuds (AON), où les activités sont représentées par les nœuds et les arcs indiquent l'ordre des opérations, selon des relations de précédence (Brucker et al., 1999). Tous les couples (i, j) dans G, représentés par une relation $i \rightarrow j$ dans l'ensemble E, montrent ainsi une contrainte de précédence de l'activité i avec l'activité j. La durée d'exécution de l'activité i est donnée par p_i. Différents ensembles sont définis. Ainsi $Pred(j)$ représente l'ensemble des prédécesseurs directs de j, $Succ(i)$ l'ensemble des successeurs directs de i, et R l'ensemble des ressources renouvelables.

La quantité de ressources k disponible au total est notée R_k, tandis que r_{ik} représente la quantité de ressources k requise pour l'activité i. Le temps de début de l'activité i est noté S_i, et l'ensemble de ces temps, qui représente l'échéancier, est noté S. De façon similaire, C_i représente le temps de complétion de l'activité i, et l'ensemble C regroupe toutes ces durées. Enfin, C_{max} représente la durée totale d'un échéancier S. C'est la fonction objectif, que l'on cherchera à minimiser.

Tableau 2.1 : Symboles et définitions

Symbole	Définition
V	Ensemble des activités (tâches)
n	Nombre d'activités
E	Ensemble des contraintes de précédence
$G=(V,E)$	Réseau orienté des activités avec contraintes de précédence
$i \rightarrow j$, (i,j)	Contrainte de précédence de i avec j
p_i	Durée de l'activité i
$Pred(j)$	Ensemble des prédécesseurs directs de j
$Succ(i)$	Ensemble des successeurs directs de i
R	Ensemble des ressources renouvelables
R_k	Quantité de ressources renouvelables k disponibles au total
r_{ik}	Quantité de ressources renouvelables k requises lors de l'activité i
S_i	Temps de début de l'activité i
$S=(S_1,...,S_n)$	Ensemble des temps de début des activités
C_i	Durée de complétion de l'activité i
$C=(C_1,...,C_n)$	Ensemble des durées de réalisation des activités
C_{max}	Durée totale de l'échéancier S
S_T	Ensemble des échéanciers S respectant les contraintes de précédence
S_R	Ensemble des échéanciers S respectant les contraintes de ressources

2.1.1.2 Contraintes

Le modèle RCPSP traite deux types de contraintes : les contraintes de précédence et les contraintes de ressources. Les contraintes de précédence définissent des conditions temporelles entre les activités. Ainsi, un successeur ne peut débuter que lorsque tous ses prédécesseurs sont terminés :

$$S_i + p_i \leq S_j \quad \forall (i,j) \in V \tag{6}$$

Pour les contraintes de ressources, il est nécessaire au préalable de définir la variable binaire $x_i^t = \begin{cases} 1 \; si \; l'activité \; i \; débute \; au \; temps \; t \\ 0 \; sinon \end{cases}$ qui est liée à la variable S_i (équation 7). Quant à l'unicité de S_i, elle est assurée par l'équation 8).

$$\sum_{t=1}^{T} t.x_i^t = S_i \quad \forall i \in V \tag{7}$$

$$\sum_{t=1}^{T} x_i^t = 1 \quad \forall i \in V \tag{8}$$

La contrainte de ressources peut ensuite être définie (équation 9). Elle assure que la quantité de ressources *k* utilisée par les activités en cours à un instant *t* est inférieure ou égale au nombre de ressources disponibles à cet instant.

$$\sum_{i \in V} \sum_{s=max\{1,t-p_i+1\}}^{t} r_{ik}.x_i^s \leq R_k \quad \begin{matrix} \forall t = 1, \dots, T \\ \forall k \in R \end{matrix} \tag{9}$$

2.1.2 Contraintes supplémentaires

À ce modèle, il convient de rajouter les contraintes logistiques que nous souhaitons traiter dans notre problème. Nous avons tout d'abord défini la ressource matériel, notée *m*. La quantité totale de matériel est elle notée *M*. Une activité *i* consommera, en plus des ressources propres au RCPSP, une certaine quantité N_{im} de matériel. Pour modéliser la livraison de ce matériel, nous avons choisi de nous assurer que le matériel soit livré en avance, pour pouvoir être utilisé en temps voulu (figure 2.1). Ainsi, nous avons défini L_{im}, qui est le nombre minimum de périodes d'entreposage nécessaire avant le début de l'activité *i*. L_{im} est défini positif ou nul.

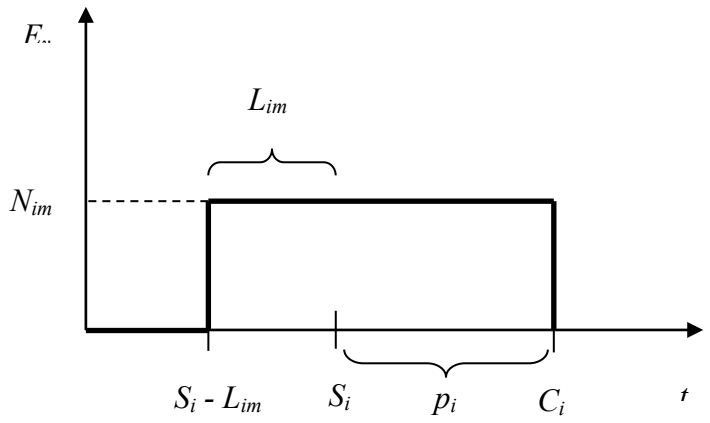


Figure 2.1 : Schéma de livraison matériel avant une activité i

Nous avons ensuite défini des zones d'entreposage. Dans ce modèle, nous avons supposé que la capacité d'un espace d'entreposage pour un matériel donné était constante. Aussi, cet espace d'entreposage est utilisé par le matériel jusqu'à la fin des activités qui lui sont associées. En conséquence, l'espace d'entreposage reste constant pour l'activité i pour la période $L_{im} + p_i$. Enfin, nous n'avons considéré que des surfaces d'entreposage, pour ne pas complexifier inutilement le traitement en utilisant des unités de volume. La capacité totale de la zone d'entreposage de type γ est notée E_γ. $S_{M\gamma}$ indique l'ensemble du matériel qui peut être entreposé dans la même zone d'entreposage γ. Le vecteur e_{im} est défini comme la surface occupée par le matériel m durant l'exécution de l'activité i. Le tableau 2.2 synthétise les nouvelles notations utilisées dans notre problème.

Tableau 2.2 : Notations supplémentaires

Symbole	Définition
M	Quantité de matériel m utilisé au total
N_{im}	Quantité de matériel m utilisé par l'activité i
L_{im}	Nombre de périodes d'entreposage nécessaires avant le début de l'activité i
E_γ	Nombre d'unités d'espace disponibles dans l'espace d'entreposage γ
$S_{M\gamma}$	Ensemble du matériel m entreposé à l'espace γ
e_{im}	Nombre d'unités d'espace occupées par le matériel m pour l'activité i
S_M	Ensemble des échéanciers S respectant les contraintes matériel
$S_f = S_T \cap S_R \cap S_M$	Ensemble des échéanciers S réalisables

Les contraintes d'entreposage et d'approvisionnement sont définies à l'équation 10. Celle-ci assure que l'espace total occupé par le matériel à un instant *t* à l'intérieur d'un même espace d'entreposage ne peut pas excéder la capacité de cet espace d'entreposage en tout temps :

$$\sum_{m \in S_{M\gamma}} \sum_{i \in V} \sum_{s=max\{0,t-p_i+1\}}^{t+L_{im}} e_{im}.x_i^s \leq E_\gamma \quad \begin{matrix} \forall t = 1, \dots, T \\ \forall \gamma = 1, \dots, \Gamma \end{matrix} \tag{10}$$

2.1.3 Modèle final

Le modèle final ainsi constitué est donc muni d'une fonction objectif minimisant la durée totale de projet, et de quatre contraintes limitant l'espace des solutions :

- Contrainte de précédence : Une activité ne peut démarrer avant la fin de tous ses prédécesseurs.
- Contrainte de ressources : La quantité de ressources requises pour une tâche ne peut dépasser la quantité totale disponible.
- Contrainte d'approvisionnement : Le matériel requis pour une activité doit être disponible avant le démarrage de l'activité.
- Contrainte d'entreposage : La surface occupée par le matériel durant l'exécution d'une activité ne peut dépasser la capacité de stockage disponible.

Muni de ces notations supplémentaires, ainsi que des contraintes logistiques détaillées ci-dessus, nous pouvons maintenant aborder la question de l'approche de résolution.

2.2 Approche de résolution

Notre approche est basée sur l'utilisation d'un algorithme génétique. Nous allons d'abord préciser de façon générale son fonctionnement, puis nous nous pencherons plus précisément sur ses caractéristiques.

2.2.1 Fonctionnement général d'un algorithme génétique

Le concept à la base de l'algorithme génétique est la méthode de recherche inspirée par les mécanismes d'évolution génétiques. De nombreux opérateurs peuvent être utilisés, et nous les

détaillerons dans la section suivante. Nous nous sommes appuyés sur l'algorithme génétique de base, qui peut différer par certains aspects de ceux présentés dans la littérature, du fait des améliorations que nous avons pu y apporter. Le schéma de fonctionnement de notre algorithme est détaillé à la figure 2.2.

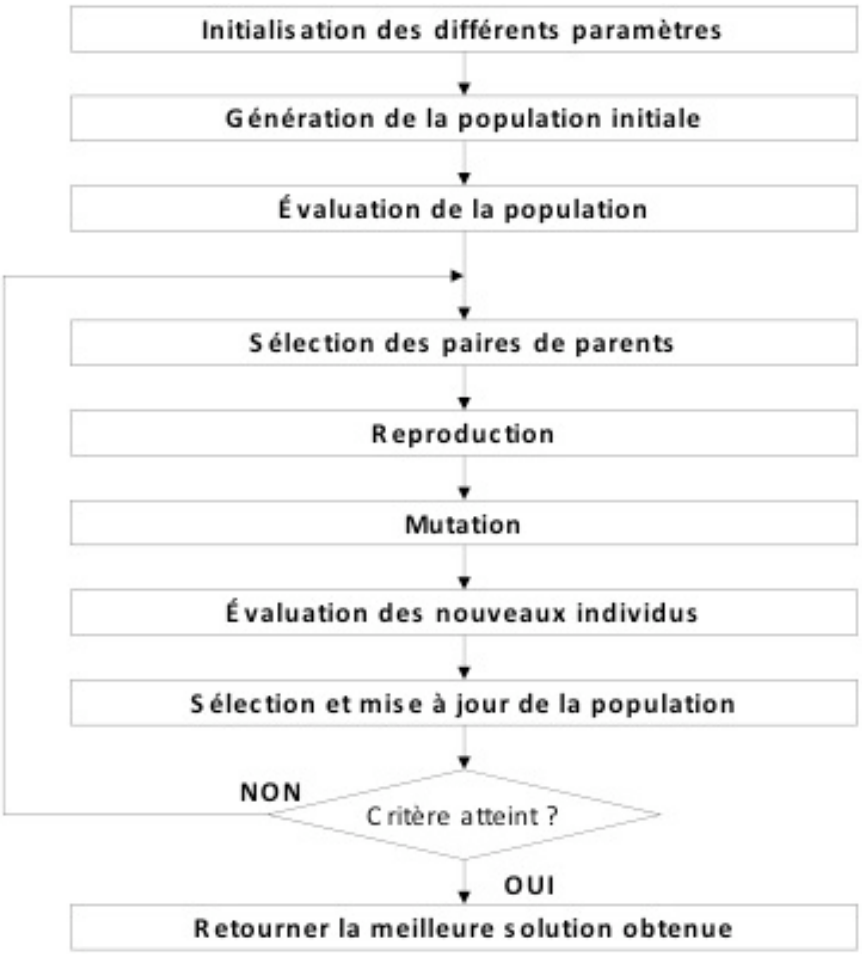


Figure 2.2 : Schéma de principe de l'algorithme génétique

Tout commence par l'initialisation des différents paramètres. Dans notre cas, ils sont au nombre de trois : la taille de la population, notée *POP*, la probabilité de croisement lors de la reproduction, notée C_p, et la probabilité de mutation M_p. Ces trois paramètres influent fortement sur le comportement de l'algorithme, et nécessitent une phase préliminaire de tests, afin de déterminer des valeurs satisfaisantes dans le cadre de l'étude. Le ou les critères d'arrêt sont aussi définis à cette étape. La population initiale est ensuite générée et évaluée. Cela permet d'apparier les parents en vue de l'étape de reproduction. Lors de cette étape, et selon le taux de croisement,

les chromosomes sont soit clonés, soit croisés, pour former de nouveaux chromosomes enfants. Ceux-ci pourront subir une mutation, puis ils seront évalués, pour être, selon leur évaluation, incorporés ou non à la population pour effectuer une nouvelle itération. Tant que le critère d'arrêt n'est pas atteint, les itérations continuent. À l'arrêt de l'algorithme, la meilleure solution obtenue est retournée.

2.2.2 Explication détaillée du modèle

Il est important de voir ici que de nombreuses méthodes, mais aussi de nombreux opérateurs différents, existent pour améliorer le fonctionnement de l'algorithme génétique. C'est ce qui fait sa puissance, mais cela le rend d'un autre côté difficile à paramétrer.

2.2.2.1 Initialisation

Lors de la première étape d'initialisation des paramètres, il faut fixer le ou les critères de terminaison (souvent le nombre de générations *GEN*), la taille de la population *POP* ainsi que les taux de croisement et de mutation C_p et M_p, respectivement. Des approches ont été développées, réévaluant la valeur de ces paramètres en fonction de l'avancement de la résolution, mais, comme première approche, nous avons préféré garder ces taux fixes de génération en génération, pour cette première approche expérimentale. Ce point constitue toutefois une voie de recherche prometteuse.

2.2.2.2 Génération de la population initiale

De manière générale, un individu est représenté par un chromosome, qui est composé de gènes. Selon le type d'encodage utilisé, chaque gène peut représenter une activité (Activity List-Based Encoding), un poids affecté à cette activité (Random Key-Based Encoding) ou une règle de priorité (Priority Rule-Based Encoding). La première technique considère les échéanciers et travaille directement dessus, ce qui a pour intérêt de simplifier la fonction d'évaluation, mais nécessite alors de relancer le processus de génération dès lors qu'un croisement ou une mutation engendrent un chromosome qui ne respecte plus les contraintes, ce qui alourdit le processus. La seconde méthode est plus souple. Elle permet de ne travailler l'essentiel du temps qu'avec des listes de priorités affectées aux activités, sans se préoccuper des contraintes inhérentes au problème durant les phases de croisement et reproduction, le décodage ne s'effectuant que lors

des phases d'évaluation. Enfin, la dernière méthode utilise une liste de règles de priorités pour sélectionner chaque activité au moment du décodage. Les deux dernières techniques assurent de ne travailler en tous temps qu'avec des solutions légales, ce qui est avantageux pour des problèmes fortement contraints. Pour notre approche, nous avons préféré privilégier la simplicité en retenant la seconde méthode.

2.2.2.3 Évaluation des individus

Cette étape, qui consiste à évaluer la qualité des différents individus, nécessite la construction préalable des échéanciers à partir des listes de priorité, selon deux schémas de génération différents. Le premier, le schéma de génération d'échéancier en série (Serial Schedule Generation Scheme), fonctionne par itération (figure 2.3). Toutes les activités dont les prédécesseurs ont déjà été planifiés sont considérées comme éligibles. Parmi elles, l'activité ayant la clef aléatoire la plus grande est sélectionnée et cédulée le plus tôt possible, tout en respectant les contraintes logistiques. Ainsi, par itération, l'échéancier est construit. La date de fin de la dernière activité donne la durée de l'ensemble du projet.

```
1  J := Ensemble des tâches
2  SJ := Ensemble des tâches planifiées
3  EJ := Ensemble des tâches éligibles
4  pour J faire
5       Mettre à jour EJ
6       Sélectionner une tâche j parmi EJ
7       Planifier j au plus tôt en respectant les contraintes
8       Mettre à jour SJ
9  fin
```

Figure 2.3 : Algorithme de génération sériel

Le deuxième schéma de génération est le schéma parallèle (Parallel Schedule Generation Scheme), qui travaille lui par période de temps, et non plus par itération. Ici, les activités dont les prédécesseurs ont été planifiés avant le dernier point temporel de décision t_g sont considérées comme éligibles. Une fois l'activité sélectionnée, un nouveau point de décision est considéré, placé à la date de fin la plus proche de l'ensemble de toutes les activités en cours d'exécution. Ce processus est détaillé à la figure 2.4. Pour notre étude, nous avons implanté le schéma sériel, plus simple à mettre en place.

```
1   J := Ensemble des tâches
2   JIP := Ensemble des activités en cours
3   SJ := Ensemble des tâches planifiées
4   EJ := Ensemble des tâches éligibles
5   Initialiser le point de décision t := 0
6   tant que SJ ≠ J faire
7        Mettre à jour t
8        Mettre à jour EJ
9        Sélectionner une tâche j parmi EJ
10       Planifier j au plus tôt en respectant les contraintes
11       Mettre à jour JIP
12       Mettre à jour SJ
13  fin
```

Figure 2.4 : Algorithme de génération parallèle

2.2.2.4 Processus de sélection

La sélection des paires de parents peut se faire selon différents mécanismes, basés là encore sur l'analogie avec la sélection naturelle. Nous avons recensé pas moins de quatre méthodes. La première est une méthode de sélection aléatoire (Random Method), appariant aléatoirement des individus différents, sans tenir compte de leur évaluation. La deuxième méthode est une méthode de sélection par rang (Ranking Method). C'est une méthode très simple où l'appariement s'opère entre les individus possédant les meilleures évaluations, et ainsi de suite, en descendant vers des évaluations plus faibles au fur et à mesure du processus. La méthode de sélection proportionnelle (Proportional Selection Method) aussi appelée Roulette Wheel Method, est dérivée de la méthode précédente, au sens où la sélection utilisera bien l'évaluation des individus, mais la sélection se fait selon la probabilité $P(\lambda)$:

$$P(\lambda) = \frac{f(\lambda)}{\sum_{\lambda' \in POP} f(\lambda')} \qquad (10)$$

où λ représente un individu et f la fonction d'évaluation. Cette méthode est basée sur le fonctionnement d'une roulette du jeu de hasard du même nom. Selon cette méthode, les individus possédant une bonne évaluation auront plus de chances d'être appariés. Enfin, la dernière méthode, qui comporte plusieurs versions dérivées, est la méthode de tournoi (Tournament

Selection Method), où les individus s'affrontent lors de tournois organisés aléatoirement et sont classés en fonction de cet affrontement. Deux tournois sont organisés, l'un pour les pères, l'autre pour les mères. L'appariement s'effectue alors par une méthode de rang, en se basant sur les résultats du tournoi.

2.2.2.5 Reproduction

Le processus de reproduction met en jeu le mécanisme de croisement, qui permet de renouveler la population en croisant et recombinant deux individus parents pour former deux nouveaux individus, selon la probabilité C_p. Pour cela, trois méthodes peuvent être utilisées : la méthode de croisement en un point (One-Point Crossover), celle de croisement en deux points (Two-Points Crossover) et la méthode de croisement uniforme (Uniform Crossover). Le premier opérateur sélectionne aléatoirement un « point de coupe » et échange les segments ainsi créés de part et d'autre du point de coupe chez les deux parents. Le deuxième opérateur agit de la même façon, mais en sélectionnant deux points de coupe (figure 2.5), ce qui permet d'affiner le processus. Enfin, l'opérateur de croisement uniforme construit les nouveaux individus en leur affectant aléatoirement un gène du père ou de la mère. Dans le cadre de notre recherche, nous avons implémenté l'opérateur de croisement en deux points.

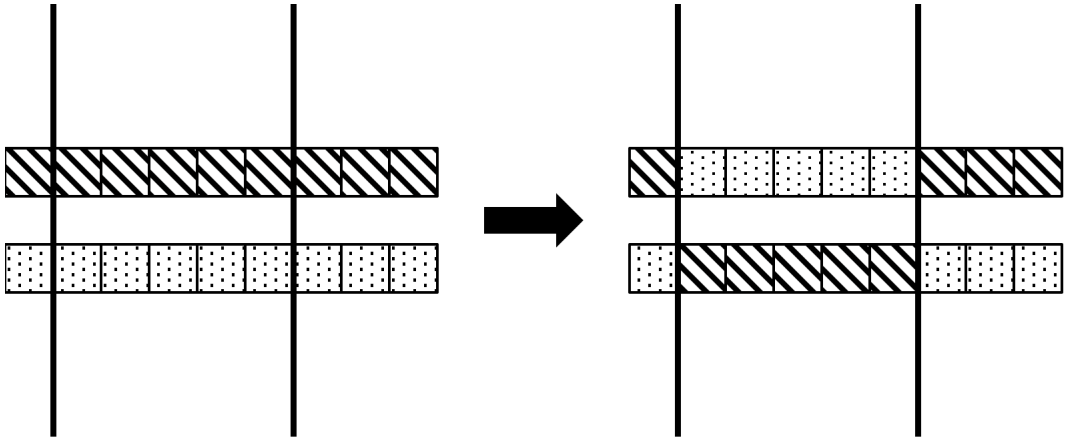


Figure 2.5 : Principe de fonctionnement du processus de croisement (deux points)

Les solution générées demeurent légales, du fait de l'utilisation de l'encodage par clef aléatoire, qui permet de ne travailler que sur des vecteurs composés des poids affectés à chaque activité, et donc de ne pas avoir à vérifier à chaque fois le respect des contraintes.

2.2.2.6 Mutation

La mutation peut s'effectuer de deux manières différentes. La première méthode, appelée mutation uniforme, consiste à permuter deux gènes voisins choisis aléatoirement sur un individu, selon une probabilité M_p. Le principe de fonctionnement du processus de mutation est détaillé à la figure 2.6. Au sein d'un même chromosome, deux gènes directement voisins peuvent être intervertis, sans que cela ait obligatoirement une incidence sur l'échéancier. La deuxième méthode intègre une composante aléatoire supplémentaire en permettant que les deux gènes permutés ne soient pas voisins. Pour notre étude, nous avons implémenté la méthode de mutation uniforme.

