Titre: Title:	Ordonnancement de projets avec contraintes de ressources dans un contexte incertain
Auteur: Author:	Ariane Duchesne
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Duchesne, A. (2013). Ordonnancement de projets avec contraintes de ressources dans un contexte incertain [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1207/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1207/
Directeurs de recherche: Advisors:	Michel Gamache, & Robert Pellerin
Programme: Program:	Mathématiques appliquées

UNIVERSITÉ DE MONTRÉAL

ORDONNANCEMENT DE PROJETS AVEC CONTRAINTES DE RESSOURCES DANS UN CONTEXTE INCERTAIN

ARIANE DUCHESNE

DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(MATHÉMATIQUES APPLIQUÉES)

AOÛT 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ORDONANNCEMENT DE PROJETS AVEC CONTRAINTES DE RESSOURCES DANS UN CONTEXTE INCERTAIN

présenté par : DUCHESNE Ariane

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

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

M. BAPTISTE Pierre, Doct., président

M. GAMACHE Michel, Ph.D., membre et directeur de recherche

M. PELLERIN Robert, Ph.D., membre et codirecteur

M. LECLERC Guy, Ph.D., membre

REMERCIEMENTS

Je tiens à adresser mes remerciements à l’ensemble des chercheurs de la chaire Jarislowsky / SNC-Lavalin. Le soutien moral ainsi que les conseils avisés de Georges, Kaouthar, Romain, Xavier, Martin et Magno ont été particulièrement utiles lors de ma maîtrise. Je remercie d’autant plus Jean-Mathieu Bieber et François Berthaut pour les solutions au problème déterministe qu’ils m’ont offertes. De plus, je voudrais remercier particulièrement mes directeurs de recherche Michel Gamache et Robert Pellerin pour l’encadrement qu’ils m’ont offert tout au long de ma recherche.

RÉSUMÉ

Les échéanciers sont très importants dans le contexte de la gestion de projet. Le problème d'ordonnancement avec contraintes de ressources consiste à séquencer un ensemble d'activités sujettes à des contraintes de précédence ainsi qu'à une disponibilité limitée des ressources requises pour effectuer ces activités. Ce problème se nomme RCPSP (*Resource Constrained project Scheduling Problem*). L'objectif de ce mémoire est de trouver un ordonnancement admissible robuste qui résiste aux variations dans la durée des activités pour le cas de projets d'envergure. La solution doit être obtenue dans un temps de calcul raisonnable.

L'idée est de partir d'une solution initiale admissible pour le cas déterministe, puis d'ajouter des tampons de temps devant les activités critiques afin de rendre l'ordonnancement robuste. Des améliorations sont apportées à la méthode jugée la plus adéquate de la littérature actuelle. L'heuristique STC (*starting time critical)* est peaufinée en élargissant l'espace des solutions visité. De plus, une étape est ajoutée lors de la génération du réseau du flux des ressources, ce qui permet d'augmenter la flexibilité de ce réseau.

Les résultats sont prometteurs puisque la méthode proposée permet d'obtenir des solutions plus robustes que celles obtenues avec la méthode originale. De plus, les tests montrent qu'il est possible de rendre un ordonnancement significativement plus robuste sans augmenter la durée totale du projet. Finalement, l'importance d'utiliser une solution initiale de courte durée est mise de l'avant.

ABSTRACT

A good schedule is very important in the context of project management. The resource constrained project scheduling problem (RCPSP) consists in sequencing a set of activities subject to precedence constraints and limited availability of the resources required to perform these activities. The purpose of this paper is to find a robust feasible schedule that resists to disturbances in activity durations for the case of large projects. The solution should be fund in a reasonable computation time.

The idea is to start from an initial feasible solution for the deterministic case, then add time buffers in front of critical activities to make the scheduling robust. Improvements are brought to the most appropriate method of the current literature. The Starting Time Critical (STC) heuristic is polished by expanding the visited solution space. In addition, a step is added during the generation of the resource flow network, thereby increasing the flexibility of the network.