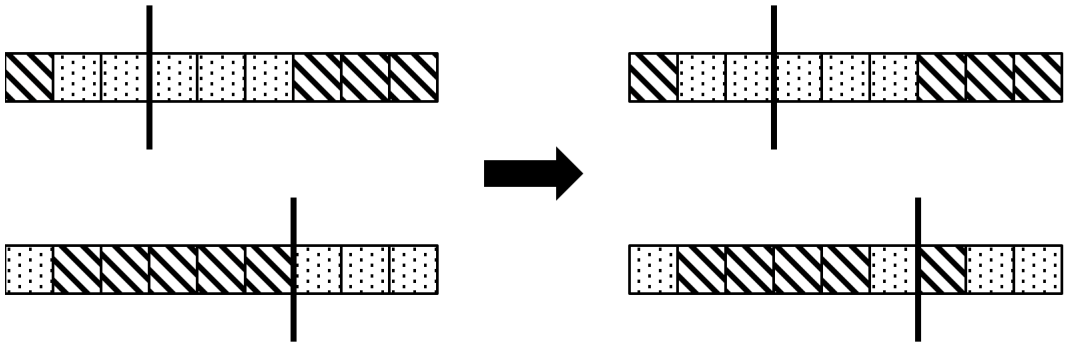


Figure 2.6 : Principe de fonctionnement du processus de mutation

2.2.2.7 Mise à jour de la population

Le processus de sélection des nouveaux individus et de mise à jour de la population se fait grâce à l'application d'un opérateur de sélection tel que défini plus haut. Nous avons opté pour l'utilisation d'une simple méthode de sélection par rang, qui remplace les mauvais individus de la population par de meilleurs, nouvellement créés.

2.2.2.8 Itération

Le processus est répété pour créer des chromosomes de meilleure qualité, jusqu'à ce que l'un des critères de terminaison soit atteint. Il en existe différents, que ce soit une limite de temps, un nombre maximum de génération, une proportion d'éléments de qualité dans la population... Dans notre cas, nous avons utilisé le nombre de générations.

2.2.3 Notation supplémentaire

De nouvelles notations sont apparues dans cette section, qui permettent d'appréhender notre approche de résolution génétique. Le tableau 2.3 les reprend et les synthétise, pour apporter une vision plus globale du processus.

Tableau 2.3 : Synthèse des notations utilisées dans l'approche de résolution

Symbole	Définition
λ	Individu / Chromosome / Échéancier
POP	Taille de la population de chromosomes
C_p	Taux de croisement
M_p	Taux de mutation
GEN	Nombre de générations / Critère de terminaison
$f(\lambda)$	Fonction d'évaluation de la qualité des individus
f_{best}	Meilleure solution au sein de la population
Random Key-Based Encoding	Opérateur utilisé pour générer les individus
Serial Schedule Generation Scheme	Schéma de génération utilisé lors de l'évaluation
Roulette Wheel Operator	Opérateur utilisé lors des étapes de sélection
Two-Points Crossover	Opérateur utilisé lors de la reproduction
Uniform Mutation	Opérateur utilisé lors de la mutation
Ranking Method	Méthode utilisée lors de la mise à jour de la population

2.3 Conclusion

À travers ce chapitre, nous avons explicité notre démarche, précisé notre approche de résolution et présenté notre modèle RCPSP hybride. Nous travaillerons ainsi sur un modèle RCPSP élargi intégrant de nouveaux paramètres et de nouvelles contraintes matériel, de stockage et de livraison. Pour résoudre le problème, l'algorithme génétique s'est avéré être la meilleure approche. Pour assurer une bonne performance de résolution, le paramétrage et l'analyse de l'impact des paramètres de l'AG seront pris en considération lors de l'implantation.

Dans le prochain chapitre, nous allons présenter les différents résultats obtenus dans nos expérimentations. Nous présenterons ainsi la façon dont nous avons utilisé nos données, dont nous avons validé notre paramétrage et l'utilisation de nos modèles, mais aussi l'exposition de nos résultats pour des modèles de plus grande envergure.

CHAPITRE 3 EXPÉRIMENTATIONS

Ce chapitre vise à présenter notre approche expérimentale. En premier lieu, nous aborderons la validation de notre paramétrage. Puis, après une première étude de cas, nous présenterons l'analyse de nos résultats pour des projets de plus grande envergure.

3.1 Approche expérimentale

Cette section présente l'approche expérimentale que nous avons adoptée en vue de résoudre notre problème. Ainsi, pour les besoins de l'étude, nous avons développé notre algorithme génétique en utilisant le langage de programmation Ruby, un langage de programmation dynamique orienté objet, qui permet le traitement des échéanciers sous forme d'objets et rend leur manipulation plus aisée. Les données de base qui ont été utilisées sont issues des travaux de Kolisch et Sprecher (1996) dans leur bibliothèque de projets PSPLIB, auxquelles nous avons rajouté nos contraintes logistiques. Finalement, les expérimentations ont été menées sur une plateforme munie d'un processeur Intel Core 2 Duo à 2,4 GHz avec 2 GB de mémoire RAM et sous le système d'exploitation Mac Os X 10.5.8.

3.1.1 Développement et implémentation

3.1.1.1 Ruby

Ruby est un langage de programmation créé par Yukihiro Matsumoto en 1995 sous licence libre, et développé depuis par l'auteur aidé par une communauté de passionnés. Principalement inspiré des langages de programmation Perl et Python, il est dit « orienté objet ». En effet, pour ce langage de programmation, tout élément est considéré comme un objet que l'on peut manipuler, nous y revenons dans la section suivante. Malgré cette forte connotation « objet », la programmation procédurale de type C ou Matlab est tout à fait possible.

Ruby est un langage qui possède plusieurs avantages. Tout d'abord sa facilité d'utilisation et son caractère dynamique en font un langage intuitif, léger et facile d'utilisation, et le rendent compétitif par rapport à d'autres langages (Flanagan, 2002). De plus, c'est un langage interprété, c'est-à-dire qu'il n'est pas nécessaire de le compiler avant de l'exécuter. Ensuite, c'est un langage totalement orienté objet, ce qui dans notre cas, nous permettra un traitement simplifié des

différentes instances (Montoya-Torres et al., 2010). Et enfin, c'est un langage sous licence libre, qui ne nécessite pas l'obtention d'une licence propriétaire.

La dernière version stable en date est la version Ruby 1.9.2 (2010). Mais pour le développement de notre algorithme, nous avons travaillé sous la version 1.9.1 (2009).

3.1.1.2 Développement orienté objet

Pour bien comprendre l'intérêt du développement orienté objet, il convient de le comparer avec le développement procédural. Celui-ci, plus simple au premier abord car ne nécessitant que peu de code pour produire des résultats, devient plus fastidieux à mesure que le problème se complexifie et que le nombre d'instances augmente. En effet, en ne définissant que des fonctions propres à chaque situation, il en devient peu évolutif. L'approche objet est plus structurée, va créer des classes et des sous-classes pour y ranger les objets. Ce qui apparaît comme fastidieux devient par la suite précieux, car il est beaucoup plus facile de s'y retrouver. De plus, manipuler des objets permet de maintenir le code à jour facilement (Montoya-Torres et al., 2010). Malgré qu'une structure plus complexe soit nécessaire le développement orienté objet a l'avantage d'assurer une compréhension plus aisée et de rendre le code plus facile à maintenir à jour et à manipuler (Cooper, 2007).

Dans notre cas, un autre facteur nous a poussé à nous tourner vers la programmation orientée objet. Les instances que nous avons à manipuler sont nombreuses et possèdent toutes les mêmes caractéristiques. En effet, un échéancier sera toujours composé du même nombre d'activités, et chaque activité possèdera un temps de début, une durée, des ressources et du matériel requis. Donc plutôt que de créer uniquement les fonctions de l'algorithme et de traiter les variables sans pouvoir les rappeler par la suite, car faisant partie du processus, il nous semble plus adéquat de créer des objets « chromosomes » possédant le même canevas de base, qui évoluerons au cours du processus et que nous pourrons rappeler au besoin pour en extraire les données qui nous intéressent.

Nous avons donc créé une classe « chromosome » (présentée figure 3.1), qui définit plusieurs caractéristiques que possèderont tous les chromosomes :

- La séquence des clefs aléatoires, définissant les priorités lors de la construction de l'échéancier

- La liste associée des temps de début d'activité, en respectant l'ensemble des contraintes
- La durée totale du projet, notée C_{max}

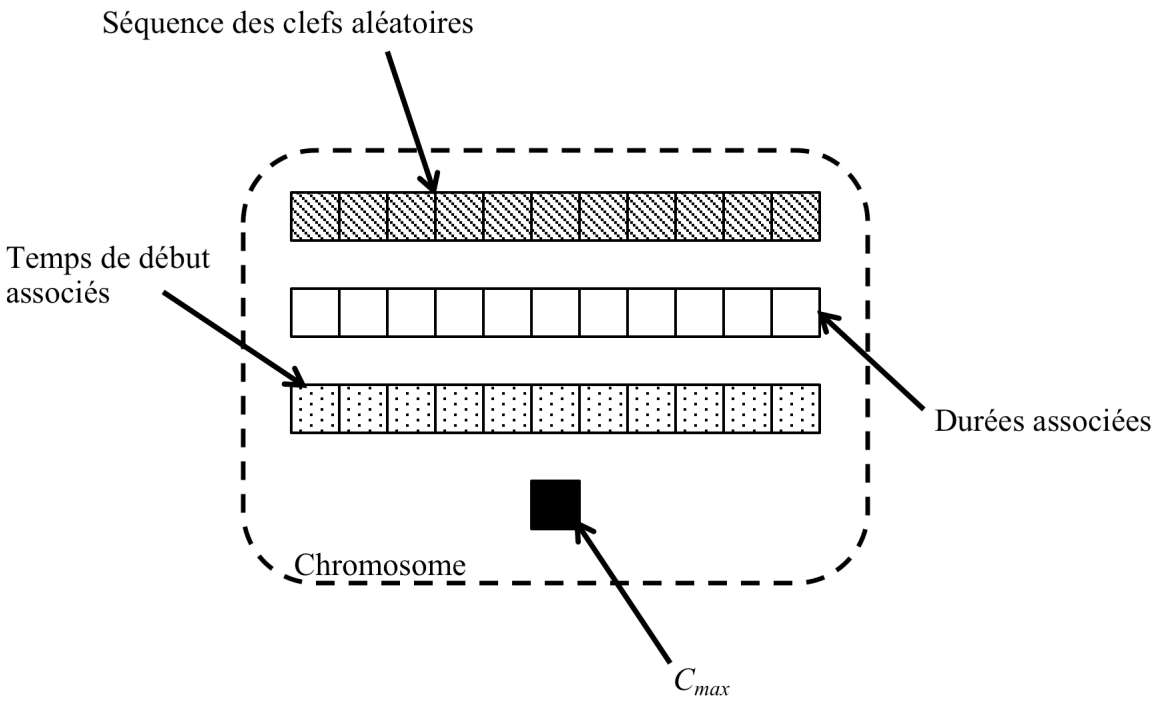


Figure 3.1 : Modèle d'un chromosome en développement orienté objet

3.1.1.3 Architecture

L'architecture d'un algorithme génétique, plus encore du fait de son orientation objet, est complexe. Dans un souci de compréhension, nous avons choisi de le diviser en plusieurs sous-programmes. Le premier sert d'interface avec l'ensemble des autres sous-programmes. Les autres ont été séparés selon que l'on considère le projet à traiter, l'algorithme génétique avec toutes ses fonctions, ou un chromosome, là aussi avec ses caractéristiques et ses fonctions. À noter que la fonction d'évaluation, du fait qu'elle s'applique sur les chromosomes, a été insérée dans le sous-programme « chromosome ».

3.1.2 PSPLIB

Pour créer des projets sur lesquels travailler, nous avons utilisé une bibliothèque libre développée par Kolisch et Sprecher en 1996 et mise à disposition sur internet, appelée PSPLIB. Adossé à

cette bibliothèque, les auteurs fournissent un programme de génération de projets, appelé ProGen. Les chercheurs ont ainsi la possibilité de consulter et d'utiliser des projets précédemment soumis, qui sont accompagnés des résultats optimaux de résolution, ou de générer eux-mêmes leurs propres projets, qu'ils pourront par la suite soumettre à la communauté. La bibliothèque s'élargit ainsi continuellement au fur et à mesure que de nouveaux résultats sont rendus accessibles.

Nous avons adopté une démarche à mi-chemin entre les deux. En effet, en vue de la validation préliminaire de notre modèle, nous avons tout d'abord utilisé un projet de PSPLIB existant et possédant une solution optimale connue. Nous avons alors rajouté manuellement les données de modélisation des contraintes logistiques à ce problème. Cela nous a permis d'avoir une base robuste de comparaison. Une fois la validation effectuée, nous avons utilisé ProGen et développé un programme de génération similaire, mais pour les contraintes de ressources, de manière à automatiser la génération de problèmes à contraintes matériel et de ressources. Nous avons ainsi pu travailler sur des projets de plus grande ampleur.

À terme, il nous sera possible de fournir une bibliothèque de projets similaire à PSPLIB, mais proposant un nouveau type de problèmes avec contraintes matériel et de ressources, se rapprochant plus de la réalité des projets de construction.

3.1.3 Exemple illustratif

Afin de tester notre algorithme et valider notre démarche, nous avons considéré une instance de projet tirée de PSPLIB. C'est un problème RCPSP composé de 30 activités et 4 ressources renouvelables. Nous avons manuellement défini les contraintes logistiques supplémentaires en se basant sur l'approche utilisée pour ProGen. Les deux paramètres N_{im} et L_{im} ont ainsi été définis.

Les données du problème sont présentées au tableau 3.1. Les activités notées 1 et 32 représentent les activités d'ouverture et de fermeture du projet, qui ne requièrent pas de ressources ni de matériel et ont des durées fixées à zéro unité de temps. Les successeurs sont donnés et doivent être convertis pour traiter les contraintes de précédence. La disponibilité des quatre ressources est donnée par $R = [12, 13, 4, 12]$.

Du matériel de cinq types différents a été considéré pour notre étude et leur réquisition a été fixée. Les délais de livraison ont aussi été générés. Deux zones de stockage $\gamma = 1$ et $\gamma = 2$ ont été créées. Les types de matériel 2 et 4 sont stockés dans l'espace 1 ($S_{M1} = \{2, 4\}$) et les types 1, 3 et

5 dans l'espace 2 (S_{M2} = {1, 3, 5}). Bien qu'il soit possible de choisir et modifier les capacités maximales de stockage, nous avons, dans un premier temps, décidé de les fixer chacune arbitrairement à 7 unités de surface $E_1 = E_2 = 7$. Enfin, le vecteur e_m qui définit le nombre d'unités de surface occupées par le matériel m est fixé défini par [1, 1, 1, 1, 1], ce qui signifie qu'un matériel m n'occupera à chaque fois qu'une unité de surface.

Tableau 3.1 : Données du projet

Activités	Durée p_i	Successeurs j	Ressources k : [R_i]	Matériel m : [N_{ij}]	Livraison L_{im}
1	0	{2, 3, 4}	[0, 0, 0, 0]	[0, 0, 0, 0, 0]	0
2	8	{6, 11, 15}	[4, 0, 0, 0]	[2, 0, 1, 0, 0]	3
3	4	{7, 8, 13}	[10, 0, 0, 0]	[1, 0, 0, 0, 0]	1
4	6	{5, 9, 10}	[0, 0, 0, 3]	[3, 1, 0, 5, 0]	0
5	3	{20}	[3, 0, 0, 0]	[0, 1, 0, 0, 0]	2
6	8	{30}	[0, 0, 0, 8]	[4, 0, 0, 1, 0]	3
7	5	{27}	[4, 0, 0, 0]	[1, 0, 2, 0, 0]	1
8	9	{12, 19, 27}	[0, 1, 0, 0]	[2, 1, 0, 4, 0]	2
9	2	{14}	[6, 0, 0, 0]	[0, 0, 3, 0, 0]	1
10	7	{16, 25}	[0, 0, 0, 1]	[0, 0, 0, 1, 0]	0
11	9	{20, 26}	[0, 5, 0, 0]	[0, 2, 0, 0, 0]	1
12	2	{14}	[0, 7, 0, 0]	[0, 1, 1, 0, 0]	2
13	6	{17, 18}	[4, 0, 0, 0]	[4, 0, 3, 3, 0]	1
14	3	{17}	[0, 8, 0, 0]	[0, 1, 0, 0, 0]	1
15	9	{25}	[3, 0, 0, 0]	[0, 0, 0, 1, 0]	1
16	10	{21, 22}	[0, 0, 0, 5]	[2, 3, 1, 0, 0]	0
17	6	{22}	[0, 0, 0, 8]	[0, 1, 0, 0, 0]	0
18	5	{20, 22}	[0, 0, 0, 7]	[3, 0, 1, 1, 0]	3
19	3	{24, 29}	[0, 1, 0, 0]	[1, 2, 4, 0, 0]	0
20	7	{23, 25}	[0, 10, 0, 0]	[0, 1, 0, 2, 0]	0
21	2	{28}	[0, 0, 0, 6]	[0, 1, 0, 1, 0]	2
22	7	{23}	[2, 0, 0, 0]	[4, 0, 0, 0, 0]	0
23	2	{24}	[3, 0, 0, 0]	[0, 1, 0, 0, 0]	1
24	3	{30}	[0, 9, 0, 0]	[0, 0, 2, 3, 0]	2
25	3	{30}	[4, 0, 0, 0]	[2, 3, 0, 0, 0]	1
26	7	{31}	[0, 0, 4, 0]	[1, 2, 3, 2, 0]	1
27	8	{28}	[0, 0, 0, 7]	[0, 3, 0, 1, 0]	1
28	3	{31}	[0, 8, 0, 0]	[2, 1, 0, 4, 0]	2
29	7	{32}	[0, 7, 0, 0]	[0, 2, 0, 1, 2]	1
30	2	{32}	[0, 7, 0, 0]	[3, 3, 1, 0, 2]	0
31	2	{32}	[0, 0, 2, 0]	[0, 4, 3, 0, 2]	1
32	0	{}	[0, 0, 0, 0]	[0, 0, 0, 0, 0]	0

3.2 Validation de la méthodologie

Étant donné la complexité de notre algorithme, il a été nécessaire de procéder d'abord à une étape de validation, qui nous a permis de déterminer les bonnes valeurs des paramètres de l'algorithme. En effet, comme présenté à la section 2.2, l'algorithme génétique dépend principalement de la taille de la population et des taux de croisement et de mutation. Réussir à définir une bonne configuration de ces trois paramètres est une étape importante pour arriver par la suite à une solution de bonne qualité.

3.2.1 Influence des paramètres

L'influence des trois paramètres est importante, en termes de qualité de la solution, mais aussi de performance de l'algorithme. Il convient donc de les choisir avec circonspection (Sastry et Goldberg, 2007).

Ainsi, une population importante fournira au final une meilleure solution, car l'algorithme couvrira un espace de recherche plus étendu et évitera une convergence vers un optimum local, mais dans le même temps, il augmentera le temps de résolution et la mémoire requise (Van Peteghem et Vanhoucke, 2010). Un taux de croisement important accélèrera l'échange de segments d'échéanciers intéressants au fil des itérations, mais présentera le risque de diluer le nombre de bonnes solutions dans la population. Inversement, un taux de croisement faible amènera une exploration moins importante de l'espace de recherche et donc une certaine forme de stagnation, mais améliorera le temps de traitement car il y aura moins d'appels de la fonction d'évaluation, les chromosomes ne se reproduisant que par clonage (Kim, 2009). Le taux de mutation introduira de la diversité, permettant une meilleure exploration de l'espace de recherche et évitant de tomber dans l'attraction d'un optimum local. Généralement, il n'est pas défini à une valeur importante, car il risquerait ainsi de détruire des séquences prometteuses (Kim, 2009).

3.2.2 Expérimentations

Afin d'examiner les effets de ces trois paramètres sur la performance de notre algorithme et ainsi de bien le configurer, nous avons fixé une valeur de départ par défaut de la taille de la population,

le taux de croisement et le taux de mutation, définis respectivement à 50, 0,5 et 0,1. Le processus est présenté à la figure 3.2 :

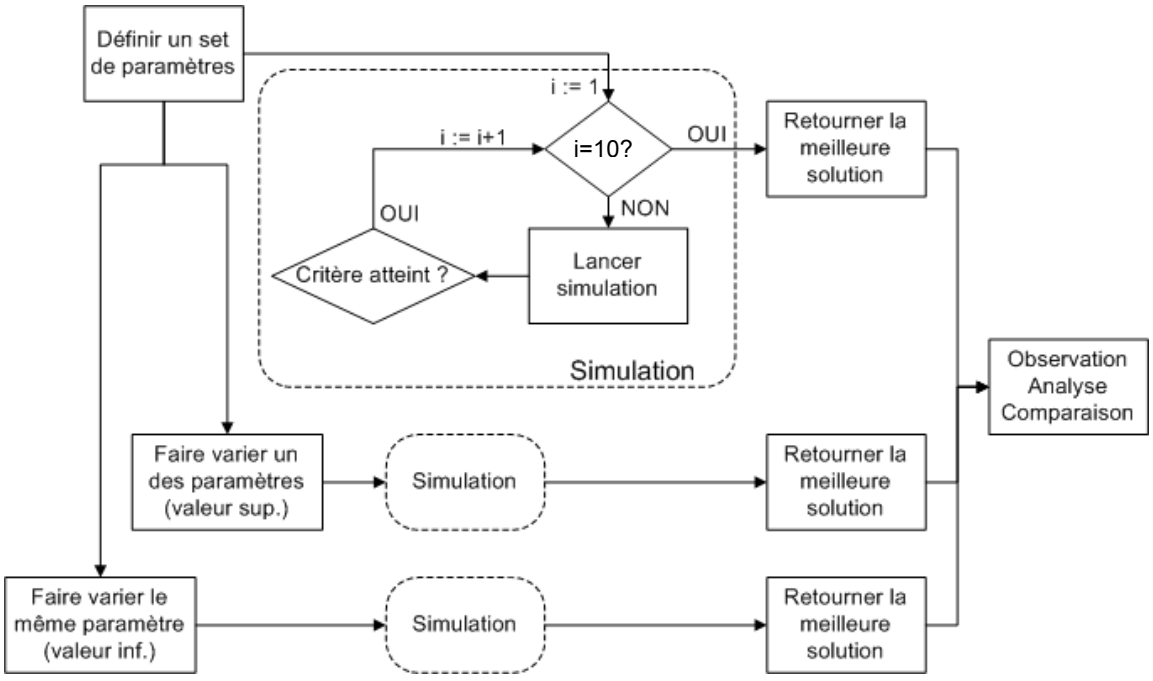


Figure 3.2 : Processus d'expérimentation pour le paramétrage de l'algorithme

Nous avons fait varier un paramètre à la fois, à une valeur supérieure et une valeur inférieure, les autres paramètres conservant leur valeur par défaut. À chaque fois, la progression dans la recherche d'une durée de projet minimale a été analysée, jusqu'à ce que le critère de terminaison soit atteint.

Une fois les paramètres bien définis, et dans l'optique de valider le bon fonctionnement de l'algorithme, nous avons voulu nous rapprocher du modèle RCPSP extrait de PSPLIB, dont la solution exacte est connue, et vérifier ainsi le bon comportement et la bonne convergence de notre modèle. Il faut pour cela diminuer l'importance des contraintes logistiques et faire tendre leur poids vers zéro. C'est effectué en augmentant fortement la capacité des zones de stockage afin de supprimer tout possible conflit d'entreposage. La contrainte d'approvisionnement demeure toutefois, mais n'ajoute aucune part variable. La solution renvoyée par l'algorithme doit donc tendre vers la solution optimale du problème RCPSP, au facteur approvisionnement près.