The results are promising since the proposed method offers more robust solutions than those obtained by the original method. In addition, tests show that it is possible to obtain a significantly more robust schedule without increasing the project makespan. Finally, the importance of using an initial solution of good quality is put forward.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

a_k	Disponibilité par période de la ressource de type k
A	Ensemble des relations de précédence dû aux contraintes techniques du projet
A_k	Ensemble des relations de précédence dû au transfert de ressources du projet
b	Tampon de temps total ajouté au projet
$buff_i$	Tampon de temps devant l'activité i
$\mathbf{d}_i$	Durée réelle de l'activité i
f_{ijk}	Flux de ressource de type k transféré de l'activité i à l'activité j
$flex$	Flexibilité d'un réseau
K	Ensemble des ressources du projet
$LPL(i, j)$	Longueur du chemin le plus long entre l'activité i et l'activité j dans le réseau de flux de ressource (*Longuest Path Lenght*)
N	Ensemble des activités du projet
Π	Politique d'exécution du projet
$Pred_j$	Ensemble des prédécesseurs de base immédiats i de l'activité j tel que $(i, j) \subset A$
r_{ik}	Quantité par période de ressource de type k nécessaire à l'exécution de l'activité i
req_k	Quantité de ressource de type k encore requis par l'activité j
RFN	Réseau du flux des ressources (*Resource Flow Network*)
s	Activité factice du début 0
s_i	Date début prévue de l'activité i
$\mathbf{s}_i$	Date début réelle de l'activité i
$s_{\max}$	Date de fin prévue du projet pour le cas déterministe

δ	Date de fin prévue du projet pour le cas stochastique
S	Ensemble des dates de début prévues de chaque activité du projet
$\mathbf{S}$	Ensemble des dates de début réelles de chaque activité du projet
S_t	Ensemble des activités en cours à la période t
stc_i	Criticité de l'activité i
STC	Temps de début critique (*Start Time Critical*)
t	Activité factice de la fin $n+1$
T	Ensemble des périodes nécessaire à la réalisation du projet
V_j	Ensemble des activités i tel que $s_i + d_i \leq s_j$, trié en ordre lexicographique
w_i	Poids de l'activité i
x	Solution

LISTE DES ANNEXES

INTRODUCTION

Le problème d'ordonnancement avec contraintes de ressources consiste à ordonner un ensemble d'activités sujettes à des contraintes de précédence ainsi qu'à une disponibilité limitée des ressources requises pour effectuer ces activités. Ce problème se nomme RCPSP (*Resource Constrained project Scheduling Problem*). La complexité du problème empêche l'utilisation d'une méthode de résolution exacte lorsque le nombre d'activités est élevé, comme dans le cas étudié des projets d'envergure.

Dans la réalité, la durée de chaque activité n'est pas connue avec certitude a priori. Des imprévus dans l'exécution du projet engendrent des variations au niveau de la durée des activités. Ainsi, un échéancier construit en considérant les durées des activités comme étant des variables déterministes ne peut pas toujours être respecté lors de l'exécution du projet. Les retards de certaines activités peuvent se propager dans l'ordonnancement et ainsi augmenter considérablement la durée totale du projet.

L'objectif de ce mémoire est donc de trouver un échéancier robuste pour des projets d'envergure. Cet ordonnancement doit résister aux perturbations pour des projets comportant un grand nombre d'activités. De plus, la solution doit être obtenue en quelques heures.

L'approche de résolution proposée consiste d'abord à construire un ordonnancement initial admissible pour le cas déterministe, c'est-à-dire en considérant que les durées des activités sont connues. Ensuite, il suffit d'ajouter des tampons de temps devant les activités les plus critiques afin de rendre l'ordonnancement robuste face à des changements de la durée des activités. La méthode présente dans la littérature s'adaptant le mieux au cas étudié est choisie. Des améliorations y sont apportées permettant ainsi d'obtenir des solutions encore plus robustes.