3.2.3 Résultats de la validation

La phase de validation a été divisée en plusieurs étapes. Nous nous sommes d'abord occupés du paramétrage, afin de trouver des valeurs pertinentes pour la taille de la population, le taux de croisement et le taux de mutation. Cela nous a aussi montré le bon comportement de notre algorithme, qui semblait bien converger, pour les différentes instances testées. Ensuite, à travers l'étude de l'impact des contraintes logistiques, nous avons pu valider le comportement de notre algorithme, en se rapprochant des conditions du problème RCPSP.

3.2.3.1 Paramétrage

Les résultats sont présentés dans les trois figures suivantes, sous la forme de graphiques montrant à chaque fois la convergence de la durée de projet selon que l'un des trois paramètres varie. Dans l'ensemble des cas, il y a bien convergence, en moins de 75 secondes. On observe des différences selon les variations des paramètres, que nous allons analyser par la suite. À chaque fois, la meilleure simulation, donnant une durée de projet de 76 unités de temps, est obtenue pour des paramètres fixés à $POP = 50$, $Cp = 0{,}5$ et $Mp = 0{,}1$.

Les observations vont dans le sens des hypothèses faites sur l'influence des différents paramètres. Tout d'abord, à la figure 3.3, l'effet de la variation de la taille de la population sur la convergence de la durée de projet semble bien corroborer le fait qu'il convient de trouver le juste milieu entre une population trop importante qui ralentirait la résolution et la situation inverse qui n'explorerait pas assez l'espace des solutions. La convergence pour le paramètre *POP* faible s'effectue en 50 secondes, avec peu de sauts, et assez graduellement. Cela s'explique par le fait que la population étant plus faible, de nombreuses possibilités n'ont pu être testées. Pour la convergence avec une population importante, celle-ci est très rapide durant les 30 premières secondes, puis ralentit, pour stagner à partir de 60 secondes. Cela peut s'expliquer par le fait que l'algorithme s'est retrouvé bloqué sur un minimum local et n'a pas réussi, malgré les nombreux individus, les croisements et les mutations, à s'en extraire. Toutefois, hormis cette stagnation, l'algorithme se comporte bien, comme le montrent les nombreux sauts de convergence.

Pareillement, on observe à la figure 3.4 que la meilleure simulation est celle effectuée avec pour valeur de C_p médiane. Là aussi, la convergence pour un taux de croisement faible s'effectue en peu de temps (40 secondes) puis stagne, pour les mêmes raisons que celles précitées. Et la

descente pour un fort taux de croisement est très rapide durant 20 secondes, puis stagne dès 50 secondes. Nous pouvons là aussi imaginer que l'algorithme est tombé dans un minimum local dont il n'a pas pu s'extraire, ce qui rend sont analyse peu pertinente.

Pour la figure 3.5 enfin, la convergence avec le plus fort taux de mutation est d'abord importante, puis ne progresse plus beaucoup au bout de 20 secondes. Cela peut s'expliquer par le fait que le fort taux de mutation au début de la simulation permet de faire de la recherche aléatoire et donc d'explorer un large espace de solution, d'où la progression rapide. Mais par la suite, la progression est bien plus lente à mesure que les meilleures solutions se raréfient, les mutations font perdre des séquences intéressantes. La descente pour le taux de mutation le plus faible converge régulièrement vers un minimum local, puis stagne aux alentours de 50 secondes. Cela est conforme aux prévisions qui faisaient état de la bonne tenue des simulations avec un faible taux de mutation, qui empêchait toutefois l'algorithme de s'extraire de minimums locaux.

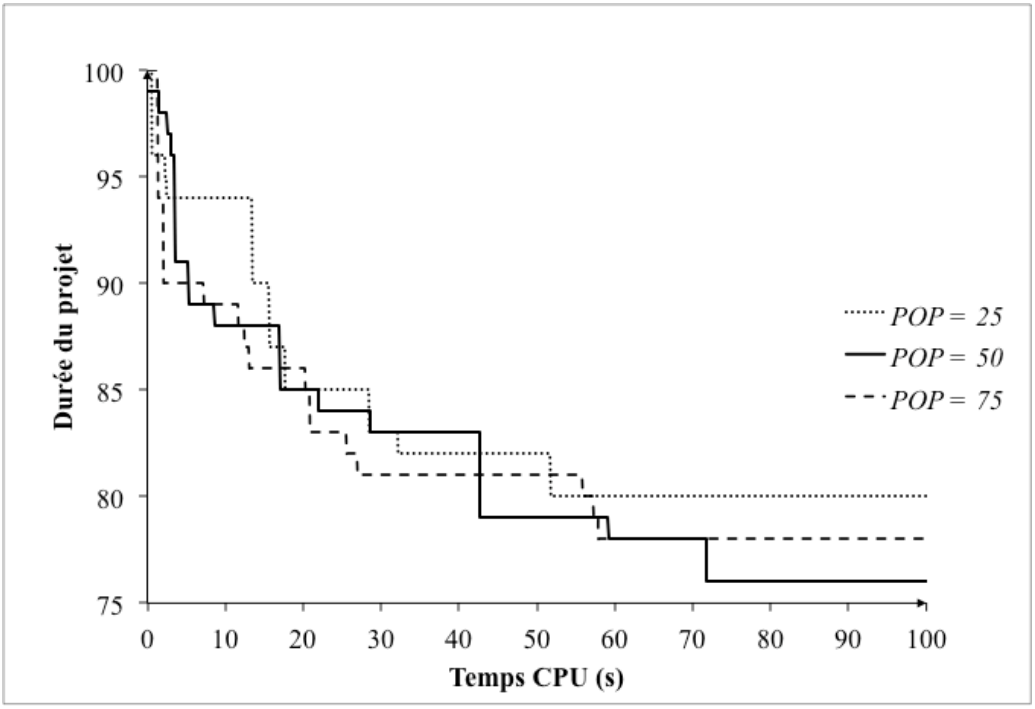


Figure 3.3 : Convergence de la durée de projet en fonction de *POP* (C_p = 0,50 et M_p = 0,10)

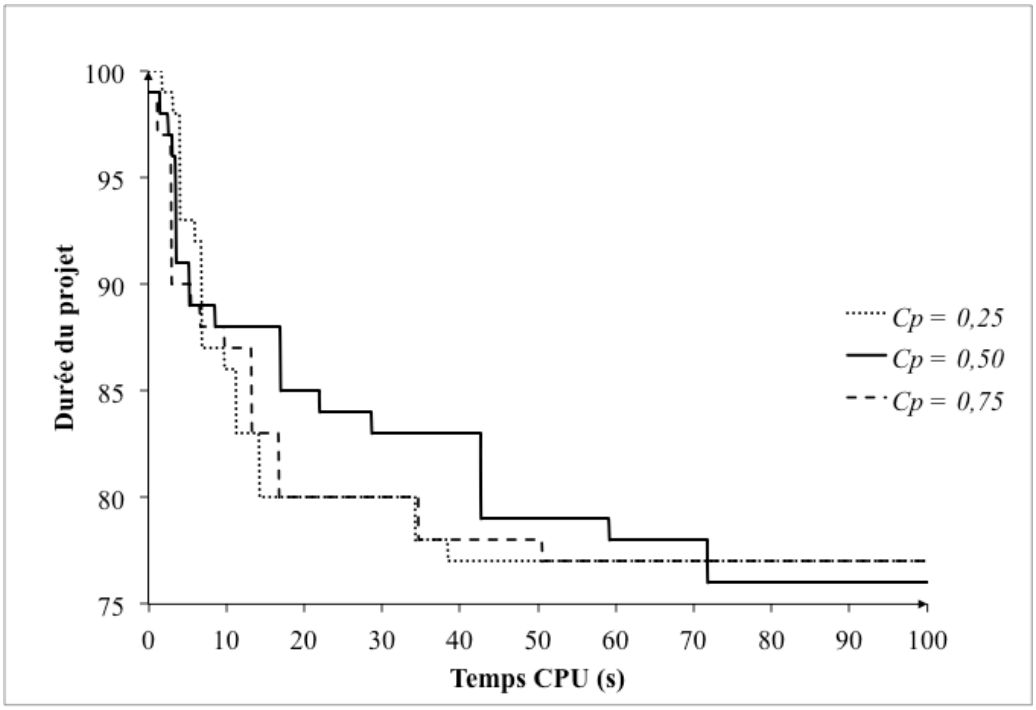


Figure 3.4 : Convergence de la durée de projet en fonction de C_p (POP = 50 et M_p = 0,10)

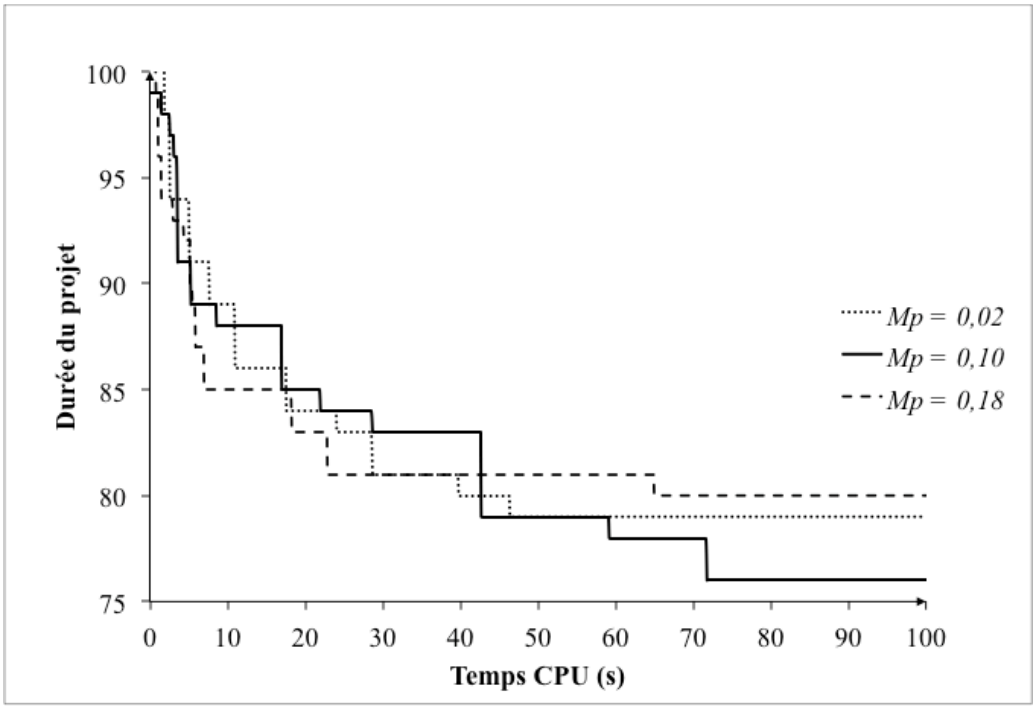


Figure 3.5 : Convergence de la durée de projet en fonction de M_p (POP = 50 et C_p = 0,50)

Au cours de cette étude de validation, nous avons bien retrouvé par simulation nos prévisions quant à l'influence des paramètres. Ainsi, pour la suite de notre étude, la configuration des paramètres de l'algorithme génétique ayant amené la meilleure solution (POP = 50, Cp = 0,5 et Mp = 0,1) a été appliquée. Nous notons toutefois que différentes phases temporelles se sont dessinées durant les simulations, montrant que l'impact des paramètres ne semble pas être le même selon l'avancement de la résolution.

3.2.3.2 Étude de l'impact des contraintes d'entreposage

Nous avons ensuite procédé à des simulations de variation de la capacité des zones de stockage, pour valider le bon comportement de l'algorithme. En diminuant la capacité de stockage, le problème est plus contraint. Tout le matériel ne peut être livré et stocké au même moment. En conséquence, certaines activités sont reportées, ce qui allonge la durée du projet. En augmentant la capacité de stockage par contre, nous nous rapprochons des conditions du problème RCPSP. La solution renvoyée par l'algorithme tend vers la solution optimale. Dans le projet original tiré de PSPLIB, la durée optimale dudit projet, sans tenir compte ni des contraintes de ressources ni des contraintes matériel, est de 38 jours. La durée optimale du problème RCPSP, obtenue par Demeulemeester et Herroelen (1997), est de 43 jours.

La figure 3.6 expose ces résultats. Les capacités de stockage (E_1, E_2) varient de (5, 5) à (24, 24). À mesure qu'elles augmentent, la durée totale du projet calculée diminue. Pour des valeurs dépassant (17, 17), la durée semble converger vers la durée optimale du problème RCPSP. Inversement, la durée totale du projet augmente rapidement lorsque la capacité de stockage diminue.

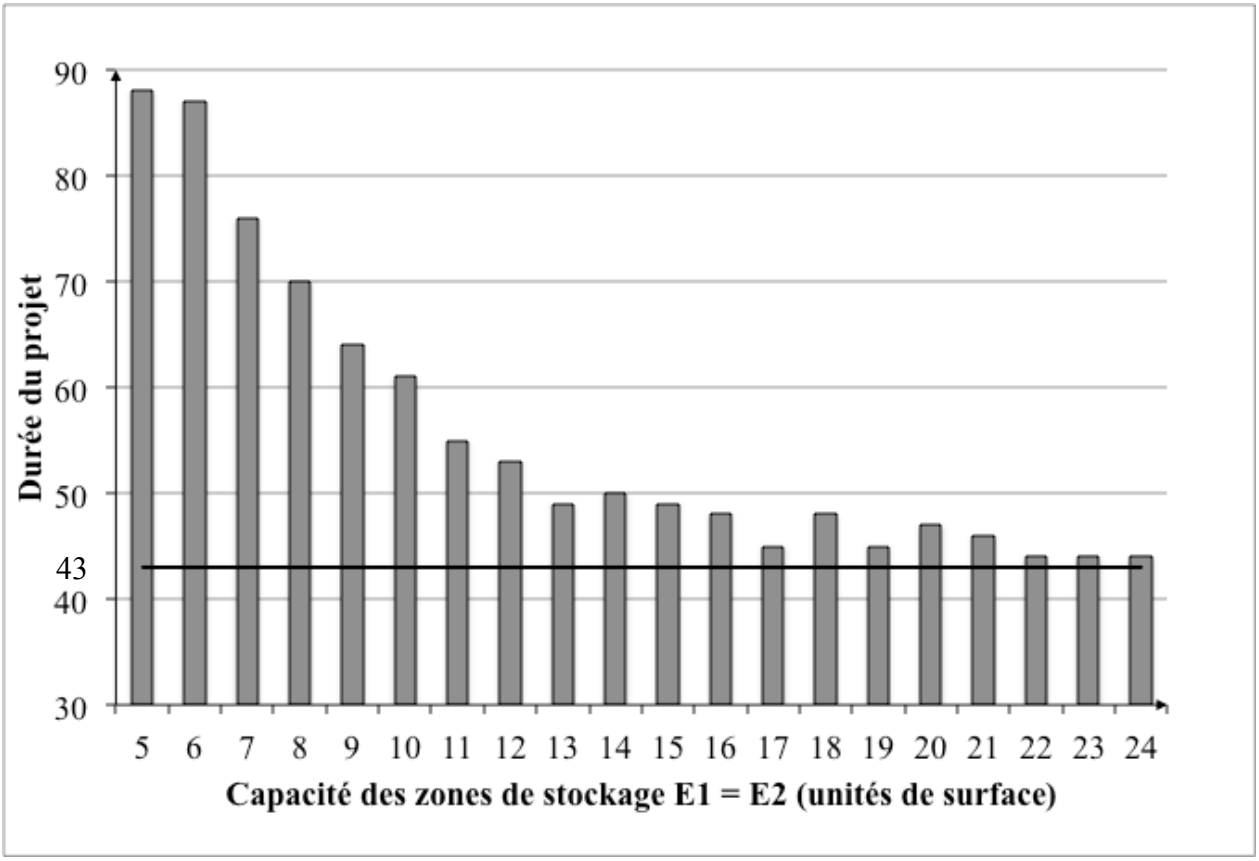


Figure 3.6 : Variation de la durée totale de projet en fonction de la capacité de stockage (30 activités)

3.3 Autres expérimentations

Nous avons aussi procédé à d'autres expérimentations, dont les résultats sont présentés dans les sections suivantes. Il est question ici de comparer la méthode métaheuristique avec deux autres approches : heuristique et exacte.

3.3.1 Comparaison avec une approche heuristique

Une première base de comparaison a été d'utiliser les résultats produits par Guèvremont (2009), qui propose une approche heuristique basée sur des schémas de génération série et parallèle avec 100 règles de priorité parmi les plus utilisées dans la littérature. Ces résultats (figure 3.7) montrent que la qualité des solutions obtenues par ce biais est très variable, allant de 87 unités de temps à 105 unités de temps, lorsque la capacité des zones de stockage est fixée à (E_1, E_2) = (7,

7). En comparaison, le meilleur résultat retourné par l'algorithme génétique pour le même projet et la même capacité de stockage est 76 unités de temps.

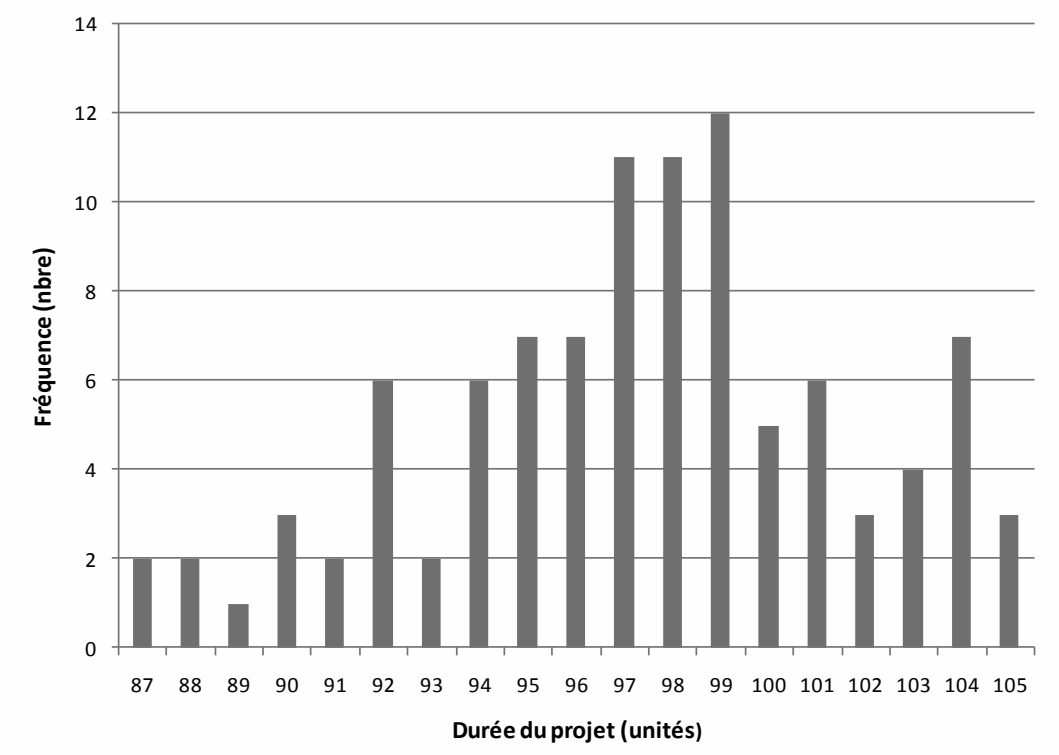


Figure 3.7 : Durée du projet avec contraintes ressources et matériel (Guèvremont, 2009)

De ces résultats, deux conclusions peuvent se dégager. Tout d'abord, assez sommairement, l'approche génétique semble offrir des résultats prometteurs, par rapport à une quelconque approche heuristique. Et ensuite, le fait que la plupart des logiciels commerciaux utilisent des approches heuristiques pour optimiser leurs problèmes d'ordonnancement implique, au vu de la qualité variable des résultats, que cela ne semble pas être la solution à envisager pour le traitement des contraintes matériel. La solution métaheuristique est donc plus que jamais pertinente.

3.3.2 Comparaison avec une approche exacte

Une méthode exacte a été développée sous AMPL utilisant le solveur CPLEX, afin de permettre un autre point de comparaison. Pour le même projet, la capacité des zones de stockage a cette fois été fixée à (E_1, E_2) = (12, 12). La figure 3.8 présente la convergence de la solution exacte d'une

part et celle de la meilleure solution retournée par l'algorithme génétique d'autre part, en fonction du temps.

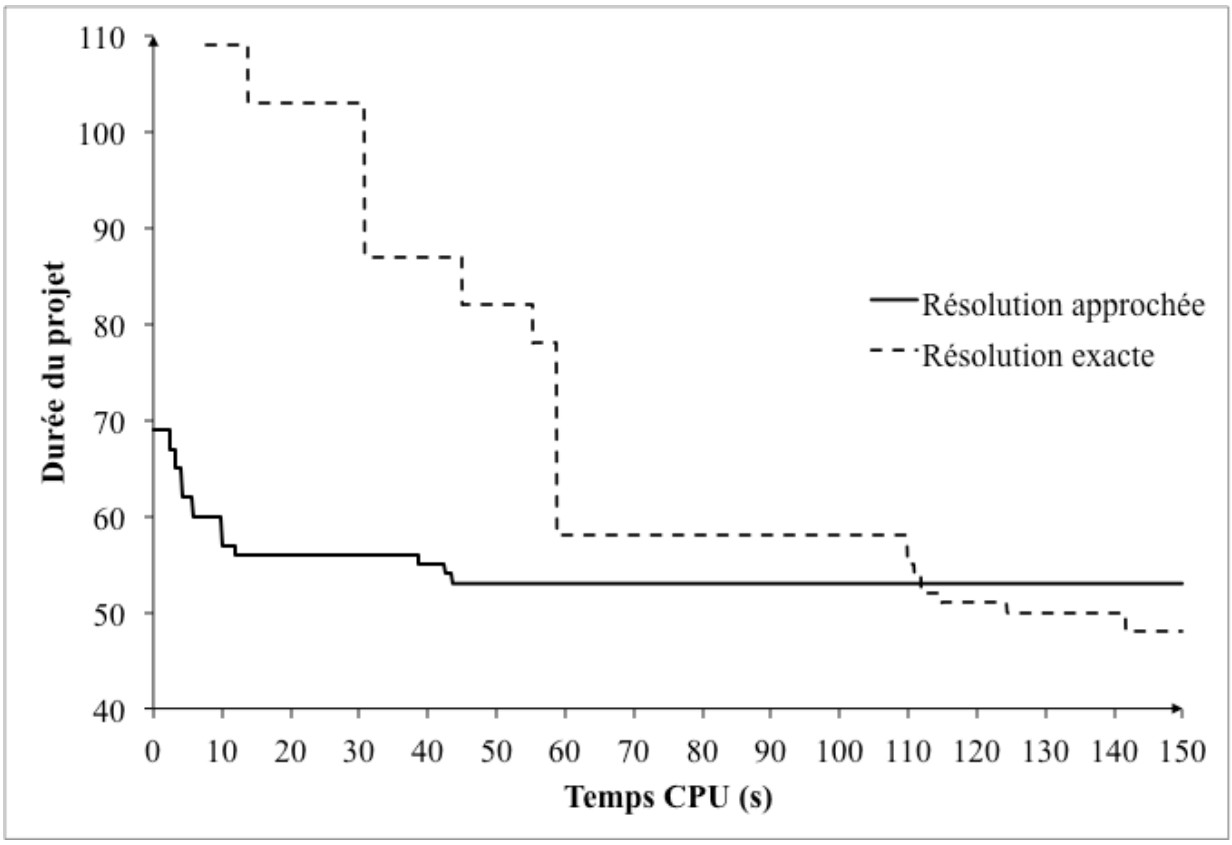


Figure 3.8 : Convergence comparée des solutions exacte et approchée

Alors que la solution exacte est retournée en moins d'une seconde lorsque les contraintes matériel sont absentes, leur ajout entraîne une complexification telle que la solution optimale, 48 unités de temps, est obtenue en 142 secondes. Au contraire, la solution offerte par l'approche métaheuristique converge vers 53 unités de temps en 45 secondes. Le point de croisement, à partir duquel l'approche exacte devient plus intéressante que la méthode approchée, intervient au bout de 110 secondes.

Nous n'avons pu parvenir à améliorer le paramétrage du « branch and bound » pour l'approche exacte à temps pour mener une campagne de test sur des projets plus contraints ou de plus grande taille, mais il apparaît envisageable que la complexité du projet augmentant, le point de croisement se déplace loin dans le temps. L'intérêt d'utiliser des métaheuristiques, et plus

particulièrement un algorithme génétique, transparaît ici plus clairement. Dès le lancement de la simulation, du fait de la population de solutions, l'algorithme génétique retourne une solution de bonne qualité. Et de plus il « possède » rapidement une population valable sur laquelle travailler. Au contraire, la résolution exacte est tributaire de la solution initiale, qui va devoir être améliorée petit à petit. De plus, le fait de devoir parcourir l'arbre des solutions et le temps pris par la recherche d'une solution entière rallongent la durée de simulation.

3.3.3 Application à des projets de plus grande envergure

3.3.3.1 Étude de l'impact des contraintes d'entreposage

Afin de se placer dans des conditions plus proches de la réalité, la dernière campagne de tests s'intéresse à l'extension des simulations à des projets de plus grande taille. Pour cela, des projets de 60, 90 et 120 tâches ont tout d'abord été extraits de PSPLIB. Afin de procéder à l'ajout des contraintes logistiques, un générateur basé sur les travaux de Kolisch et al. (1995) sur ProGen a été développé, pour créer aléatoirement les contraintes et permettre de contrôler les paramètres de génération. Une fois générées, les nouvelles contraintes logistiques sont greffées aux différents projets.

Le modèle n'étant plus le même, il eût été nécessaire de refaire le paramétrage de l'algorithme, afin d'assurer une bonne qualité de résolution pour l'algorithme. Mais l'optimisation n'étant pas fondamentale ici pour observer le bon comportement de l'algorithme, et dans un souci de simplicité, nous avons supposé que la configuration des paramètres effectuée pour des projets de faible amplitude reste toujours valable à mesure que leur taille augmente. Hartmann (2001) explique d'ailleurs que les résultats de configuration sont identiques pour les projets de faible et de grande envergure. Toutefois, la complexité ayant tout de même fortement augmenté, il a été décidé de limiter le nombre de générations pour rester à chaque fois dans des temps de simulations acceptables pour un utilisateur. Ce temps a ainsi été fixé à 100 secondes, valeur dont le choix sera validé dans la section suivante. En effet, au delà de cette valeur, le gain engendré ne vaut pas le temps dépensé.

Les trois figures suivantes représentent les variations de la durée de projet en fonction de la capacité de stockage, pour un projet de 60 (figure 3.9), 90 (figure 3.10) et 120 activités (figure 3.11).

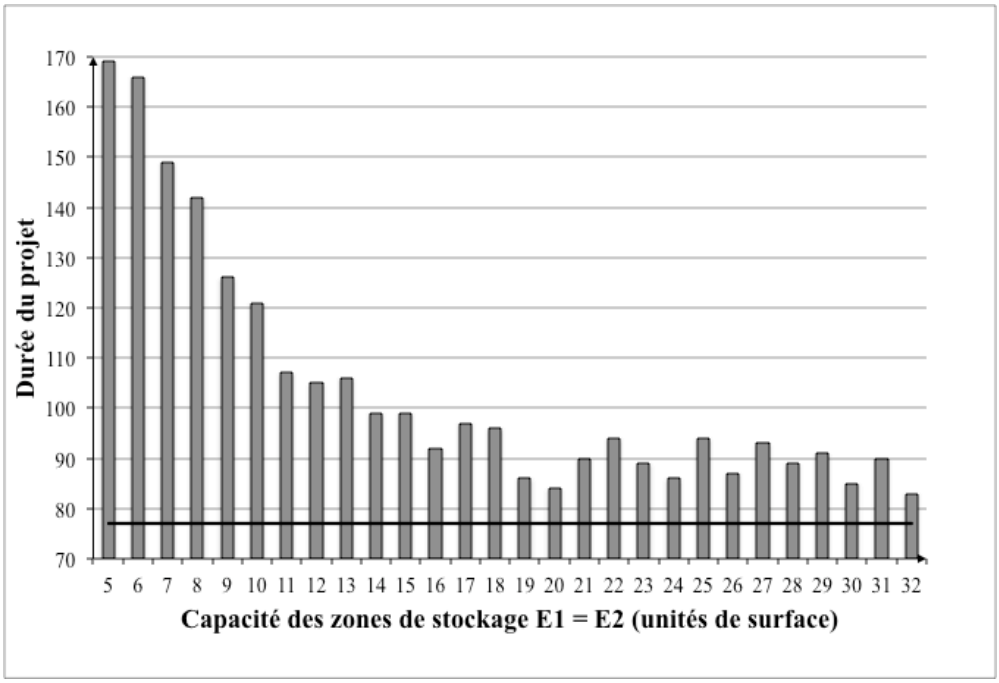


Figure 3.9 : Variation de la durée totale de projet en fonction de la capacité de stockage (60 activités)

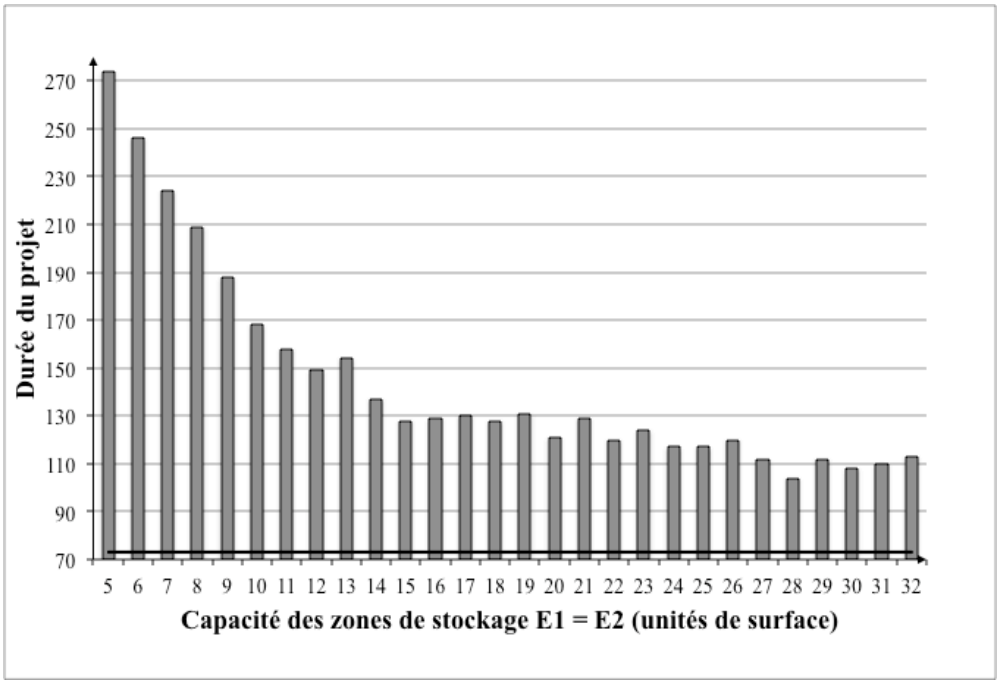


Figure 3.10 : Variation de la durée totale de projet en fonction de la capacité de stockage (90 activités)

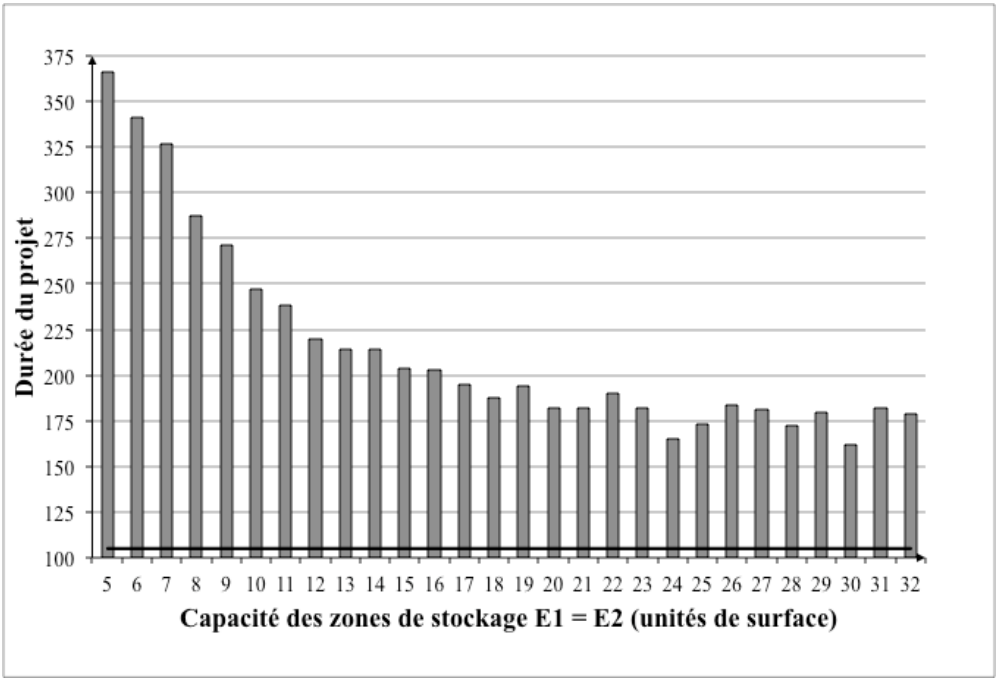


Figure 3.11 : Variation de la durée totale de projet en fonction de la capacité de stockage (120 activités)

On observe dans les trois cas que pour des valeurs de la capacité de stockage faibles, la courbe converge bien. Puis au fur et à mesure que les capacités augmentent, les courbes entrent dans une zone « variable », où les valeurs des durées de projet semblent globalement stagner, tout en étant particulièrement variables. Dans cette zone, alors que les capacités sont grandes et tendent à s'annuler, et où l'on devrait donc retrouver les valeurs du problème RCPSP, l'écart est important, proportionnellement à la taille du projet. Ainsi, on observe un écart d'environ 10 unités de temps pour le projet de 60 tâches, 40 unités de temps pour le projet de 90 tâches et 70 unités de temps environ pour le projet de 120 tâches.

L'analyse majeure que nous pouvons faire concernant ces résultats est que lorsque les contraintes sont fortes, l'espace de recherche est réduit et l'algorithme retourne une solution proche de l'optimale. D'où la bonne allure de la courbe de convergence pour des zones de stockage de faible capacité. Et à mesure que les capacités de stockage augmentent, les contraintes diminuent, et l'espace de recherche s'élargit. L'algorithme, du fait de la limitation du nombre de génération,

ne parvient pas à couvrir un espace assez grand et retourne une solution assez loin de l'optimale. D'où la mauvaise allure de la courbe convergence lorsque la capacité des zones de stockage est assez grande pour considérer le problème comme purement RCPSP. Le problème est d'autant plus flagrant pour le projet de 120 tâches, ce qui est logique.

Il peut sembler étrange que même lorsque les capacités de stockage sont importantes et donc qu'il n'y a plus de contraintes logistiques, le temps de résolution soit aussi important. Cela doit être dû au fait que même si les contraintes sont annulées, les données matériel doivent néanmoins être traitées par la fonction d'évaluation, et le processus reste donc très lourd.

3.3.3.2 Application à un grand projet

Une application a été réalisée à partir d'un grand projet de 120 tâches, en reprenant la capacité des zones de stockage (E_1, E_2) = (7, 7). La limite de temps est ici repoussée à 500 secondes. Les paramètres sont les mêmes que dans la section précédente. Le résultat est présenté à la figure 3.12.

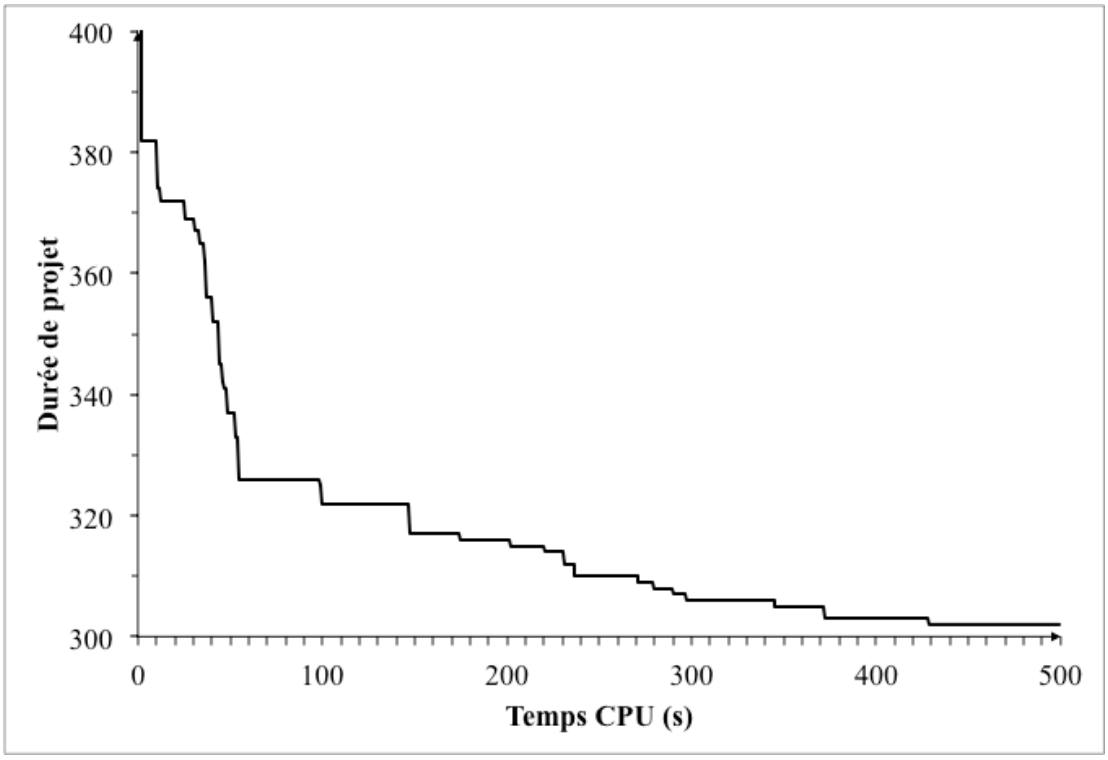


Figure 3.12 : Convergence de la durée de projet en fonction du temps (120 activités)

On observe une décroissance forte en début de simulation jusqu'à 100 secondes, où les solutions de bonne qualité sont encore nombreuses, et où il est donc facile d'améliorer le résultat. Celui-ci s'améliore de 80 unités de temps durant ces 100 premières secondes. Par la suite, et jusqu'à la fin de la simulation, la convergence est beaucoup moins rapide (20 unités de temps en 400 secondes).

Cela peut s'expliquer par le faible nombre de très bonnes solutions au sein de l'espace des solutions. L'algorithme n'a pas le temps de le parcourir entièrement, même en 500 secondes, pour pouvoir atteindre de telles solutions. Néanmoins, la convergence a une belle allure, et les résultats sont satisfaisants. Et enfin, cette simulation justifie le fait que nous ayons limité le temps de traitement à 100 secondes lors des précédentes simulations. L'amélioration postérieure à ces 100 secondes ne vaut pas le temps CPU dépensé, dans un contexte de simulations lors de la recherche, pour limiter le temps de simulation. Il est certain que la question se poserait plus fortement dans un contexte industriel, selon la taille des projets considérés et des sommes d'argent investies.

3.4 Conclusion

Les différentes expérimentations effectuées durant l'étude ont été détaillées dans ce chapitre. Tout d'abord l'étude de validation des paramètres, puis la comparaison avec des approches de résolution différentes, et enfin l'extension de la résolution à des projets de plus grande envergure. L'étude de validation a permis de définir les valeurs des trois paramètres de base définis pour l'algorithme. L'extension de la résolution à des projets plus grands a permis de montrer le bon comportement de l'algorithme face à des projets plus proches de situations réelles. Et la comparaison avec d'autres approches a montré la performance intéressante de l'algorithme génétique en terme de qualité ou de temps de résolution.

Nous allons maintenant mettre en lumière, à travers un chapitre de discussion, les différentes limitations auxquelles nous avons été confrontés, mais aussi les perspectives qui se dégagent de cette étude.

CHAPITRE 4 DISCUSSION

Le précédent chapitre a présenté les différents résultats obtenus, tant au niveau de la validation des paramètres et du bon comportement de l'algorithme que de son utilisation à plus grande échelle. Il est ainsi apparu que l'algorithme se comportait bien pour des projets de grande envergure, tout en offrant des performances intéressantes, de meilleure qualité que dans le cas de l'utilisation d'autres approches. Le présent chapitre va étendre la discussion pour aborder la pertinence et les limitations ainsi que les perspectives nouvelles qui s'ouvrent avec la production de cette étude.

4.1 Pertinence de l'étude

Nous avons vu, particulièrement dans le premier chapitre, que malgré une recherche abondante dans le domaine de la gestion de projets, les spécificités logistiques relatives aux projets internationaux continuent d'être ignorées dans les approches scientifiques. Il y a pourtant un réel besoin et une attente forte de la part des gestionnaires de projets pour des outils intégrant ces nouvelles contraintes. En ce sens, l'étude effectuée est pertinente car elle considère la problématique logistique et l'intègre pleinement dans le modèle développé. De plus, il est nécessaire de retourner une solution admissible dans un temps court, et vu la complexité du modèle, l'approche de résolution exacte n'est plus pertinente. Pour pallier cet obstacle, l'approche envisagée permet de contourner le problème de complexité en se concentrant sur la recherche de solutions approchées, ce qui amène un gain de temps indéniable et rend légitime le choix de l'approche métaheuristique. Enfin, la simplicité de la modélisation et le temps de résolution faible permettent d'envisager une intégration de notre modèle à un logiciel professionnel de gestion de projet. Cette perspective donnera certainement plus de valeur à notre étude et répondra aux besoins et attentes de nos partenaires.

Néanmoins, il est nécessaire de pondérer nos propos. En effet, notre modèle n'a été testé que sur des cas théoriques, nous n'avons donc pas eu de retours d'utilisateurs et nous ne savons pas la portée qu'il pourra avoir auprès d'eux, qui préfèreront peut-être s'en remettre préférentiellement à leur expérience. De plus, des logiciels professionnels existent déjà, mais l'optimalité de leurs résultats n'est pas prouvée, le processus de résolution demeurant inconnu. Enfin, nous avons dû,

pour effectuer cette étude, poser des hypothèses et faire certaines concessions, qui ont inévitablement limité le problème.

4.2 Limitations

Dans cette section, nous allons nous intéresser aux différentes limitations apparaissant au niveau du modèle, de la résolution et de l'analyse des résultats, qui sont rendues nécessaires par la nouveauté de la recherche.

4.2.1 Hypothèses de modélisation

Au cours de la recherche, diverses limitations sont apparues, qui ne nous ont pas permis d'aller aussi profondément que nous le voulions dans notre recherche. Les hypothèses que nous avons dû poser limitent clairement le problème et donc l'étude en général. Ainsi, pour la livraison tout d'abord, le processus de commande et de livraison a été modélisé « à la demande ». La réalité est souvent bien différente et les gestionnaires de projets doivent regrouper les livraisons de matériel selon une fréquence définie, par exemple hebdomadaire, pour réaliser ainsi des économies d'échelles, ou parce que l'éloignement du projet ne permet pas de faire autrement. Pour l'entreposage du matériel, le fait de considérer l'espace occupé par le matériel uniquement en terme de surface est limitant, au sens où l'on ne considère que l'entreposage au sol sans envisager de possibilité de rangement et d'optimisation du stockage en volume. De même, l'espace occupé par le matériel pour une activité l'est pour l'ensemble de son exécution, alors qu'au fur et à mesure de la consommation du matériel, l'espace de stockage devrait se libérer progressivement.

4.2.2 Paramétrage

Une des hypothèses majeure qui a été faite suppose que la configuration des paramètres de l'algorithme génétique pour des simulations sur des projets de 30 activités reste utilisable à mesure que la taille des projets augmente. Nous avons en effet effectué des tests de paramétrage au moment de la validation de notre algorithme, sur un projet de 30 activités, afin d'assurer une bonne combinaison des paramètres. Mais une nouvelle étude de paramétrage aurait dû être effectuée au fur et à mesure de la complexification du projet, car il est possible que les valeurs des trois paramètres de base aient été différentes. En effet, le temps de résolution augmente avec

la taille et la complexité des projets, et il serait peut-être opportun de réévaluer les paramètres pour ne pas ralentir la résolution, en sacrifiant un peu de la qualité au profit de la performance.

4.2.3 Analyse statistique

Le dernier type de limitation provient du traitement statistique des résultats. En effet, les procédures d'analyse et de sélection des résultats n'ont jamais été clairement indiquées dans la littérature, et il a été difficile de dégager un consensus clair sur le mode de sélection des résultats, entre par exemple une moyenne de *n* simulations ou la meilleure des *n* simulations. Cette dernière solution a été privilégiée, car elle mettait mieux en évidence la présence de paliers de stagnation et reste plus proche d'un usage courant.

4.3 Améliorations envisageables

L'étude, de par sa complexité et ses multiples facettes, présente plusieurs points limitants. Cela mérite une réflexion pour envisager d'éventuelles extensions et améliorations, qui concernent principalement l'optimisation du modèle et de l'algorithme génétique.