Ce mémoire se divise en quatre chapitres. Au chapitre 1, une revue de littérature recense les différentes mesures de robustesse ainsi que les méthodes permettant de rendre un ordonnancement robuste. Ensuite, au chapitre 2, la mesure de robustesse utilisée ainsi que la méthode proposée pour résoudre le problème sont détaillées. Le chapitre 3 présente des tests numériques qui permettent de mesurer la performance de la méthode proposée. Finalement, les résultats obtenus sont analysés et les perspectives de recherches énumérées.

CHAPITRE 1 REVUE DE LITTÉRATURE

Il est important d'avoir un bon échéancier afin de planifier et de suivre l'avancement d'un projet. Un projet est composé de plusieurs activités qui sont liées entre elles par des relations de précédence, imposées par des contraintes techniques. Parfois, les activités requièrent des ressources renouvelables disponibles en quantité limitée. Les ressources limitées empêchent certaines activités d'être faites en parallèle. Ce problème constitue le problème d'ordonnancement de projet avec contraintes de ressources (RCPSP). Dans la pratique, il est difficile d'estimer avec précision la durée réelle de chaque activité. C'est pourquoi ce projet s'intéresse au cas où la durée des activités est incertaine. La littérature sur le sujet est vaste et fait l'objet de ce chapitre. L'état de l'art y est présenté en abordant en premier lieu les types d'approches utilisées afin de résoudre ce problème. En deuxième lieu, le concept de robustesse, une mesure couramment employée pour évaluer la qualité du plan de projet, sera défini. Finalement, les méthodes d'insertion de tampon de temps seront décrites plus en détail.

1.1 Types d'ordonnancement pour un contexte incertain

Le problème d'ordonnancement de projet avec contraintes de ressources (RCPSP) est un problème d'ordonnancement d'activités soumis à des contraintes de précédence et où les activités requièrent des ressources renouvelables disponibles en quantité limitée. Le problème déterministe présume une information complète sur l'usage des ressources et la durée des activités. L'objectif est de trouver un ordonnancement admissible qui minimise la durée totale du projet. Le problème RCPSP est un problème NP-dur (Kolisch 1996). Ainsi, trouver la solution optimale aux problèmes de plus de 120 activités est laborieux, car le temps de calcul est très grand (Herroelen 2005).

Dans la pratique, il est difficile d'estimer avec précision la durée de chaque activité (Herroelen 2005; Winch et Kelsey 2005). De plus, plusieurs facteurs peuvent influencer la durée d'une activité, tels que les conditions météorologiques, la disponibilité des ressources, la livraison du matériel, etc. (Deblaere et al. 2011a; Klimek et Łebkowski 2009). C'est pourquoi plusieurs recherches portent sur le cas particulier où les durées des activités sont incertaines.

Un des moyens pour traiter ce problème consiste à trouver un échéancier robuste. Un échéancier est dit robuste si sa durée totale est peu affectée par des événements imprévus rencontrés au cours du projet. La section 1.2 traitera plus en détail la notion de robustesse. D'autres recherches

s'intéressent à la construction d'un échéancier flexible. Un échéancier est dit flexible s'il peut facilement être modifié sans affecter considérablement la durée totale du projet (Aloulou et Artigues 2010; Jensen 2003; Policella et al. 2009). La flexibilité est donc la facilité à reprogrammer les activités suite à des imprévus, alors que la robustesse est la capacité à respecter la programmation initiale même si des imprévus surviennent.