4.3.1 Amélioration du modèle de contraintes

4.3.1.1 Livraison

Le modèle développé l'a été dans un souci de simplicité volontaire, afin d'avoir un modèle de base et une approche complète de résolution. Mais il serait bon de s'approcher plus de la réalité en envisageant par exemple des livraisons groupées hebdomadaires, ou de prendre en compte de fournisseurs multiples. Certains compagnies utilisent maintenant des systèmes passant par des clusters régionaux de fournisseurs dont il serait bon de s'inspirer, comme SNC-Lavalin avec son système Global Procurement System (GPS), qui recense l'ensemble des besoins en matériel dans une région pour passer une commande globale et réaliser ainsi des économies d'échelle (SNC-Lavalin, 2008).

4.3.1.2 Entreposage

Pour l'entreposage, deux problèmes apparaissent : le stockage en volume et la libération de l'espace requis par une ressource consommée. Le modèle actuel mériterait d'être actualisé, pour d'une part prendre en compte différents types de matériaux pouvant être entreposés dans

différentes zones de stockage en fonction de leur volume, et d'autre part envisager que la consommation d'une ressource matérielle permette de libérer de l'espace de stockage au fur et à mesure de l'avancement de l'activité. Cela rapprocherait le modèle de la réalité et permettrait de céduler certaines activités plus tôt dans le projet. Il faudrait se rapprocher alors des systèmes dits 4D, évoqués précédemment (Akinci et al., 2002; Guo, 2002).

4.3.2 Amélioration de l'algorithme de résolution

4.3.2.1 Mécanisme de prédation

Des améliorations peuvent aussi être apportées à l'algorithme génétique, au niveau de son mode de fonctionnement ou de ses paramètres. Saenz de Ugarte et al. (2009) proposent par exemple de rajouter un système de prédation afin d'éliminer certaines des solutions qui ne satisfont pas à un critère statique ou dynamique. Une probabilité de prédation est ainsi définie pour chaque chromosome, qui risque de disparaître s'il ne satisfait pas à certains critères.

4.3.2.2 Mémoire cache

Une « mémoire cache » peut aussi être implantée, pour éviter la redondance consistant à tester à nouveau des solutions qui ont déjà été évaluées (Saenz de Ugarte et al., 2009). Cette stratégie n'est intéressante que si le temps de recherche dans la mémoire est inférieur au temps d'évaluation, ce qui est le cas pour notre problème, la fonction d'évaluation consommant beaucoup de temps CPU.

4.3.2.3 Paramétrage variable

Une dernière amélioration concerne la gestion dynamique des paramètres (Sastry et al., 2004). En effet, alors qu'au début de la simulation il est important d'avoir une population de grande taille pour explorer rapidement un grand espace de solutions, cette nécessité devient inverse à mesure que la simulation avance et que l'espace est couvert, et une population de plus faible taille en fin de simulation pourrait être importante. De la même manière, il est possible de travailler sur les taux de mutation et de croisement pour faire varier leur valeur durant la simulation (Sastry et Goldberg, 2007). La gestion dynamique peut ou bien s'effectuer de façon semi-automatique, les paramètres adoptant des valeurs fixées au cours du temps, ou bien de manière totalement

automatique, avec un algorithme évolutionnaire dédié à l'optimisation des paramètres au fil des itérations (Talbi, 2004).

4.4 Perspectives

4.4.1 Intégration ERP

Enfin, cette étude nous ouvre un certain nombre de perspectives et d'opportunités de recherche intéressantes, qui ne manqueront sans doute pas d'être étudiées par la suite. L'une d'elle consisterait à travailler sur une intégration, même partielle, de l'algorithme de résolution métaheuristique à un système professionnel de gestion de projet, tel MsProject, Primavera, ou plus vraisemblablement, PM+, logiciel dont SNC-Lavalin est propriétaire. Il s'agirait ici de rajouter deux modules permettant de récupérer automatiquement les données d'un projet et d'effectuer un ordonnancement retournant un échéancier valable à l'opérateur.

4.4.2 Création d'une bibliothèque de projets

Une deuxième perspective serait la création, à terme, d'une bibliothèque de projets avec contraintes logistiques, sur le même principe que PSPLIB, et permettant à chacun dans la communauté scientifique de présenter ses résultats pour des problèmes extraits de cette bibliothèque. Le générateur de contraintes logistiques, pourrait, là aussi comme pour PSPLIB, être fourni pour permettre de générer ses propres projets sur lesquels travailler.

4.4.3 Hybridation

La dernière perspective, qui semble prometteuse, est celle consistant à associer deux métaheuristiques, pour optimiser le temps de résolution. Cela serait envisageable pour des projets complexes de grande envergure, où l'algorithme génétique serait lancé pour obtenir rapidement une solution de bonne qualité, qui serait ensuite bonifiée par un algorithme utilisant pas exemple la recherche tabou. Ou sinon pour effectuer une routine au sein de l'algorithme génétique pour optimiser ses paramètres ou l'un de ses opérateurs.

4.5 Conclusion

À travers ce chapitre de discussion, des sujets tel que la pertinence de l'étude, les limitations amenant des améliorations, ainsi que les perspectives nouvelles ont été abordés. L'étude se révèle pertinente, mais des hypothèses ont dû être posées pour simplifier la modélisation, ce qui a amené des limitations. Celles-ci pourront être corrigées par diverses améliorations, sur le modèle ou l'approche de résolution. Enfin, la recherche concernant ce sujet ne s'arrête pas là, et de nombreuses perspectives s'offrent à nous.

CONCLUSION

Cette étude portait sur le développement d'un nouveau modèle d'ordonnancement de projets internationaux intégrant des contraintes logistiques ainsi que sur le développement d'une approche de résolution dédiée. Elle a été entamée pour répondre à un besoin des gestionnaires de projets de posséder des outils performants et surtout plus proches de la réalité. En effet, malgré l'abondance de la recherche dans ce domaine, ces contraintes continuent à être ignorées et forcent les gestionnaires à se baser uniquement sur leur expérience pour réaliser manuellement les ordonnancements, ou à utiliser les méthodes heuristiques fournies, de qualité variable et imprévisible.

Pour répondre à ce besoin, différentes méthodes de modélisation et de résolution ont été envisagées. Néanmoins, l'envergure des projets internationaux étant particulièrement importante, tout comme leur complexité, il a été décidé de se focaliser d'une part sur un modèle simple tenant compte des contraintes, et d'autre part de privilégier une méthode de résolution approchée. Le modèle retenu est un modèle RCPSP classique tenant compte de contraintes logistiques :

- Des contraintes de livraison du matériel, forçant celui-ci à être livré quelques jours avant le début de l'activité pour laquelle il est nécessaire.
- Des contraintes d'entreposage, définissant des zones de stockage, et créant des situations de conflit pouvant retarder le démarrage de certaines activités.

Au niveau de la résolution, une méthode approchée a été préférée à une méthode exacte, du fait de la complexité des problèmes obtenus par l'intermédiaire de notre modèle. Parmi ces méthodes approchées, les métaheuristiques se sont révélées être les plus intéressantes, car retournant des résultats faisables de meilleure qualité que la plupart des règles heuristiques utilisées dans la littérature. Ce sont donc vers elles que nous nous sommes tournés. Et parmi ces méthodes, l'algorithme génétique a été choisi pour sa robustesse et son adaptabilité à un grand nombre de problèmes.

L'approche expérimentale a été construite de manière progressive, se concentrant tout d'abord sur un projet de faible envergure, afin d'effectuer une bonne configuration des paramètres et de valider le modèle de résolution. Cela a aussi permis de comparer l'approche métaheuristique avec des approches exactes et heuristiques, légitimant ainsi l'utilisation d'un algorithme génétique.

Ensuite, d'autres expérimentations ont été réalisées à plus grande échelle en augmentant le nombre d'activités. Les résultats sont intéressants, mais montrent que des ajustements sont encore nécessaires pour obtenir des solutions directement applicables dans l'industrie.

En effet, les limitations sont multiples, du fait des hypothèses qui ont dû être posées, ou du modèle qui a été volontairement simplifié. Malgré cela, les perspectives d'avenir sont multiples. Tout d'abord, certaines améliorations peuvent être réalisées au niveau de l'algorithme en ajoutant de nouveaux opérateurs ou de nouvelles routines. Ensuite, du point de vue de la recherche, des opportunités s'offrent à nous, avec notamment l'intégration de la méthode aux ERP pour offrir un premier outil intéressant aux gestionnaires de projets, la création d'une nouvelle bibliothèque de projets pour permettre la collaboration de la communauté scientifique dans ce domaine et enfin l'étude de l'éventualité d'effectuer une hybridation de l'algorithme génétique avec une autre méthode métaheuristique pour optimiser le traitement des données dans le cas de projets de grande envergure.

Notre recherche possède une portée intéressante et présente deux intérêts majeurs. Tout d'abord, nos modèles reflètent mieux la réalité que certains autres proposés dans la littérature, et qui se contentent d'analyser un aspect du problème à la fois. Et ensuite, l'utilisation d'un algorithme métaheuristique permet d'obtenir des solutions en un temps court aisément utilisables dans les cas pratiques.

Notre contribution à la recherche est non négligeable. Nous avons tout d'abord produit une méthode de résolution utilisant un algorithme génétique aisément réutilisable et adaptable à un grand nombre de problèmes. Nous avons ensuite rédigé un article scientifique afin de présenter nos travaux (en annexe 1), article que nous avons présenté lors d'une conférence au Danemark. De plus, nous avons contribué à l'élaboration d'un générateur de contraintes, qui aboutira à la création d'une nouvelle bibliothèque de projets type PSPLIB.

L'étude réalisée est une bonne base pour la recherche future, dans ce domaine en perpétuelle évolution. En adaptant un modèle simple, mais néanmoins plus réaliste que ceux présents dans la littérature et en choisissant une approche de résolution aisément paramétrable, nous donnons une grande liberté d'adaptation et d'amélioration de notre méthode, ce qui, nous l'espérons, assurera une forte implication dans la recherche future sur ce sujet.

BIBLIOGRAPHIE

Agapiou, A., Clausen, L.E., Flanagan, R., Norman, G., & Notman, D. (1998). The role of logistics in the materials flow control process. *Construction Management and Economics*, *16*(2), 131-137.

Agarwal, R., Tiwari, M.K., & Mukherjee, S.K. (2007). Artificial immune system based approach for solving resource constraint project scheduling problem. *International Journal of Advanced Manufacturing Technology*, *34*(5-6), 584-593.

Akinci, B., Fischer, M., & Kunz, J. (2002a). Automated generation of work spaces required by construction activities. *Journal of Construction Engineering and Management*, *128*(4), 306-315.

Akinci, B., Fischer, M., Levitt, R., & Carlson, R. (2002b). Formalization and automation in time-space conflict analysis. *Journal of Computing in Civil Engineering*, *16*(2), 124-134.

Ala-Risku, T., & Kärkkäinen, M. (2006). Material delivery problems in construction projects: In possible solution. *International Journal of Production Economics*, *104*(1), 19-29.

Artigues, C. & Demassey, S. (2005) Gestion de projet, dans *Gestion de production et ressources humaines* : Méthodes de planification dans les systèmes productifs (Baptiste, P., Giard, V., Hait, A. & Soumis, F.), Chapitre 5, Presses internationales Polytechnique.

Azadivar, F., & Tompkins, G. (1999). Simulation optimization with qualitative variables and structural model changes: A genetic algorithm approach. *European Journal of Operational Research*, *113*, 169-182.

Baki, B. (2007). Planification et ordonnancement probabilistes sous contraintes temporelles. Ph.D. Université de Caen/Basse-Normandie, France.

Ballard, G., & Howell, G. (1998). Shielding production: Essential step in production control. *Newspaper of Building Engineering and Management*, *124*(1), 11-17.

Bertelsen, S. (1995) *Building logistics: A means for improvement of productivity in the building sector*. Copenhagen: Nelleman, Nielsen, & Rauschenberger A/S Consulting Engineers and Planners.

Bjornfot, A., & Jongeling, R. (2007). Application of line-of-balance and 4D CAD for lean planning. *Construction Innovation*, *7*(2), 200-11, UK: Emerald.

Blazewicz, J., Lenstra, J.K., & Rinnooy Kan, A.H. G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11-24.

Brucker, P., Knust, S., Schoo, A., & Thiele, O. (1998). A branch & bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *107(*2), 272–288.

Brucker, P., Drexl, A., Möhring, R., Newman, K., & Pesch, E. (1999). Resource constrained project scheduling problem: Notation, classification, models and methods. *European Journal of Operational Research*, *112*(1), 3-41.

Caron, F., Marchet, G., & Perego, A. (1998). Project logistics: Integrating the procurement and construction processes. *International Journal of Project Management*, *16*(5), 311-319.

Chaiy, S., Ciarlette, D., Cross, B., Manwani, S., Iandoli, L., Shore, B., et al. (2009). Developing a body of knowledge for the management of large-scale international science projects. *PICMET '09 - 2009 Portland International Conference on Management of Engineering & Technology*, 1481-1487.

Chaudhry, S.S., & Luo, W. (2005). Application of genetic algorithms in production and operations management: A review. *International Journal of Production Research*, *43*, 4083-4101.

Chen, W., Shi, Y.J., Teng, H.F., Lan, X.P., & Hu, L.C. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, *180*(6), 1031-1039.

Cheng, X., & Wu, C. (2006). Hybrid algorithm for complex project scheduling. *Computer Integrated Manufacturing Systems*, *12*(4), 585-9.

Cheng, M.Y., & O'Connor, J.T. (1996). ArcSite: Enhanced GIS for construction site layout. *Journal of Construction Engineering and Management*, *122*(4), 329-336.

Chong, C.S., Sivakumar, A.I., Low, M.Y.H., & Gay K.L. (2006). A bee colony optimization algorithm to job shop scheduling. *Proceedings - Winter Simulation Conference,, art. no. 4117838*, 1954-1961.

Cooper, P. (2007). *Beginning Ruby: From Novice to Professional*. Berkeley, Ca.: Apress. Tiré de Books 24x7: http://www.books24x7.com/marc.asp?bookid=21142.

Dawood, N., & Sriprasert E. (2006). Construction scheduling using multi-constraint and genetic algorithms approach. *Construction Management and Economics*, *24*(1), 19-30.

De Reyck, B., & Van de Velde, S. (1999). Informatiesystemen voor projectplanning: Meer communicatie dan optimalisatie. *Business Logistics*, *99*(10), 104-110.

Debels D., & Vanhoucke, M. (2005). The electromagnetism meta-heuristic applied to the resource-constrained project scheduling problem. *Artificial Evolution*, pp.259-270, Berlin: Springer.

Deckers, M. (2001). Exploratief onderzoek naar het gebruik van informatiesystemen voor project planning. Eindverhandeling. Department of Applied Economics, K. U. Leuven, Belgium.

Demassey, S., Artigues, C., & Michelon, P. (2005). Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, *17*(1), 52-65.

Demeulemeester, E.L., & Herroelen, W.S. (1997). New benchmarks results for the resource-constrained project scheduling problem. *Management Science*, *43*(11), 1485-1492.

Demeulemeester, E.L., & Herroelen, W.S. (2002). *Project Scheduling : a Research Handbook*. (49), International Series in Operations research and Management Science, Kluwer Academic Publishers.

Dominic, L. (2008). Projet de recherche et développement sur un outil de planification pour une entreprise manufacturière fonctionnant par projets, M.Ing., École Polytechnique de Montréal, Qc., Canada.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms, Ph.D., Politecnico di Milano, Italie, 1992.

Elbeltagi, E., Hegazy, T., Hosny, A.H., & Eldosouky, A. (2001). Schedule-dependent evolution of site layout planning. *Construction Management and Economics*, *19*(7), 689-697.

Farmer, J.D., Packard, N.H., & Perelson, A.S. (1986). The immune system, adaptation, and machine learning. *Physica D*, *22D*(1-3), 187-204.

Flanagan, D. (2002). *Java in a nutshell, fourth edition*. O'Reilly Media.

Georgy, M., & Basily, S.Y. (2008). Using genetic algorithms in optimizing construction material delivery schedules. *Construction Innovation*, *8*(1), 23-45.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research. 13*(5). 533-549.

Gu, J., Gu, X., & Jiao, B. (2008). A Coevolutionary Genetic Based Scheduling Algorithm for stochastic flexible scheduling problem. *7th World Congress on Intelligent Control and Automation, WCICA'08*, 4154-4159.

Guèvremont, M. (2009). Ordonnancement de projet sous contraintes de ressources, matériels et d'espaces d'entreposage. M.Ing. École Polytechnique de Montréal, Qc., Canada.

Guo S.J. (2002). Identification and resolution of work space conflicts in building construction. *Journal of Construction Engineering and Management*, *128*(4), 287-295.

Haque, M.E., & Rahman, M. (2009). Time-Space-Activity Conflict Detection Using 4D Visualization in Multi-storied Construction Project. *Visual Informatics: Bridging Research and Practice. Proceedings First International Visual Informatics Conference, IVIC 2009*, 266-278.

Hartmann, S. (1999). *Project Scheduling under Limited Resources*, Berlin: Springer.

Hartmann, S. (2001). Project Scheduling with Multiple Modes: A Genetic Algorithm. *Annals of Operations Research*, *102*, 111-135.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *207*(1), 1-14.

Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *127*(2), 394-407.

He, X., & Hu, W. (2007). An evaluation model for political risks of projects in international context. *Journal of Tongji University (Natural Science)*, *35*(11), 1572-7.

Herroelen, W.S. (2005). Project scheduling - Theory and practice. *Production and Operations Management*, *14*(4), 413-432.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Horman, M.J., & Thomas, H.R. (2005). Role of inventory buffers in construction labor performance. *Journal of Construction Engineering and Management*, *131*(7), 834-843.

Howell, G., & Ballard, G. (1997). Factors affecting project success in the piping function. *Lean Building*, 161-185, Rotterdam: Balkema.

Jacobus (1997). *PlantSpace Enterprise Navigator: User Guide*. Gaithersburg, MD: Jacobus Technology Inc., http://www.jacobus.com.

Jang, H.S. (2004). Genetic algorithm for construction space management. *KSCE J. Civil Eng.*, 8(4), 365-369.

Karan, E.P., & Ardeshir, A. (2008). Safety assessment of construction site layout using geographic information system. *Proceedings of the AEI 2008 Conference - AEI 2008: Building Integration Solutions*, *328*.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *1995 IEEE International Conference on Neural Networks - Conference Proceedings*, *4*, 1942-1948.

Kim, J.L. (2009). Improved genetic algorithm for resource-constrained scheduling of large projects. *Canadian Journal of Civil Engineering*, *36*(6), 1016-27.

Kini, D.U. (1999). Materials management: the key to successful project management. *Journal of Management in Engineering*, *15*(1), 30-34.

Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671-680.

Klein, R., & Scholl, A. (1999). Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, *112*(2), 322-346.

Kolisch, R. (1999). Resource allocation capabilities of commercial project management software packages. *Interfaces, 29*(4), 19-31.

Kolisch, R., & Hartman, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computationnal analysis, dans *Handbook on recent Advances in Project Scheduling* (Wergalrz), Ch. 7, Kluwer Academic Publishers.

Kolisch, R., & Hartman, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, *174*, 23-37.

Kolisch, R., & Sprecher, A. (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, *96*(1), 205-216.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, *41*(10), 1693-1703.

Koushki, P.A., & Kartam, N. (2004). Impact of construction materials they project time and cost in Kuwait. *Engineering, Construction and Architectural Management*, *11*(2), 126-132.

Krishnanand, K., Ghose, D. (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. *Proceedings of IEEE Swarm Intelligence Symposium*, 84-91.

Laedre, O., Austeng, K., Hougen, T.I., & Klakegg, O.J. (2006). Procurement roads in public building and construction projects. *Journal of Construction Engineering and Management*, *132*(7), 689-696.

Liberatore, M. J., & Pollack-Johnson, B. (2003). Factors influencing the usage and selection of project management software. *IEEE Transactions on Engineering Management, 50*(2), 164-174.

Lientz, B.P., & Rea, K.P. (2003). *International Project Management*, Academic Press, San Diego, USA.

Mallasi, Z. (2009). Towards minimizing space-time conflicts between site activities using simple genetic algorithm the best execution strategy. *Electronic Journal of Information Technology in Construction*, *14*, 154-79.

Mawdesley, M.J., Al-jibouri, S.H., & Yang, H. (2002). Genetic algorithms for construction site layout in project planning. *Journal of Construction Engineering and Management*, *128*(5), 418-426.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., & Bianco, L. (1998). An exact algorithm for the multiple resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, *44*, 714-729.

Montoya-Torres, J.R., Gutierrez-Franco, E., & Pirachicán-Mayorga, C. (2010). Project scheduling with limited resources using a genetic algorithm. *International Journal of Project Management*, *28*(6), 619-628.

Mucherino, A., & Seref, O. (2007). Monkey Search : A Novel Meta-Heuristic Search for Global Optimization. *AIP Conference Proceedings 953, Data Mining, System Analysis and Optimization in Biomedicine*, 162–173.

Ng, S.T., Shi, J., & Fang, Y. (2009). Enhancing the logistics of construction materials through activity-based simulation approach. *Engineering, Construction and Architectural Management*, *16*(3), 224-37.

Ning, X., Lam, K.C., & Lam, M.C.K. (2010). Dynamic construction site layout planning using max-min ant system. *Automation in Construction*, *19*(1), 55-65.

Okada, I., Lin, L., Gen, M. (2009). Solving resource constrained multiple project scheduling problems by random key-based genetic algorithm. *Electronics and Communications in Japan*, *92*(8), 25-35.

Osman, H.M., Georgy, M.E., & Ibrahim, M.E. (2003). A hybrid CAD-based construction site layout planning system using genetic algorithms. *Automation in Construction*, *12*(6), 749-764.

Pellerin, R. (1997). Modèle d'ordonnancement dynamique de projets de réfection. Ph.D. École Polytechnique de Montréal, Qc., Canada.

Pellerin, R., Sadr, J., Guevremont, M., & Rousseau, L.M. (2009). Planning of international projects with material constraints. *International Conference on Industrial Engineering and Systems Management*, Montreal, Canada.

Persona, A., Regattieri, A., & Romano, P. (2004). An integrated reference model for production planning and control in SMEs. *Journal of Manufacturing Technology Management*, *15*(7), 626-640.

Pheng, L.S., & Chuan, C.J. (2001). Just-in-time management of precast concrete components. *Journal of Construction Engineering and Management*, *127*(6), 494-501.

Pollard, J.M. (1978). Monte Carlo methods for index computation mod p. *Mathematics of Computation*, *32*(143), 918-924.

Polat, G., Arditi, D., & Mungen, U. (2007). Simulation-based decision support system for economical supply chain management of rebar. *Journal of Construction Engineering and Management*, *133*(1), 29-39.

Polat, G., & Arditi, D. (2005). The JIT materials management system in developing countries. *Construction Management and Economics*, *23*(7), 697-712.