Deux familles d'approches ont été développées afin d'obtenir un échéancier robuste. La première consiste à construire ou à réparer un échéancier de manière réactive au cours de la réalisation du projet. La deuxième famille d'approches consiste à déterminer au début du projet un échéancier robuste qui résistera aux perturbations rencontrées lors de la réalisation du projet. La première famille d'approches est utilisée dans des environnements très dynamiques, alors que la deuxième famille d'approches est préférée lorsque les perturbations sont moins grandes (Klimek et Łebkowski 2009). Les deux approches peuvent toutefois se compléter. Un ordonnancement prédictif est alors construit durant la phase de planification du projet pour être accompagné d'une phase d'ordonnancement réactif lors de l'exécution du projet (Van de Vonder et al. 2007b). Cela permet de réparer l'ordonnancement prédictif à la suite d'importants bouleversements dans le projet.

1.1.1 Ordonnancement réactif

Plusieurs articles portent sur l'ordonnancement réactif (Deblaere et al. 2011b; Van de Vonder et al. 2007a; Vieira et al. 2003). Alsakini et al. (2004) soutiennent que de continuer une planification détaillée au cours de l'exécution d'un projet permet de prendre en compte les nouvelles informations acquises lors de l'exécution du projet. Ainsi, il est possible de détecter les perturbations plus tôt et d'agir plus rapidement afin de limiter leurs impacts.

L'ordonnancement réactif consiste à traiter les activités du projet selon une politique d'exécution préétablie. Une telle politique consiste à définir des règles de priorité entre les activités afin de déterminer la prochaine activité à exécuter. Ces règles de priorité sont appliquées lors d'un point de décision au cours de la réalisation du projet, c'est-à-dire à chaque fois qu'une activité termine. En terminant, une activité libère les ressources qu'elle accaparait. À chaque point de décision, les activités candidates à débuter sont celles dont tous les prédécesseurs ont été traités. S'il y a plusieurs activités candidates mais que les ressources sont insuffisantes, une règle de priorité est appliquée afin de déterminer quelle(s) activité(s) débute(nt). Plusieurs critères de sélection

existent : choisir d'abord l'activité dont la durée est la plus courte (*Shortest Processing Time*) ou bien celle dont la date requise ou de livraison est la plus tôt (*Earliest Due Date*). Il peut y avoir un échéancier de base (Klimek et Łebkowski 2009) ou non (Chtourou et Haouari 2008). S'il y a un échéancier de base, il doit être réparé en cas de perturbations dans le projet. Cela peut aller du simple décalage des activités au réordonnancement complet du projet. En ce sens, des points de contrôle au cours du projet permettent d'amener les correctifs nécessaires à l'échéancier (Raz et Erel 2000). Un des objectifs de l'ordonnancement réactif est la stabilité, c'est-à-dire de conserver le plus possible l'ordre d'exécution des activités (Klimek et Łebkowski 2009). En effet, minimiser les changements dans l'échéancier permet de réduire les perturbations dans le projet.

1.1.2 Ordonnancement prédictif

Dans la pratique, il est préférable d'avoir un ordonnancement prédictif sur lequel se reposer. Aytug et al. (2005) démontrent qu'un bon échéancier est un outil précieux pour les gestionnaires de projet. En effet, cela permet d'avoir des dates fiables de fin d'activité à communiquer au client et de s'entendre sur des dates de livraison avec les fournisseurs. C'est pourquoi plusieurs approches consistent à trouver un ordonnancement prédictif. En anglais, l'ordonnancement prédictif est appelé *predictive schedule* ou *baseline schedule* (Klimek et Łebkowski 2009). Puisque cet ordonnancement est construit durant la phase de planification du projet, la littérature parle d'*offline scheduling*, en opposition au *online scheduling* utilisé pour l'ordonnancement réactif.

1.2 Robustesse

Il existe plusieurs manières de définir la robustesse d'un ordonnancement. Al-Fawzan et Haouari (2005) définissent la robustesse d'un échéancier comme étant sa capacité à faire face à de petites augmentations de la durée de certaines activités dues à des facteurs incontrôlables. Chtourou et Haouari (2008) proposent douze mesures de la robustesse permettant de quantifier la robustesse d'un ordonnancement. Klimek et Lebkowski (2011) proposent deux mesures de la robustesse basées sur l'impact de l'augmentation de la durée de chaque activité d'une unité de temps.

Pour leur part, Van de Vonder et al. (2006) définissent deux types de robustesse. La robustesse de la qualité (*quality robustness*) et la robustesse de la solution (*solution robustness*). La robustesse de la qualité fait référence à l'invariabilité de la durée totale du projet lors de perturbations. Elle

peut être calculée par la probabilité que la date de fin du projet $\mathbf{s}_{n+1}$ ne dépasse pas la date de livraison du projet δ : $P(\mathbf{s}_{n+1} \le \delta)$, où $n+1$est l'activité factice représentant la fin du projet. La robustesse d'une solution, ou la stabilité d'un ordonnancement, fait référence à la différence entre l'ordonnancement planifié et l'ordonnancement effectué. Elle peut être calculée comme $\Delta(S,\mathbf{S}) = \sum w_i E\,|\,\mathbf{s}_i - s_i\,|$ où s_i est le temps de début planifié de l'activité i dans l'ordonnancement de base planifié S et $\mathbf{s}_i$ est une variable aléatoire représentant le temps de début de l'activité i dans l'ordonnancement effectué $\mathbf{S}$. Il s'agit donc de calculer la somme des différences pondérées entre le temps de début planifié et le temps de début réalisé de chaque activité. Le poids d'une activité w_i représente le coût marginal associé au fait de commencer l'activité avant ou après sa date de début prévue. Ce coût peut inclure les coûts d'entreposage, les coûts reliés au changement, les coûts reliés aux arrangements contractuels avec les sous-traitants, etc. (Van de Vonder et al. 2007b).

Ghezail et al. (2010) proposent une approche complémentaire à l'analyse quantitative, soit l'analyse qualitative de la robustesse. Les auteurs affirment que mesurer la robustesse avec un seul nombre peut être restrictif. Ils introduisent donc deux types de graphiques pour représenter la robustesse de l'ordonnancement. Le premier montre la capacité de l'ordonnancement à absorber les perturbations, c'est-à-dire la durée des variations qui n'affectent pas l'ordonnancement. Le deuxième expose les conséquences des perturbations, c'est-à-dire le retard engendré par une variation d'une certaine durée. Ces graphiques sont un outil permettant d'analyser visuellement la robustesse d'un projet.

1.3 Trouver un ordonnancement robuste

Plusieurs techniques peuvent être utilisées pour construire un ordonnancement prédictif robuste. La première approche consiste à minimiser la durée du projet en tenant compte de la variabilité de la durée des activités. La deuxième approche, l'ordonnancement proactif, consiste à insérer des tampons de temps dans un ordonnancement déterministe de durée minimale.

1.3.1 Ordonnancer en tenant compte de la variabilité des durées

Un bon ordonnancement doit être de courte durée et les dates prévues pour le commencement des activités doivent être respectées. Bien que ces deux objectifs soient contradictoires, certaines approches tentent d'optimiser les deux objectifs simultanément.

1.3.1.1 Cas sans contraintes de ressources

Barraza (2011) présente une méthode pour le cas où il n'y a pas de contraintes de ressources. Cette méthode permet de prendre en compte la variabilité de la durée des activités et d'ajouter de la contingence aux activités. D'abord, les durées des activités sont estimées par leur durée médiane. Ensuite, la simulation de Monte Carlo est utilisée afin d'estimer la contingence à ajouter au projet. Finalement, cette contingence est répartie entre les activités de manière à ce qu'elles aient toutes la même probabilité de ne pas excéder leur durée planifiée.