Pritsker, A., Watters, L., & Wolfe, P. (1969). Multi-project scheduling with limited resources: a zero-one programming approach. *Management Science*, *16*, 93-108.

Retik, A., & Shapira, A. (1999). VR-based planning of construction site activities. *Automation in Construction*, *8*(6), 671-680.

Riley, D. (1994). Modeling the space behaviour of construction activities. Ph.D. Pennsylvania State University, University Park, PA, USA.

Riley, D.R., & Sanvido, V.E. (1997). Space planning method for multistory building construction. *Journal of Building Engineering and Management*, *123*(2), 171-180.

Saenz de Ugarte, B. (2009). Aide à la prise de décision en temps réel dans un contexte de production adaptative. Ph.D. École Polytechnique de Montréal, Qc., Canada.

Saenz de Ugarte, B., Pellerin, R., & Artiba, A. (2009). Enhancing Genetic Algorithm with predator mechanisms for On-line Optimization. *Proceedings of International Conference on Industrial Engineering and Systems Management 2009.*

Sastry, K., Goldberg, D.E., & Pelikan, M. (2004). Efficiency enhancement of probabilistic model building genetic algorithm. *Illinois Genetic Algorithms Laboratory*, Univeristy of Illinois, IL, USA.

Sastry, K., & Goldberg, D.E. (2007). Let's get ready to rumble redux: Crossover versus mutation head to head on exponentially scaled problems. *9th Annual Genetic and Evolutionary Computation Conference, GECCO 2007*, London, UK, 1380-1387.

SNC-Lavalin. (2008). Mining & Metallurgy. Canada: SNC-Lavalin. Rapport tiré de http://www.snclavalin.com/expertise.php?lang=en&id=12&sub=0/

Sprecher, A. (2000). Scheduling resource-constrained projects competitively at modest resource requirements. *Management Science, 46*(5), 710–723.

Talbi, E.D. (2004). Sélection et réglage de paramètres pour l'optimisation de logiciels d'ordonnancement industriel. Ph.D. Institut Nationale Polytechnique de Toulouse, France.

Tam, C.M., Tong, T.K.L., Leung, A.W.T., & Chiu, G.W.C. (2002). Site layout planning using nonstructural fuzzy decision support system. *Journal of Construction Engineering and Management*, *128*(3), 220-231.

Tang, Q., & Tang, L. (2008). Heuristic particle swarm optimization for resource-constrained project scheduling problem in chemical industries. *Chinese Control and Decision Conference, 2008, CCDC 2008,* 1475-1480.

Tommelein, I.D., Levitt, R.E., Hayes-Roth, B., & Confrey, T. (1991). SightPlan experiments: Alternative strategies for site layout design. *Journal of Computing in Civil Engineering*, *5*(1), 42-63.

Trautmann, N., & Baumann, P. (2009). Resource-allocation capabilities of commercial project management software:An experimental analysis. *2009 International Conference on Computers and Industrial Engineering, CIE 2009*, 1143-1148.

Tserng, H.P., Yin, S.Y.L., & Li, S. (2006). Developing a resource supply chain planning system for construction projects. *Journal of Construction Engineering and Management*, *132*(4), 393-407.

Valls, V., Ballestin, F., & Quintanilla, A. (2003). A hybrid genetic algorithm for the RCPSP, Rapport technique, Département de statistiques et recherche opérationnelle, Université de Valence, Espagne.

Van Peteghem, V., & Vanhoucke, M. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, *201*(2), 409-18.

Winch, G.W., & North, S. (2006). Critical space analysis. *Journal of Construction Engineering and Management*, *132*(5), 473-481.

Yang, X.S., & Deb, S. (2009). Cuckoo search via Levy flights. *Proceedings of World Congress on Nature & Biologically Inspired Computing, India* (pp. 210-214). USA: IEEE Publications.

Yourdon, E. (2003). *Death March, 2nd ed*. New Jersey: Prentice Hall, Inc.

Zhang, H., Li, X., Li, H., & Huang, F. (2005). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction*, *14*(3), 393-404.

Zouein, P.P., Harmanani, H., & Hajar, A. (2002). Genetic algorithm for solving site layout problem with unequal-size and constrained facilities. *Journal of Computing in Civil Engineering*, *16*(2), 143-152.

Zouein, P.P., & Tommelein, I.D. (2001). Improvement algorithm for limited space scheduling. *Journal of Construction Engineering and Management*, *127*(2), 116-124.

ANNEXE 1 – Article présenté en conférence

A GENETIC ALGORITHM FOR SOLVING MATERIAL SPACE AND RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEMS

Boucherit, Jean-Baptiste[1], Pellerin, Robert[1], Berthaut, François[1], Hajji, Adnène[2]

[1]Department of Mathematics and Industrial Engineering

Ecole Polytechnique de Montreal

C.P. 6079, succ, Centre-ville, H3C 3A7, Montreal

Canada

[2]Department of Operations and Decision Systems

Université Laval

Pavillon Palasis-Prince, 2325 rue de la terrasse, G1V 0A6, Quebec

Canada

E-mail: jean-baptiste.boucherit@polymtl.ca, robert.pellerin@polymtl.ca, francois.berthaut@polymtl.ca, adnene.hajji@fsa.ulaval.ca

Abstract:

The management of international projects is complex as it involves several sub-contractors and requires the transportation of lots of construction equipment and materials. In that context, long material delivery times and storage capacity constraints may lead to project delays. Indeed, in spite of all research efforts accomplished to develop project management tools, project plans taking into account space and equipment availability for the execution of tasks are mostly developed on the basis of planner's intuition and experience. The proposed model addresses this issue by formulating this problem as a scheduling problem with limited resources - resources can be either employees or work areas - and by defining material delivery constraints. The resolution of the formulated problem is performed by implementing a genetic algorithm to determine a feasible project plan. Results obtained are very promising and demonstrate the effectiveness of genetic algorithms to solve this class of problems.

Keywords:

Project planning, Logistics, Genetic algorithm, Space constraints, International project.

INTRODUCTION

Project scheduling concerns the allocation of limited resources to a finite set of activities over time in agreement with certain precedence requirements between activities. The problem of project scheduling under resource constraints, so called RCPSP (Resource-Constrained Project Scheduling Problem), has attracted great attention in recent years both from science and practice and various models were developed in an attempt to optimize resource allocation decisions.

However, the realization of projects can face other restrictions, which may affect the project schedule. For example, equipment and material storage capacity has a major impact on project scheduling and procurement activities. The management of building materials is a crucial activity as the costs of equipment and material represent a major portion of total project costs [9][1]. According to several studies, building materials generally account for 40 to 60 % of the complete project budget [8]. Besides, poor planning of storage space utilization can draw away a lack of necessary equipment to carry out tasks and postpone construction works, or in the opposite, to increase unnecessarily the inventory costs by accepting materials on the building site too much beforehand.

To prevent these problems, several researchers have promoted the use of dynamic project schedules instead of using a stable project plan [5]. This argument is in line with the theory of lean construction, which is based on the uninterrupted improvement of a plan established in the short term and just-in-time procurement. However, this approach is difficult to implement in an international context. The management of projects conducted in foreign countries are often more complex and more risky as they involve many sub-contractors and necessitate the transport of several construction equipment and materials. In addition, commercial project planning tools are incapable of resolving work space conflicts and ignore the importance of material delivery schedule and storage requirements in international construction projects.

Recognizing the difficulty of managing international projects, this research explores the use of a genetic algorithm to determine a feasible project plan under resource and storage space constraints. In the remainder of the paper, the problem notation and modeling are first presented followed by the presentation of the proposed genetic algorithm (GA) developed to solve it. An illustrative example and computational results are then presented. The paper concludes with a summary of the research and recommendations for future work.

THE PROPOSED APPROACH

Notation and Assumptions

The resource-space constrained project is defined by a set of activities (tasks), V, including two fictitious activities 0 and n+1 which correspond to the project start and the project end, respectively, with zero processing time. The project can be represented by an acyclic oriented graph $G = (V, E)$ described by a so called activity-on-node network where the nodes and the arcs represent the activities and the precedence relations, respectively. All (i,j) pairs in G are denoted by an arc $i \rightarrow j$ in the set of constraints E. Each activity i is described in terms of its processing time p_i.

S_i and C_i denote the start time and completion time of activity i, respectively. Consequently, $S = (S_1,\ldots,S_n)$ is a schedule and $C = (C_1,\ldots,C_n)$ is the vector of completion times. C_{max} represents the total completion time of schedule S.

Also, we assume that a pre-defined number of units of each resource, R_k, is available for every period of the planning horizon, T. R_{ik} represents the quantity of resource k required to conduct activity i. Resources are here classified as renewable resource. Moreover, a given activity i consumes materials, N_{im}. The quantity of materials used in the project is given by M while the minimum availability date of material m used in activity i is represented by L_{im}. Therefore, L_{im} indicates the number of time units that the material m should be present in storage area before the starting time of activity i.

Sa_γ indicates the set of material which can be stored together in the same storage area γ. N_{im} is defined as the amount of material m which is used in activity i and Es_γ indicates the total

capacity of storage area of type γ. The vector e_m is defined as the surface occupied by material m.

Problem formulation

Precedence constraints define timing requirements between activities. Equation (1) specifies that a predecessor activity must end before its successors may start. This finish-to-start constraint is the most common type of precedence constraints.

$$S_i + p_i \leq S_j \quad \forall (i,j) \in E \tag{1}$$

Resource constraints are expressed by (2). For cumulative and renewable resources constraints, this expression states that the amount of resource k used by activity i such that $i \in A_t$ should be less than the amount of available units of resource k. Available units of resources are here assumed to be constant.

$$\sum_{i \in A_t} R_{ik} \leq R_k \quad \begin{matrix} \forall k = 1, \dots, h \\ t = 1, \dots, T \end{matrix} \tag{2}$$

where $A_t = \{i = 1, \dots, n | S_i \leq t \leq S_i + p_i\}$

In international projects, material delivery can rarely occurred in a just-in-time manner. Planners usually request suppliers to deliver material before the actual start of activities in order to minimize the risk of late delivery or plan changes. This practice states that the material m used in activity i should be available L_{im} time units before the planning starting time (S_i) of activity i. Also, L_{im} must be strictly positive:

$$L_{im} \geq 0 \quad \begin{matrix} \forall i = 1, \dots, n \\ m = 1, \dots, M \end{matrix} \tag{3}$$

The total space occupied by all materials within the same storage area cannot exceed the capacity of that storage area at any time. Storage space constraints are expressed by (4). In our model, we suppose that the storage area capacity for a given material is constant. We also assume that storage area is used by material until the end of their related activities. Consequently, the storage area usage stays constant for activity i for the period $L_{im} + p_i$. Material storage usage is defined in terms of surface used and not in terms of volume.

The following constraint expresses that the storage area capacity must not be exceeded at any time.

$$\sum_{i \in A_t} \sum_{m} e_{im} \leq Es_\gamma \quad \begin{matrix} \forall i = 1, \dots, n \\ m \in Sa_\gamma \\ \gamma = 1, \dots, \Delta \end{matrix} \tag{4}$$

Finally, the objective function consists in minimizing the total project duration and can be expressed as follow:

$Min\ C_{max}$ such that $C_{max} \geq S_i + p_i \quad \forall i \in V$

Genetic algorithm approach

Various exact solution techniques, and heuristics and metaheuristics have been proposed so far in the literacy to solve the RCPSP. Due to the high dimension of the solution space combined with the difficulty to simultaneously allocate available manpower, machines, materials and other

resources, the RCPSP is well known as a NP-hard combinatorial optimization problem. Optimal procedures have been shown to be nearly computationally infeasible except for small size projects. As a result, heuristics and metaheuristics are extensively used to solve large project scheduling problems even though they do not guarantee optimal solutions. In fact, commercialized project management software packages are based on heuristics using serial or parallel algorithms with various priority rules that compute a precedence and resource feasible solution in a reasonable amount of time. However, the principal limitation of these heuristics arises from the poor accuracy of the obtained solution.

Metaheuristics usually consist in globally searching the optimal solution through a stochastic iterative algorithm. Among them, GA has proved successful in finding optimal or near-optimal solution for a variety of combinational optimization problems, including the RCPSP [2]-[3]. The basic concept behind GA is the searching strategy which is inspired by evolutionary mechanisms found in biological species. Genetic algorithms consist in determining an optimized solution by simulating evolution over a sufficient number of generations of individuals (or chromosomes), with reproduction, mutation and selection techniques. Each individual is composed of genes which symbolize the tasks. The principle of selection is the survival of the fittest individuals (with the best objective function values). The following sections detail the features of the GA applied in this study. A framework of the model is illustrated in Figure 1.

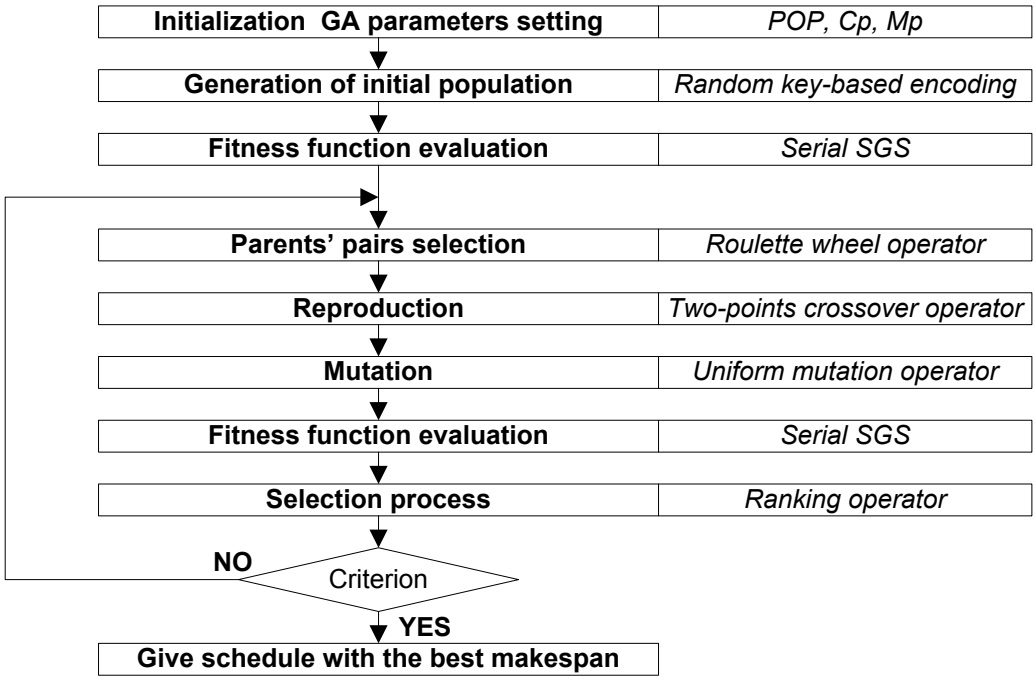


Figure 1: Framework of the proposed GA approach

Initialization and GA parameters setting

First of all, data are stored and variables initialized. There are three important parameters in the GA program: the size of the population (*POP*), which remains the same after each generation, the crossover probability (*Cp*), which determines whether a pair of chromosomes will be reproduced, and the mutation probability (*Mp*), which indicates whether a chromosome will mutate or not [2].

Generation of initial population - Random key-based encoding

An initial population of *POP* individuals is generated with a random key-based encoding method. According to this method, each randomly-generated individual is a vector representing a priority list which attributes a weight to each task [1]. Random key-based encoding methods are more suitable than activity list-based encoding methods for scheduling problems, because it always produces legal offsprings' chromosomes [6].

Fitness function evaluation - Serial SGS

The first generation of chromosomes is evaluated with the fitness function, in order to obtain the fitness values - the project time completion in our study. To this end, and for each individual, the fitness function creates the schedule in accordance with the Serial Schedule Generation Scheme (Serial SGS) and the random key-based method, as described by Hartmann [1]. All tasks whose predecessors have been scheduled are called eligible tasks. Among them, the task with the highest random key is selected and scheduled at the earlier possible start time. By iteration, a schedule is completed until the last task, the end time of which gives the duration of the entire project.

Parents' pairs selection - roulette wheel operator

Afterwards, pairs to be reproduced are selected by using a roulette wheel operator. This method, one of the most popular in the field of GA [3], assigns a selection probability to each chromosome, depending on their fitness value. Fit individuals, who have a higher selection probability, are more likely to be paired [3].

Reproduction – two-points crossover operator

Once parents have been paired, they will either be cloned or reproduced to create two different children according to the crossover probability, *Cp*. The reproduction is conducted through a crossover operator which exchanges and recombines parents sequence segments to produce two new individuals. The proposed GA uses a two-points crossover operator that consists in randomly selecting two cutting points and exchanging the genetic material that falls within the two points [1]-[3].

Uniform mutation operator

The uniform mutation, which is defined as the permutation of two randomly-chosen neighbouring genes, is conducted within an offspring with a mutation probability, *Mp*.

Selection process – ranking operator

Each child is then evaluated through the fitness function to determine its makespan. The population, composed of both parents and children, is reduced to its standard size by using a simple ranking method to keep the *POP* fittest individuals and remove the remaining ones [1]. The new set of individuals represents the next generation.

Iteration

When the stop criterion is reached - number of generations in our study, the GA stops and returns the schedule with the best makespan found so far among the evaluated individuals from initialization. Otherwise, the process is repeated to create a fitter chromosomes generation.

EXPERIMENTAL RESULTS

For the purpose of the study, we developed our GA program with the Ruby programming language, a dynamic object-oriented programming language, which is suitable to handle objects

like chromosomes - seen as arrays. The experiments were carried out on a Intel Core 2 Duo 2.4 GHz processor and 2 GB of RAM under the Mac Os X10.5.8 operating system.

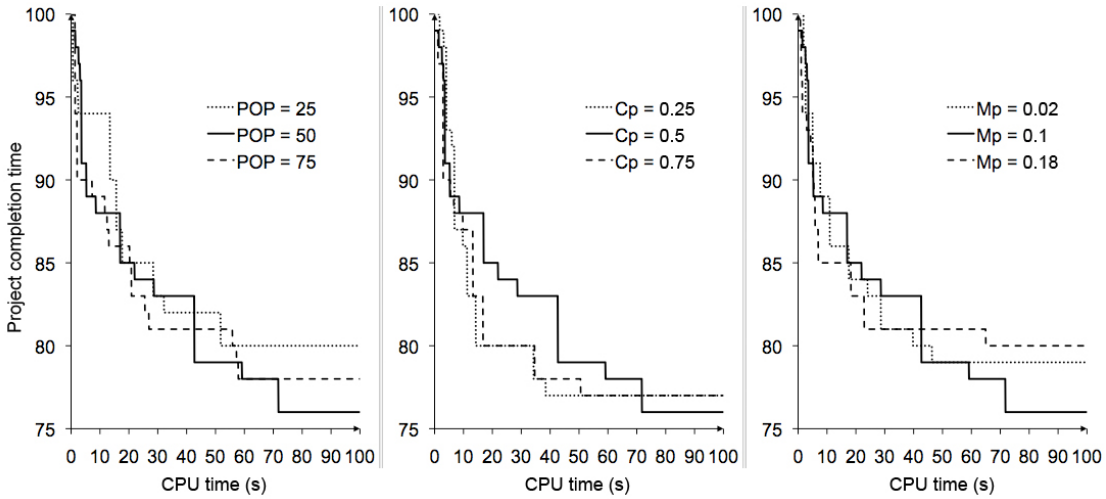
Illustrative case

We consider a project instance generated by Kolisch and Sprecher [4] composed of 30 tasks and 4 renewable resources. As we defined more constraints compared to the original RCPSP problem, two more parameters were defined (i.e. N_{im} and L_{im}). The project data are similar to those of [7]. Five materials have been considered of two different types (related to separate material storage areas). Materials 2 and 4 are stored in area 1 (γ = 1) and materials 1, 3 and 5 are stored in area 2 (γ = 2). Maximum storage capacity is defined as (Es_1 = 7, Es_2 = 7) and the vector e_m is defined as [1,1,1,1,1].

Configuration of GA parameters

As presented in section 2.3, the GA depends on the population size, the crossover probability and the mutation probability. A well-defined configuration of these parameters is a critical step to arrive at optimal solutions. Indeed, a large population size generally results in better solution, because the GA covers more space search and prevents convergence to local solutions, but it also increases computational times and memory requirements. A higher crossover accelerates the exchange of segments in the population, but with the risk of eliminating high performing segments faster than the selection can produce improvements. At the opposite, a lower crossover rate may lead to a lower exploration rate and causes stagnation, but it decreases the number of fitness evaluations required to solve the problem and allows some individuals with high performance to be copied directly to the next generation. The mutation rate is usually set to a low value. It introduces diversity, allows better exploration of the space search and avoids to become stuck in a local minimum. However it may destroy promising sequences. A high mutation rate is equivalent to a random search.

We propose a sensitivity analysis of the GA with respect to these three parameters in order to examine their effects on the performance of the algorithm. The default values of the population size, the crossover probability and the mutation probability are set to 50, 0.5 and 0.1, respectively. Each parameter is varied to higher and lower value, one at a time, with the other parameters set a their default value. The progress of the minimum project duration found so far among the tested solutions from initialization until the stop criterion is met is shown on Figure 2 for each combination of GA parameters.



(a) with Cp = 0.5, Mp = 0.1 (b) with POP = 50, Mp = 0.1 (c) with POP = 50, Cp = 0.5

Figure 2: Effects of GA parameters on the convergence of the project duration

All the GA experiments presented in Figure 2 converge to a value in less than 75 seconds. The best project duration, 76 time units, is provided with *POP* = 50, *Cp* = 0.5 and *Mp* = 0.1. This configuration of GA parameters is subsequently applied in the remainder of the paper.

Results

In the original project extracted from PSPLIB [4], the optimal project makespan was 38 time units (with no storage area capacity and no delay between material availability and activity starting time). Because of the storage limits constraints defined in our illustrative example, all material cannot be delivered at the beginning of the project which forces material delivery to be postponed. Consequently, some activities are also delayed which results in a new total calculated optimal makespan of 43 time units (still without considering the storage area constraints).

As a benchmark, our results obtained with the proposed GA for the RCPSP with material constraints are first compared with the solutions obtained in [7] using heuristics based on both serial and parallel algorithms with 100 priority rules including the most commonly used in the literature. The quality of the solutions obtained was highly variable, varying from 87 time units to 105 time units when storage area capacities are set to (7,7), compared to 76 time units for the proposed GA. Poor schedules are explained by long storage periods of material before their actual usage which limit the access of material storage for other activities.

Figure 3 shows total project durations estimated by our proposed GA for different values of storage area capacities. As the capacity of the storage areas become larger, the total completion time for the project decreases. For capacity values exceeding (Sa_1,Sa_2)>(19,19), the project completion time seems to converge towards the original problem's optimal solution. On the other side, the project completion time increases quickly as the storage area capacity is reduced. For example, the total calculated project duration is 76 time units, when storage area capacities are set to (7,7).