1.3.1.2 Ordonnancement par étapes multiples

L'ordonnancement par étapes multiples considère le RCPSP comme un processus de décisions en plusieurs étapes (multi-étages). L'exécution du projet se fait selon une politique d'ordonnancement qui agit comme une règle de priorité à chaque point de décision. Un point de décision est un point où une ou plusieurs activités peuvent commencer. S'il n'y a pas assez de ressources disponibles, les activités sélectionnées sont celles ayant la plus haute priorité. Deux heuristiques très utilisées pour le problème déterministe sont l'algorithme sériel et l'algorithme parallèle (*serial and parallel schedule generation scheme*) (Kolisch 1996). L'algorithme sériel passe en revue les activités en ordre de priorité. L'ordonnancement des activités s'effectue selon leur date de début au plus tôt, en respectant les contraintes de précédence et de ressource. L'algorithme parallèle avance de la date de début du projet jusqu'à la date de fin et programme, pour chaque date, les activités qui respectent les contraintes de précédence et de ressources. Si plusieurs activités peuvent commencer à la même date mais que les ressources sont insuffisantes, les activités sont sélectionnées selon l'ordre de priorité. Plusieurs approches consistent à adapter ces heuristiques au cas où les durées des activités sont incertaines.

Baradaran et al. (2010) s'intéressent au problème de type PERT avec contraintes de ressources. L'heuristique *Scatter Search* cherche un ordonnancement qui minimise la durée totale du projet.

Une solution est représentée par une liste des activités et l'ordonnancement est effectué selon l'algorithme sériel ou parallèle. La durée totale est évaluée par des simulations de Monte Carlo.

Rabbani et al. (2007) présentent une méthode combinant l'ordonnancement par étapes multiples et les principes de la chaîne critique. Ils introduisent une nouvelle règle de priorité utilisée aux points de décision. La priorité d'une activité est la multiplication entre sa durée moyenne, sa criticité et sa crucialité. Les auteurs définissent la criticité comme étant la probabilité que l'activité soit sur la chaîne critique et la crucialité comme étant le coefficient de corrélation entre la durée de l'activité et la durée du projet. La chaîne critique est déterminée et un tampon de temps est ajouté à la fin du projet. L'objectif est de réduire la durée totale du projet ainsi que sa variance.

Bhaskar et al. (2011) proposent une méthode pour résoudre le RCPSP avec des durées d'activités floues (*fuzzy*). Le chemin critique est calculé à l'aide des nombres flous. La criticité d'une activité est la longueur du plus long chemin partant de cette activité vers la fin du projet. La méthode utilise l'algorithme parallèle où la règle de priorité est la criticité des activités.

Bruni et al. (2011) ont développé une heuristique qui utilise le concept des probabilités jointes pour trouver un ordonnancement qui a une bonne probabilité d'être respecté. L'heuristique SDGS (*stochastic dynamic generation scheme*) vise la robustesse de la solution. Cette heuristique est inspirée de l'algorithme parallèle. L'heuristique SDGS suppose que les fonctions de distribution des durées des activités sont connues. À chaque point de décision, lorsqu'une ou des activités débutent, un problème de probabilité conditionnelle (*joint probability*) est résolu. Il permet de choisir la durée des activités débutées de façon à ce que la probabilité que ces activités retardent le reste du projet n'excède pas le facteur de risque choisi. Cette méthode offre l'avantage de résoudre le RCPSP tout en trouvant une solution robuste. Cependant, la qualité des solutions trouvées par cette méthode, en termes de durée totale du projet, est comparable à celle des solutions trouvées par l'algorithme parallèle. C'est-à-dire que la durée totale du projet trouvée est souvent loin de la durée minimale. De plus, cette méthode considère que les activités ont toutes le même poids. Pour une instance où une seule activité commence à chaque point de décision, cette méthode revient à ajouter de la contingence à chaque activité individuellement.