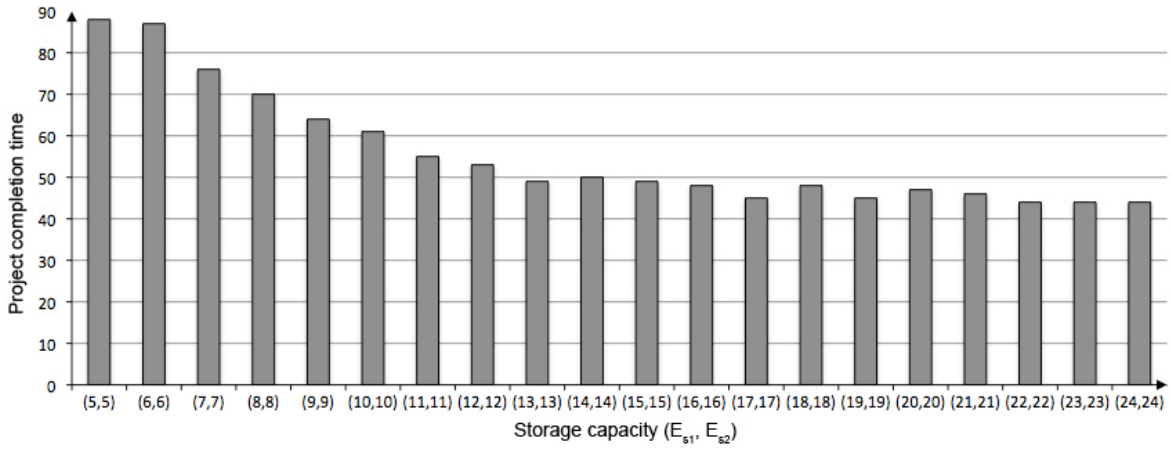


Figure 3: Project completion time vs storage area capacity

CONCLUSIONS

In spite of all research efforts accomplished these years to develop strong project management tools, project planning taking into account space and equipment availability are mostly developed on the basis of planners intuition and experience. Unfortunately, this task is nearly infeasible to perform in the case of large projects, due to the combinational nature of the resource allocation problem. In this research, we explored this issue and proposed a way of formulating this problem as a resource-constrained project scheduling problem. We considered constant storage area capacity and material delivery constraints. Subsequently, a genetic algorithm was developed in order to determine a feasible project plan. Our results demonstrated that the use of such a metaheuristic approach surpasses most of heuristic approaches known in literacy.

Further testing with multiple project data sets is however required in order to generalize these results. In future works, we will develop and improve the GA algorithm, by implementing new mechanisms and testing their impact on its performance. Finally, the comparison of our GA resolution approach with other metaheuristics also represents interesting research perspectives.

REFERENCES

[1] Hartmann, S. (1999). Project Scheduling under Limited Resources, Berlin: Springer, ISBN 3-540-66392-4, pp. 88-93.

[2] Kim, J.L., Ellis, R.D. (2008). “Permutation-based elitist genetic algorithm for optimization of large-sized resource-constrained project scheduling”, in Journal of Construction engineering and management, 134(11): 904-913.

[3] Kim, J.L., Ellis, R.D. (2010). “Comparing schedule generation schemes in resource-constrained project scheduling using elitist genetic algorithm”, in Journal of Construction engineering and management, 136(2):160-169.

[4] Kolisch, R., Sprecher, A. (1996). “PSPLIB - A project scheduling problem library”, in European Journal of Operational Research, 96(1):205-216.

[5] Koskela, L., Howell, G. (2002). “The underlying theory of project management simple percentage is obsolete”, in Proceedings of the PMI research conference, Seattle, WA, pp. 293-302.

[6] Okada, I., Lin, L. Gen, M. (2009). “Solving resource constrained multiple project scheduling problem by random key-based genetic algorithm”, in Electronics and Communications in Japan, 92(8):25-35.

[7] Pellerin, R., Sadr, J., Guevremont, M., Rousseau, L.M. (2009). “Planning of international projects with material constraints”, in International Conference on Industrial Engineering and Systems Management, Montreal, Canada.

[8] Polat, G., Arditi, D. (2005). “The JIT materials management system in developing countries”, in Construction Management and Economics, 23(7):697-712.

[9] Tserng, H.P., Yin, S.Y.L., Li, S. (2006). “Developing a resource supply chain planning system for construction projects”, in Journal of Construction Engineering and Management, 132(4):393-407.

ANNEXE 2 – Programme « Algorithme génétique » - Interface principale

```
# ------------------------------------------------------------------------------
# Class Name::    GeneralInterface
# Release::       3.0
# Created by::    Benoit Saenz de Ugarte
# Updated by::    Jean-Baptiste Boucherit
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2008-11
# Extend::        -
# ------------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ==========================
#    1.0       Benoit       2008-11      Creation
#    2.0       J.Baptiste   2009-11-06   Update for RCPSP
#    2.1       J.Baptiste   2010-03-14   Creation of output file
#    2.2       J.Baptiste   2010-03-25   Extraction from Excel
#    3.0       J.Baptiste   2010-07-13   Update for bigger projects
#
# = Description
#    General interface of the programme
#     * parameters
#     * launching
#     * analysis


autoload(:Project, "project_generator.rb")
autoload(:GeneticAlgorithm, "genetic_algorithm.rb")
autoload(:Chromosome, "chromosome.rb")
#autoload(:Project, "initial_project.rb")

# prompt project informations
puts "How many tasks? (30,60,90,120)"
@@nb_task = gets.chomp.to_i
puts "Which set of parameters? (1 to 5)"
@@project_nb = gets.chomp.to_i
puts "Choose area capacity [x,y]:"
puts "x="
area1 = gets.chomp.to_i
puts "y="
area2 = gets.chomp.to_i
@@areas_capacity = [area1,area2]

# GA parameters
population_size = 51
generation_gap = 0.999
mutation_prob = 0.18
@@crossover_prob = 0.5
priority_list = nil

# simulation parameters
iterations = 200
seed = 2990792146479947289
@@tempo = 0
simulations = 5
simulation_time = 100 #seconds

# generate instance problem
```

```
instance = Project.new
total = []
(simulation_time * 10).times do |j|
  total << []
end


# simulation
puts ""
puts "Simulations start time : #{Time.now.hour}:#{Time.now.min}:#{Time.now.sec}"
simulations.times do |j|
  puts ""
  puts "Simulation ##{j+1} - #{Time.now.hour}:#{Time.now.min}:#{Time.now.sec}"
  start_time = Time.now
  @@simtime = []
  @@value = []
  ga = GeneticAlgorithm.new(population_size, generation_gap, iterations, seed)
{Chromosome.new(@@nb_task, mutation_prob, instance, priority_list)}
  ga.run
  end_time = Time.now
  processing_time = end_time - start_time

  puts "Scheduled tasks : #{ga.best.schedule}"
  puts "Makespan : #{ga.best.makespan}"
  puts "Processing time : #{processing_time.round}s -
#{processing_time.to_i/60}min#{((processing_time/60-
processing_time.to_i/60)*60).to_i}s"

  # print the results in output file
  @z = @@value[0]
  (1..(simulation_time * 10)).each do |x|
    x = x.to_f / 10
    (0..@@simtime.size-2).each do |i|
      if x >= @@simtime[i] && x < @@simtime[i+1]
        @z = @@value[i]
      else
        if x >= @@simtime[@@simtime.size-1]
          @z = @@value[@@simtime.size-1]
        end
      end
    end
    total[10*x-1] << @z
  end
end

(simulation_time * 10).times do |i|
  output = File.open("c:/ruby/output_#{@@nb_task}.#{@@project_nb}.txt", "a")
  output.puts "#{total[i].join(" ")}"
  output.close
end

puts ""
puts "Simulations end time : #{Time.now.hour}:#{Time.now.min}:#{Time.now.sec}"
puts ""
puts "Printed in output_#{@@nb_task}.#{@@project_nb}.txt"
```

ANNEXE 3 – Programme « Algorithme génétique » - Algorithme génétique

```
# ------------------------------------------------------------------------------
# Class Name::    GeneticAlgorithm
# Release::       2.0
# Created by::    Benoit Saenz de Ugarte
# Updated by::    Jean-Baptiste Boucherit
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2008-11-14
# Extend::        -
# ------------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ================
#    1.0       Benoit       2008-11-14   Creation
#    2.0       J.Baptiste   2009-11-05   Update for RCPSP
#
# = Description
#    Main class of the Basic GA
#     * fitness evaluation
#     * pairing (random, roulette, ranking,...)
#     * survivors selection (random, tournament,...)
#     * initialization
#     * iteration
#     * best chromosome saving


class GeneticAlgorithm

  # : [Array] Genetic algorithm population
  attr_reader :population
  # : [Chromosome] Best chromosome fund during the evolution process
  attr_reader :best
  # : [Integer] Best chromosome iteration id
  attr_reader :best_iteration
  # : [Integer] Number of iteration to process
  attr_reader :nb_iteration
  # : [Array] Memory of the Genetic Algorithm used for statistical purpose
  attr_accessor :memory
  # : [Integer] The current iteration ID
  attr_accessor :iterationID
  # : [Time] GA start time
  attr_accessor :start_time
  # : [Time] Time set
  #  attr_accessor :time_set


  # : Genetic algorithm type
  GATYPE = 'WITHOUT_MEMORY'


  ## INITIALIZE
  #  Initialize the Genetic Algorithm and its population
  #    population_size  #=> [integer] The population size of the genetic algorithm
  #    generation_gap   #=> [float] The ratio of chromosome created at each iteration
  #    nb_iteration     #=> [integer] The number of iteration to process
  def initialize(population_size, generation_gap, nb_iteration, seed)
    @start_time = Time.now
    #    puts @start_time.inspect
    @memory = Hash.new
```

```
    # check/valid the input arguments
    population_size = population_size.to_i
    generation_gap = generation_gap.to_f
    nb_iteration = nb_iteration.to_i
    seed = seed.to_i
    raise ArgumentError, 'the population size must be > 0' if population_size <= 0
    raise ArgumentError, 'generation_gap must be <= 1 and >= 0' if generation_gap > 1
and generation_gap < 0
    raise ArgumentError, 'the number of iteration must be > 0' if population_size <= 0
    raise ArgumentError, 'the seed must be > 0' if population_size <= 0

    # initialize the rand seed (for repeatability)
#    srand(seed)

    # Generate the initial population
    @population = (0..population_size-1).map{|i| yield i}
    @selection_size = [(population_size * generation_gap).to_i,2].max
    # Initialize id and iteration number
    @nb_iteration = nb_iteration-1
    @iterationID = 1
    @best_iteration = -1

    # Evaluate the initial population makespan
    @population.each do |chromo|
      insert_into_memory(chromo)
      chromo.eval_makespan
    end

    # eval fitness
    eval_fitness

    # Keep the best
    remember_best

    # Initialization = step 1
    @iterationID = @iterationID + 1
    @@tempo = 0
  end

  ##
  # Find the n chromosome with the highest fitness value (order by decreasing fitness)
  #   n #=> [integer] Number of chromosome to select
  def fittest(n = @selection_size)
    @population.sort_by{|member| -member.fitness}[0, n]
  end

  ##
  # Find the n chromosome with the highest fitness value (order by increasing fitness)
  #   n #=> [integer] Number of chromosome to select
  def less_fittest(n = @selection_size)
    @population.sort_by{|member| -member.fitness}[-n, n].reverse
  end

  ##
  # Process one iteration
  # * pairs creation
  # * children creation
  # * children evaluation
  # * population update
  # * population fitness evaluation
  def step
    # select survivors
    survivors = select_survivors
```

```
    # create pairs of survivors
    paired_survivors = pair_parents(survivors, :roulette)

    # create children
    children = []
    children_generated = 0
    paired_survivors[:mothers].each_index do |index|
#      crossover_trial = rand
      # create the 2 children from those parents
#      if crossover_trial <= @@crossover_prob
#        temp =
paired_survivors[:mothers][index].reproduce_with(paired_survivors[:fathers][index],
:twopointscrossover)
#      else
#        temp = [paired_survivors[:mothers][index], paired_survivors[:fathers][index]]
#      end
      temp =
paired_survivors[:mothers][index].reproduce_with(paired_survivors[:fathers][index],
:twopointscrossover)
      temp.each do |child|
        if children_generated < @selection_size
          # increase the number of childre generated
          children_generated+=1

          # insert the child in the children collection
          children << child
        end
      end
    end

    # eval children makespan
    children.each do |chromo|
      insert_into_memory(chromo)
      chromo.eval_makespan
    end

    # delete the less fittest chromosome
    less_fittest_chromos = less_fittest(@selection_size)
    objective_size = @population.size - @selection_size
    #If the new size is right by substracting the 2 arrays : do it
    if (@population - less_fittest_chromos).size == objective_size
      @population = @population - less_fittest_chromos
    else
      # for each chromosome to remove : find it and remove it
      less_fittest_chromos.each do |chromo|
        @population.delete_at(@population.index(chromo))
      end
    end

    # insert children in the population
    @population = @population + children


    # eval fitness
    eval_fitness()

    # Keep and return the best
    remember_best
    return @best
  end

  ##
  # Run the evoluation process during a given number of iteration
  #   steps #=> [integer] The number of iteration to process
  def run(steps = @nb_iteration)
```

```
    steps.times do |i|
      step
      @iterationID = @iterationID + 1
    end
    return @best
  end

  ##
  # Update the best chromosome of the population
  def remember_best
    current = fittest(1).first
    if @best.nil? || current.fitness > @best.fitness
      @best = current
      @best_iteration = @iterationID
      #      puts "iteration #{@best_iteration.inspect} / makespan
#{@best.makespan.inspect} / time = #{(Time.now - @start_time).inspect} s / scheduling =
#{@best.schedule}"
      #      puts @best.makespan.inspect + "; " + (Time.now - @start_time).inspect
      @@simtime << ((((Time.now - @start_time)*10).round).to_f)/10
      @@value << @best.makespan
    end
  end

  ##
  # fitness uniformement réparti entre 0 et 1 sur l'ensemble des chromosomes
  #   subpopulation #=> [array] array of chromosomes to evaluate
  def eval_fitness(subpopulation = @population)
    # Size of the population
    subsize = subpopulation.size
    # Fitness value (if all chromosome have a unique cost function value)
    frun = (0..subsize-1).map { |i| i.to_f / (subsize - 1).to_f }
    # Sort the population and cost function value array
    subpopulation_sorted = subpopulation.sort_by{|member| -member.makespan}
    obj_values = (0..subsize-1).map { |i| subpopulation_sorted[i].makespan }
    # Eval fitness based on frun
    # When 2 (or more) chromosome have the same cost function value, they must have the
same fitness
    i=0
    while i < subsize
      # Find all the Chromosome with the same cost
      j = obj_values.rindex(obj_values[i])
      # Mean of frun
      fitness =  (eval frun[i..j].join('+')).to_f / (j-i+1).to_f
      #Update the fitness and the rank
      subpopulation_sorted[i..j].each do |chromo|
        chromo.fitness = fitness
        chromo.rank = i + 1
      end
      i = j+1
    end
  end

  ##
  # select survivors (possible parent)
  # methods :
  #   1. a finite and fix part of the initial population
  #   2. dynamic thresholding (NOT IMPLEMENTED)
  def select_survivors
    survivors_size = @selection_size % 2 > 0 ? @selection_size + 1 : @selection_size
    fittest(survivors_size)
  end

  ##
  # Create pairs of parents for the reproduction process
  #   parents        #=> [array] array of chromosomes that can become a parent
```

```
  #   method         #=> [:toptobottom, :random, :ranking, :roulette, :tournament] the
method used to create pairs
  #   distinctparent #=> [nil, :distinctparents] +:distinctparents+ if the 2 parents
need to be unique
  def pair_parents(parents, method = :roulette, distinctparent = nil)
    # Number of parents pair to create (each pair create 2 offspring)
    nb_pairs = (@selection_size + 1) / 2

    # flag for distinc parents (symbol or nil)
    # distinctparent = :distinctparents

    # Avoid infinite loops when trying to have 2 different parents
    limit = 10

    # The parents size always agrees with the number of pairs to create because
    # the survivor selection take care of it

    # Parent pairing
    fathers = []
    mothers = []
    case method
    when :toptobottom
      # pair parent from top to bottom
      (1..nb_pairs).map do |i|
        mothers << parents[2*i-2]
        fathers << parents[2*i-1]
      end
    when :random
      # random integer are used for pairing
      parents_size = parents.size

      # generate random pairing
      (1..nb_pairs).map do
        # The first selected parent is the mother
        index = (rand()*parents_size).floor
        mother = parents[index]

        # The select the father
        index = (rand()*parents_size).floor
        father = parents[index]

        if distinctparent == :distinctparents
          # We won't select the same chromosome again
          i = 0
          while mother.signature == father.signature and i < limit
            index = (rand()*parents_size).floor
            father = parents[index]
            i += 1
          end
          raise RuntimeError, 'unable to select two different parents (signature of the
selected parent: '+ mother.signature.to_s + ')' if mother.signature == father.signature
        end

        mothers << mother
        fathers << father
      end
    when :ranking
      # ranking method
      # calculate selection probabilities
      probabilities = []
      ranks = (1..2*nb_pairs).map{|i| i}
      rank_sum = (eval ranks.join('+'))
      ranks.each { |rank| probabilities << (2*nb_pairs - rank + 1).to_f / rank_sum.to_f
}
```

```
      # calculate cumulative probabilities
      sum = 0.0
      cumulative_probabilities = (0..2*nb_pairs-1).map {|i| sum = sum +
probabilities[i]}

      # Random number _generation( @selection_size numbers)
      trials = (1..2*nb_pairs).map { |i| sum * rand }

      # select fathers and mothers
      (1..nb_pairs).map do |i|
        trial = trials[2*i-2]
        mother = parents[cumulative_probabilities.index(cumulative_probabilities.find {
|p| p >= trial })]
        trial = trials[2*i-1]
        father = parents[cumulative_probabilities.index(cumulative_probabilities.find {
|p| p >= trial })]

        if distinctparent == :distinctparents
          # We won't select the same chromosome again
          i = 0
          while mother.signature == father.signature and i < limit
            trial = sum * rand
            father =
parents[cumulative_probabilities.index(cumulative_probabilities.find { |p| p >= trial
})]
            i += 1
          end
          raise RuntimeError, 'unable to select two different parents (signature of the
selected parent: '+ mother.signature.to_s + ')' if mother.signature == father.signature
        end

        mothers << mother
        fathers << father
      end
    when :roulette
      # roulette method
      # Cumulative sum of fitness
      sum = 0.0
      cumulative_fitness = (0..@selection_size-1).map {|i| sum = sum +
parents[i].fitness}

      # Random number generation( @selection_size numbers)
      trials = (1..2*nb_pairs).map { |i| sum * rand }

      # select fathers and mothers
      (1..nb_pairs).map do |i|
        trial = trials[2*i-2]
        mother = parents[cumulative_fitness.index(cumulative_fitness.find { |p| p >=
trial })]
        trial = trials[2*i-1]
        father = parents[cumulative_fitness.index(cumulative_fitness.find { |p| p >=
trial })]

        if distinctparent == :distinctparents
          # We won't select the same chromosome again
          i = 0
          while mother.signature == father.signature and i < limit
            trial = sum * rand
            father = parents[cumulative_fitness.index(cumulative_fitness.find { |p| p
>= trial })]
            i += 1
          end
          raise RuntimeError, 'unable to select two different parents (signature of the
selected parent: '+ mother.signature.to_s + ')' if mother.signature == father.signature
        end
```

```
        mothers << mother
        fathers << father
      end
    when :tournament
      # tournament method
      parent_size = parents.size

      # for each parent to select, organize a tournament (random size)
      (1..nb_pairs).map do |i|
        # Mother tournament
        # eval the tournament size
        tournament_size = [(nb_pairs * rand).to_i + 1,2].max
        # all players
        indexes = (1..parent_size).map {|index| index-1}
        # select tournament players
        selected_indexes = (1..tournament_size).map
{indexes.delete_at(rand(indexes.size))}
        # find the player with the best fitness
        mother = nil
        selected_indexes.each {|index| mother = parents[index] if mother.nil? ||
parents[index].fitness > mother.fitness}

        # Father tournament
        # eval the tournament size
        tournament_size = [(nb_pairs * rand).to_i + 1,2].max
        # all players
        indexes = (1..parent_size).map {|index| index-1}
        # select tournament players
        selected_indexes = (1..tournament_size).map
{indexes.delete_at(rand(indexes.size))}
        # find the player with the best fitness
        father = nil
        if distinctparent == :distinctparents
          # We won't select the same chromosome again
          selected_indexes.each {|index| father = parents[index] if father.nil? ||
(parents[index].fitness > father.fitness and parents[index].signature !=
mother.signature)}
        else
          selected_indexes.each {|index| father = parents[index] if father.nil? ||
parents[index].fitness > father.fitness}
        end

        #But sometimes the tournament players are all the sames...
        if distinctparent == :distinctparents
          # We won't select the same chromosome again
          i = 0
          while mother.signature == father.signature and i < limit
            # Father tournament
            # eval the tournament size
            tournament_size = [(nb_pairs * rand).to_i + 1,2].max
            # all players
            indexes = (1..parent_size).map {|index| index-1}
            # select tournament players
            selected_indexes = (1..tournament_size).map
{indexes.delete_at(rand(indexes.size))}
            # find the player with the best fitness
            father = nil
            if distinctparent == :distinctparents
              # We won't select the same chromosome again
              selected_indexes.each {|index| father = parents[index] if father.nil? ||
(parents[index].fitness > father.fitness and parents[index].signature !=
mother.signature)}
            else
              selected_indexes.each {|index| father = parents[index] if father.nil? ||
```

```
parents[index].fitness > father.fitness}
            end
          end
          raise RuntimeError, 'unable to select two different parents (signature of the
selected parent: '+ mother.signature.to_s + ')' if mother.signature == father.signature
        end

        mothers << mother
        fathers << father
      end
    else
      raise ArgumentError, 'Unknown pairing method'
    end

    # Return values
    {:fathers => fathers, :mothers => mothers}
  end

  # Insert chromosome in memory
  def insert_into_memory(chromosome)
    signature = chromosome.signature
    @memory[signature]=[] if @memory[signature].nil?
    @memory[signature] << chromosome
  end

  # Memory analysis
  def analyse_memory
    memory_size = 0
    unique_members = 0
    @memory.each do |signature, signature_table|
      memory_size = memory_size + signature_table.size
      unique_members = unique_members + signature_table.uniq.size
    end
    return [memory_size, unique_members]
  end
end
```

ANNEXE 4 – Programme « Algorithme génétique » - Chromosome

```
# ------------------------------------------------------------------------------
# Class Name::    Chromosome
# Release::       2.0
# Created by::    Benoit Saenz de Ugarte
# Updated by::    Jean-Baptiste Boucherit
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2008-10-31
# Extend::        Array
# ------------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ============================
#    1.0       Benoit       2008-10-31   Creation
#    1.1       J.Baptiste   2009-09-25   Add uniform_crossover method
#    1.2       J.Baptiste   2009-10-14   Add fitness function
#    1.3       J.Baptiste   2009-10-15   Add schedule method
#    2.0       J.Baptiste   2009-11-03   Update for RCPSP
#
# = Description
#    Chromosome of the GA
#     * mutation
#     * crossover
#     * fitness function
#     * makespan
#     * schedule


class Chromosome < Array

  # : [float] fitness value
  attr_accessor :fitness
  # : [integer] rank in the population
  attr_accessor :rank
  # : [float] makespan value
  attr_accessor :makespan
  # : [float] object signature
  attr_reader :signature
  # : [float] probability of mutation
  attr_accessor :mutation_prob
  # : [time] end time of iteration
  attr_accessor :end_time_iteration

  tempo = 0


  ## INITIALIZE
  #  Initialize the chromosome with a random or a given list of priorities
  #    length               #=> [integer] The length of the chromosome
  #    mutation_probability #=> [float] The probability of mutation
  #
  #    priorities_list      #=> [integer] A given list of priorities used to initialize
the chromosome
  def initialize(length, mutation_probability, project_instance = nil, priorities_list
= nil)
    # Avoid errors
    raise ArgumentError, "mutation probabilty must be <= 1" if mutation_probability > 1

    # Initialize mutation_prob, project_instance, makespan and fitness
```

```
    @mutation_prob = mutation_probability.to_f
    @@project_instance = project_instance unless (project_instance.nil?)
    @makespan = nil
    @fitness = nil

    # Generate a list of priorities (task 1 to length)
    priorities_list = nil if @@tempo == 1
    if priorities_list.nil?
      tasks = (1..length).map {|index| index}
      super(length) do |index|
        tasks.delete_at(rand(tasks.size))
      end
    else
      super(priorities_list)
      @@tempo = 1
    end

    # Give the chromosome a signature
    @signature = self.hash
    #    puts "initialized prio = " + self.inspect
  end


  ## REPRODUCTION
  #  Reproduction process
  #    father #=> [chromosome] Another chromosome used for the children creation
  #    method #=> [:onepointcrossover, :twopointscrossover] The method used to generate
the children
  def reproduce_with(father, method = :twopointcrossover)
    crossover_trial = rand
    if @signature != father.signature

      if crossover_trial <= @@crossover_prob



        chromosome_size = father.size

        # Crossover methods
        case method

          # 1-point crossover
        when :onepointcrossover
          # The crossover point must not be at the end of the chromosome
          first_point = rand(chromosome_size-2)

          # Initialize the sequences
          child1_sequence = []
          child2_sequence = []

          # The first part of children is the same as parents
          child1_sequence = self[0..first_point]
          child2_sequence = father[0..first_point]

          # The second part is the crossover
          child1_sequence += (father - child1_sequence)
          child2_sequence += (self - child2_sequence)

          # 2-points crossover
        when :twopointscrossover
          # The two crossover points must be different and not be at the end of the
chromosome
          first_point = 0
          temp = 0
          while first_point == temp
```

```
            first_point = rand(chromosome_size-2)
            temp = rand(chromosome_size-2)
          end
          second_point = temp > first_point ? temp : first_point
          first_point = temp if temp < first_point

          # Initialize the sequences
          child1_sequence = []
          child2_sequence = []

          # The first part of children is the same as parents
          child1_sequence = self[0..first_point]
          child2_sequence = father[0..first_point]

          # The second part is the crossover
          child1_sequence += ((father - child1_sequence))[0..second_point - first_point
- 1]
          child2_sequence += ((self - child2_sequence))[0..second_point - first_point -
1]

          # The third part is the remaining tasks, without crossover
          child1_sequence += (self - child1_sequence)
          child2_sequence += (father - child2_sequence)

          # Uniform crossover
        when :uniformcrossover
          # Generate the binary sequence
          binary_sequence = (0..chromosome_size-1).map {rand(2)}

          # Initialize the sequences
          child1_sequence = []
          child2_sequence = []

          # The first child
          child1_sequence = binary_sequence[0] == 0 ? [self.first].push :
[father.first].push
          (1..chromosome_size-1).each do |index|
            if binary_sequence[index] == 0
              child1_sequence << (self-child1_sequence).first
            else
              child1_sequence << (father-child1_sequence).first
            end
          end

          # The second child
          child2_sequence = binary_sequence[0] == 1 ? [self.first].push :
[father.first].push
          (1..chromosome_size-1).each do |index|
            if binary_sequence[index] == 1
              child2_sequence << (self-child2_sequence).first
            else
              child2_sequence << (father-child2_sequence).first
            end
          end

          # Otherwise, it's a wrong method
        else
          raise ArgumentError, 'Unknown crossover method'
        end

      else
        child1_sequence = self
        child2_sequence = father

      end
```

```
      # If children are the same, then a mutation is necessary
      if child1_sequence.hash == child2_sequence.hash
        mutation_trial1 = 0
        mutation_trial2 = 0
      else
        mutation_trial1 = rand
        mutation_trial2 = rand
      end

      # If parents are the same, children too, then mutation is necessary
    else
      child1_sequence = Array.new(father)
      child2_sequence = Array.new(father)
      mutation_trial1 = 0
      mutation_trial2 = 0
    end

    # Create the 2 children based on the sequence
    child1 = Chromosome.new(chromosome_size, self.mutation_prob, nil, child1_sequence)
    @@tempo = 0
    child2 = Chromosome.new(chromosome_size, self.mutation_prob, nil, child2_sequence)
    @@tempo = 0

    # Process mutation
    child1.mutation if mutation_trial1 <= child1.mutation_prob
    child2.mutation if mutation_trial2 <= child2.mutation_prob

    # while loop to avoid to have the same 2 children (possible if the 2 parents are
identical)
    while child1.signature == child2.signature
      child2.mutation
    end

    # return the 2 children
    #    puts "c1 = " + child1.inspect
    #    puts "c2 = " + child2.inspect
    [child1, child2]
  end


  ## MUTATION
  #  Mutation process
  def mutation
    chromosome_size = self.size

    # Choose different random indexes
    index1 = 0
    temp = 0
    while index1 == temp
      index1 = rand(chromosome_size)
      temp = rand(chromosome_size)
    end

    # Reorder the 2 indexes
    index2 = temp > index1 ? temp : index1
    index1 = temp if temp < index1

    #permutation of the 2 elements
    temp = self[index2]
    self[index2] = self[index1]
    self[index1] = temp

    # update the signature
    @signature = self.hash
```

```
  end


  ## MAKESPAN
  #  Evaluate makespan
  def eval_makespan
    @makespan = 0
    t=1
    endtime = Array.new(@@project_instance.nb_tasks, 0)

    # Schedule the tasks
    @scheduled_tasks = [1]
    #    puts "st = " + @scheduled_tasks.inspect
    eligible_tasks = []

    (2..@@project_instance.nb_tasks-1).each do |i|
      #      puts i
      pred_scheduled_tasks = [1]
      (2..@@project_instance.nb_tasks-1).each do |j|
        essai = []
        @@project_instance.task_pred[j-1].each { |k| @scheduled_tasks.include?(k) ?
essai << "yes" : essai << "no" }
        pred_scheduled_tasks << j if essai.include?("no") == false
      end
      #      puts "pst = " + pred_scheduled_tasks.inspect
      #      puts "self = " + self.inspect
      eligible_tasks = (pred_scheduled_tasks - @scheduled_tasks).sort_by do |m|
        -self[m-2]
      end
      #      puts "prior =
[30,29,28,27,26,25,24,23,22,21,19,17,20,16,14,13,6,5,18,12,11,3,15,7,4,10,1,9,8,2]"
      #      puts "et non classe = " + (pred_scheduled_tasks -
@scheduled_tasks).inspect
      #      puts "et = " + eligible_tasks.inspect
      @scheduled_tasks << eligible_tasks[0]
      #      puts "st = " + @scheduled_tasks.inspect
      #      puts ""

    end
    @scheduled_tasks = @scheduled_tasks.push(@@project_instance.nb_tasks)
    #    puts "scheduled tasks = " + @scheduled_tasks.inspect
    #    @scheduled_tasks =
[1,2,3,4,5,6,7,8,9,10,11,13,15,12,16,18,19,26,27,14,20,21,29,17,25,28,22,31,23,24,30,32
]

    # Initialize ressources and storage usages
    rsrcs_usage = []
    (0..@@project_instance.nb_rsrcs-1).each do |i|
      rsrcs_usage << Array.new(@@project_instance.rsrcs_capacity[i],0)
    end
    storage_usage = []
    (0..@@project_instance.nb_areas-1).each do |i|
#      storage_usage << Array.new(@@project_instance.areas_capacity[i],0)
    storage_usage << Array.new(@@areas_capacity[i],0)
#puts @@areas_capacity.inspect
    end

    # For each task
    @scheduled_tasks.each do |task|

      ## Data for the task
      #   Last precedence
      preds_end_list = [0]
      @@project_instance.task_pred[task-1].each do |i|
        preds_end_list += [endtime[i-1]]
```

```
      end
      preds_end = preds_end_list.max

      #   Ressources availability
      rsrcs_avail_time_list = [0]
      (0..@@project_instance.nb_rsrcs-1).each do |i|
        rsrcs_avail_time_list += rsrcs_usage[i][0, @@project_instance.task_rsrcs[task-
1][i]]
      end
      rsrcs_avail_time = rsrcs_avail_time_list.max

      #   Materials availability
      mtrls_avail_time_list = [0]
      (0..@@project_instance.nb_areas-1).each do |i|
        needed_surface = 0
        @@project_instance.area_mtrls[i].each do |j|
          needed_surface += @@project_instance.task_mtrls[task-1][j-1]
        end
        mtrls_avail_time_list += storage_usage[i][0, needed_surface]
      end
      mtrls_avail_time = mtrls_avail_time_list.max + (@@project_instance.with_delay *
@@project_instance.delay[task-1])


      ##
      thestart = [preds_end, rsrcs_avail_time, mtrls_avail_time].max
      theend = thestart + @@project_instance.task_dur[task-1]
      endtime[task-1] = theend


      ## Update ressources and storage usages
      #   Ressources usage
      (0..@@project_instance.nb_rsrcs-1).each do |i|
        (0..@@project_instance.task_rsrcs[task-1][i]-1).each do |j|
          rsrcs_usage[i][j] = theend
        end
        rsrcs_usage[i] = rsrcs_usage[i].sort
      end

      #   Storage usage
      (0..@@project_instance.nb_areas-1).each do |i|
        needed_surface = 0
        @@project_instance.area_mtrls[i].each do |j|
          needed_surface += @@project_instance.task_mtrls[task-1][j-1]
        end
        (0..needed_surface-1).each { |k| storage_usage[i][k]=theend }
        storage_usage[i]= storage_usage[i].sort
      end

      ## Give the makespan
      @makespan = theend if theend > @makespan
      t += 1

    end
    #@end_time_iteration = Time.now
    #    puts "makespan = " + @makespan.inspect
    #    puts ""
  end

  def schedule
    # Schedule the tasks
    @scheduled_tasks = [1]
    eligible_tasks = []
    (2..@@project_instance.nb_tasks-1).each do |i|
      pred_scheduled_tasks = [1]
```

```
      (2..@@project_instance.nb_tasks-1).each do |j|
        essai = []
        @@project_instance.task_pred[j-1].each { |k| @scheduled_tasks.include?(k) ?
essai << "yes" : essai << "no" }
        pred_scheduled_tasks << j if essai.include?("no") == false
      end
      eligible_tasks = (pred_scheduled_tasks - @scheduled_tasks).sort_by {|m| -self[m-
2]}
      @scheduled_tasks << eligible_tasks[0]

    end
    @scheduled_tasks = @scheduled_tasks.push(@@project_instance.nb_tasks)

    return @scheduled_tasks.join(",")


  end
end
```

ANNEXE 5 – Programme « Algorithme génétique » - Projet initial

```
# ------------------------------------------------------------------------
# Class Name::    ProjectData
# Release::       1.0
# Created by::    Jean-Baptiste Boucherit
# Updated by::    -
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2009-09-06
# Extend::        -
# ------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ===========
#    1.0       J.Baptiste   2009-09-06   Creation
#    1.1       J.Baptiste   2009-11-15   Update
#
# = Description
#    Data of the initial project
#     * tasks
#     * precedence
#     * constraints


class Project

  attr_accessor :nb_rsrcs
  attr_accessor :rsrcs_capacity
  attr_accessor :nb_mtrls
  attr_accessor :mtrls_surface
  attr_accessor :nb_areas
  attr_accessor :areas_capacity
  attr_accessor :area_mtrls
  attr_accessor :with_delay
  attr_accessor :delay
  attr_accessor :nb_tasks
  attr_accessor :task_dur
  attr_accessor :task_rsrcs
  attr_accessor :task_mtrls
  attr_accessor :task_pred


  ## PROJECT DATAS
  #  The different data used by the algorithm
  def initialize

    # Number of ressources and their capacities
    @nb_rsrcs = 4
    @rsrcs_capacity = [12,13,4,12]

    # Number of materials and their surfaces (in square feet)
    @nb_mtrls = 5
    @mtrls_surface = [1,1,1,1,1]

    # Number of storage area and their capacities (in square feet)
    @nb_areas = 2
    @areas_capacity = @@areas_capacity
    # Which material in which area
    @area_mtrls = [[2,4],[1,3,5]]
```

```
    # Delay for storage
    @with_delay = 1
    @delay = [0,3,1,0,2,3,1,2,1,0,1,2,1,1,1,0,0,3,0,0,2,0,1,2,1,1,1,2,1,0,1,0]

    # Number of tasks and their properties
    @nb_tasks = 30 + 2
    @task_dur = [0,8,4,6,3,8,5,9,2,7,9,2,6,3,9,10,6,5,3,7,2,7,2,3,3,7,8,3,7,2,2,0]
    @task_rsrcs =
[[0,0,0,0],[2,0,0,0],[10,0,0,0],[0,0,0,3],[3,0,0,0],[0,0,0,8],[4,0,0,0],[0,1,0,0],[6,0,
0,0],[0,0,0,1],

[0,5,0,0],[0,7,0,0],[3,0,0,0],[0,8,0,0],[3,0,0,0],[0,0,0,5],[0,0,0,8],[0,4,0,7],[0,1,0,
0],[0,10,0,0],[0,0,0,6],

[2,0,0,0],[3,0,0,0],[0,9,0,0],[4,0,0,0],[0,0,4,0],[0,0,0,7],[0,8,0,0],[0,7,0,0],[0,7,0,
0],[0,0,2,0],[0,0,0,0]]
    @task_succ =
[[2,3,4],[6,11,15],[7,8,13],[5,9,10],[20],[30],[27],[12,19,27],[14],[16,25],[20,26],[14
],[17,18],[17],

[25],[21,22],[22],[20,22],[24,29],[23,25],[28],[23],[24],[30],[30],[31],[28],[31],[32],
[32],[32],[]]
    @task_mtrls =
[[0,0,0,0,0],[2,0,1,0,0],[1,0,0,0,0],[3,1,0,5,0],[0,1,0,0,0],[4,0,0,1,0],[1,0,2,0,0],[2
,1,0,4,0],

[0,0,3,0,0],[0,0,0,1,0],[0,2,0,0,0],[0,1,1,0,0],[4,0,3,3,0],[0,1,0,0,0],[0,0,0,1,0],[2,
3,1,0,0],[0,1,0,0,0],

[3,0,1,1,0],[1,2,4,0,0],[0,1,0,2,0],[0,1,0,1,0],[4,0,0,0,0],[0,1,0,0,0],[0,0,2,3,0],[2,
3,0,0,0],[1,2,3,2,0],
      [0,3,0,1,0],[2,1,0,4,0],[0,2,0,1,2],[3,3,1,0,2],[0,4,3,0,2],[0,0,0,0,0]]

    # Convert task_succ into task_pred
    @task_pred = []
    (0..@nb_tasks-1).each do |i|
      (0..@task_succ[i].size-1).map do |j|
        @task_pred[@task_succ[i][j]-1].nil? ? @task_pred[@task_succ[i][j]-1] =
[i+1].push : @task_pred[@task_succ[i][j]-1] << i+1
      end
    end
    @task_pred[0] = []
  end
end
```

ANNEXE 6 – Programme « Algorithme génétique » - Générateur de projets

```
# ------------------------------------------------------------------------
# Class Name::    ProjectGenerator
# Release::       3.0
# Created by::    Jean-Baptiste Boucherit
# Updated by::    Jean-Baptiste Boucherit
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2009-11-06
# Extend::        -
# ------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ==========================
#    1.0       J.Baptiste   2009-11-06   Creation
#    2.0       J.Baptiste   2010-03-20   Update for random projects
#    2.1       J.Baptiste   2010-04-03   Extraction with Excel
#    3.0       J.Baptiste   2010-07-10   Udpate for bigger projects
#
# = Description
#    Data of a PSPLIB project
#     * tasks
#     * precedence
#     * constraints
#    Extraction from Excel


class Project

  ##
  # Load the win32ole library
  require 'win32ole'

  # Parameters
  attr_accessor :nb_rsrcs
  attr_accessor :rsrcs_capacity
  attr_accessor :nb_mtrls
  attr_accessor :mtrls_surface
  attr_accessor :nb_areas
  attr_accessor :areas_capacity
  attr_accessor :area_mtrls
  attr_accessor :with_delay
  attr_accessor :delay
  attr_accessor :nb_tasks
  attr_accessor :task_dur
  attr_accessor :task_rsrcs
  attr_accessor :task_mtrls
  attr_accessor :task_pred


  ## PROJECT DATAS
  #  The different data used by the algorithm
  def initialize

    ##
    # Open the project input file
    excel = WIN32OLE.new("excel.Application")
    workbook = excel.Workbooks.Open("c:/ruby/Instances_#{@@nb_task}_avec
approvisionnement.xlsx")
    worksheet = workbook.Worksheets(@@project_nb)
```

```
    @nb_tasks = worksheet.Range("e6").value.to_i

    # Number of ressources and their capacities
    @nb_rsrcs = worksheet.Range("e9").value.to_i
    @rsrcs_capacity =
worksheet.Range("b#{19+@nb_tasks+4+@nb_tasks+3}:e#{19+@nb_tasks+4+@nb_tasks+3}").value[
0].collect{|i| i.to_i}

    # Number of materials and their surfaces (in square feet)
    @nb_mtrls = 5
    @mtrls_surface = [1,1,1,1,1]

    # Number of storage area and their capacities (in square feet)
    @nb_areas = 2
    @areas_capacity = @@areas_capacity
    # Which material in which area
    @area_mtrls = [[2,4],[1,3,5]]
    # Delay for storage
    @with_delay = 0
    @delay =
worksheet.Range("b#{19+@nb_tasks+4+@nb_tasks+8}:b#{19+@nb_tasks+4+@nb_tasks+8+@nb_tasks
-1}").value.collect {|i| i[0].to_i}

    # Number of tasks and their properties
    @task_dur = worksheet.Range("c#{19+@nb_tasks+4}:c#{19+@nb_tasks+4+@nb_tasks-
1}").value.collect {|i| i[0].to_i}
    @task_rsrcs = worksheet.Range("d#{19+@nb_tasks+4}:g#{19+@nb_tasks+4+@nb_tasks-
1}").value.collect {|i| i.collect {|j| j.to_i}}
    max_succ = worksheet.Range("c19:c#{19+@nb_tasks-1}").value.max[0].to_i
    @task_succ = worksheet.Range("d19:#{(100+max_succ-1).chr}#{19+@nb_tasks-
1}").value.collect {|i| i.collect {|j| j.to_i}.delete_if{|k| k==0}}
    @task_mtrls =
worksheet.Range("c#{19+@nb_tasks+4+@nb_tasks+8}:g#{19+@nb_tasks+4+@nb_tasks+8+@nb_tasks
-1}").value.collect {|i| i.collect {|j| j.to_i}}

    # Convert task_succ into task_pred
    @task_pred = []
    (0..@nb_tasks-1).each do |i|
      (0..@task_succ[i].size-1).map do |j|
        @task_pred[@task_succ[i][j]-1].nil? ? @task_pred[@task_succ[i][j]-1] =
[i+1].push : @task_pred[@task_succ[i][j]-1] << i+1
      end
    end
    @task_pred[0] = []
  end
#  Project.new

end
```

ANNEXE 7 – Programme « Algorithme génétique » - Convertisseur

```
# ------------------------------------------------------------------------
# Class Name::    ConversionInterface
# Release::       2.0
# Created by::    Jean-Baptiste Boucherit
# Updated by::    Jean-Baptiste Boucherit
# Affiliation::   Ecole Polytechnique de Montréal
# Creation date:: 2009-11-15
# Extend::        -
# ------------------------------------------------------------------------
# = Changes
#    Release   Changed by   Date         Description
#    =======   ==========   ==========   ===========================
#    1.0       J.Baptiste   2009-11-15   Creation
#    1.1       J.Baptiste   2010-04-03   Add the intopriority method
#    2.0       J.Baptiste   2010-10-04   Update for random projects
#
# = Description
#    Conversion interface
#     * convert a schedule into priority list
#     * convert a priority list into a schedule


autoload(:Project, "initial_project.rb")
#autoload(:Project, "project_generator.rb")
@@areas_capacity = [7,7]
@@project = Project.new

class Array

  def intopriority
    # the schedule you want to convert
#    schedule =
[2,3,7,4,8,9,10,5,11,13,27,12,16,19,18,14,29,21,17,22,26,28,31,6,15,20,23,24,25,30]

    # construction of the priority list
    priority_list = (1..self.size).to_a.reverse.sort_by { |m| self[30-m] }
    puts priority_list.join(',')
  end

  def intoschedule
    # The priority list you want to convert
#    priority_list =
[30,29,27,23,7,28,26,25,24,22,19,21,15,6,18,12,16,17,5,13,11,4,3,2,10,20,9,14,1,8]
    # Schedule the tasks
    scheduled_tasks = [1]
    eligible_tasks = []
    self.size.times do |i|
      pred_scheduled_tasks = [1]
      (2..self.size+1).each do |j|
        essai = []
        @@project.task_pred[j-1].each { |k| scheduled_tasks.include?(k) ? essai <<
"yes" : essai << "no" }
        pred_scheduled_tasks << j if essai.include?("no") == false
      end
      eligible_tasks = (pred_scheduled_tasks - scheduled_tasks).sort_by {|m| -self[m-
2]}
      scheduled_tasks << eligible_tasks[0]
```

```
    end
    schedule = scheduled_tasks + [self.size+2]
    schedule
    puts schedule.join(",")
  end


[30,29,27,23,7,28,26,25,24,22,19,21,15,6,18,12,16,17,5,13,11,4,3,2,10,20,9,14,1,8].into
schedule

[2,3,7,4,8,9,10,5,11,13,27,12,16,19,18,14,29,21,17,22,26,28,31,6,15,20,23,24,25,30].int
opriority


end
```