1.3.1.3 Détermination d'intervalles

Bendavid et Golany (2009) utilisent la méthode appelée entropie croisée (*Cross-Entropy)* pour déterminer pour chaque activité un moment au plus tôt (*gate*) où celle-ci peut débuter. Des coûts

sont associés à chaque activité si elle commence avant ou après le moment prévu. L'objectif est de minimiser la somme des coûts espérés. Bendavid et Golany (2011) améliorent leur modèle précédent afin de déterminer pour chaque activité un intervalle de temps dans lequel elle peut commencer. Il y a une pénalité si, lors de l'exécution, l'activité est prête à commencer avant l'intervalle de temps ou si elle commence après l'intervalle. Plus l'intervalle de temps est grand, plus le coût de l'intervalle est élevé. L'objectif consiste à minimiser la somme des coûts espérés, c'est-à-dire de minimiser les pénalités par rapport à la date de début de l'activité et les coûts relatifs à la grandeur de l'intervalle.

1.3.1.4 Méthodes avec objectif double

Al-Fawzan et Haouari (2005) tentent de résoudre le problème d'ordonnancement en minimisant la durée totale du projet et en maximisant la robustesse de la solution en même temps. Les auteurs présentent un algorithme de recherche taboue à objectifs multiples qui génère un ensemble de bonnes solutions au RCPSP en considérant deux objectifs. Le premier est de minimiser la durée du projet et le deuxième est de maximiser la somme des marges libres des activités. Abbasi et al. (2006) reprennent cette idée, mais utilisent le recuit simulé pour résoudre le problème.

Chtourou et Haouari (2008) ont développé un algorithme en deux étapes pour trouver un ordonnancement robuste. La première étape permet de résoudre le RCPSP en minimisant la durée totale du projet. Ils utilisent une heuristique basée sur des règles de priorité (*enhanced multi-pass random-biased serial schedule generation scheme*). La deuxième phase résout le problème en maximisant la robustesse. La durée totale trouvée lors de la première phase est utilisée comme seuil acceptable lors de la deuxième phase. Ainsi, les seules solutions retenues sont celles dont la durée totale ne dépasse pas celle trouvée lors de la première phase.

1.3.2 Méthodes d'insertion de tampons de temps

Une autre approche est de partir avec un ordonnancement déterministe de durée minimale et d'y ajouter des tampons de temps. Les durées des activités utilisées pour l'ordonnancement déterministe sont les durées moyennes. L'insertion de tampons de temps protège l'arrangement prévu des activités contre un retard dans l'exécution d'une activité. Cependant, cela augmente la durée totale du projet, c'est-à-dire que cela engendre une détérioration de la qualité de l'ordonnancement.

1.3.2.1 Méthode de la chaîne critique

La technique d'insertion de tampons de temps la plus connue est la méthode de la chaîne critique et de gestion des tampons de temps (*Critical Chain/buffer management CC/BM*) introduite par Goldratt (1997). Les hypothèses et les principes derrière l'approche de la chaîne critique sont discutés dans Herman (2001). L'idée est d'utiliser une estimation optimiste de la durée des activités basée sur la durée moyenne ou médiane des activités. La contingence de temps est concentrée dans un tampon de temps à la fin du projet (*project buffer*) et dans des tampons de temps devant les activités les plus critiques (*feeding buffers*).

La méthode consiste à résoudre le RCPSP déterministe à l'aide d'une heuristique puis d'ajouter des tampons de temps à la solution initiale. Dans un ordonnancement admissible, une chaîne est définie comme une suite d'activités reliées par des contraintes de précédence et/ou qui dépendent de la même ressource. La chaîne critique est la plus longue chaîne d'activités, soit la chaîne qui détermine la durée du projet. S'il existe plusieurs chaînes critiques, l'une est choisie au hasard. Un tampon de temps est ajouté à la fin de la chaîne critique pour protéger la date de fin du projet contre les perturbations dans la chaîne critique. Aussi, lorsqu'une chaîne non critique rejoint la chaîne critique, un tampon de temps est ajouté à la fin de la chaîne non critique. Cela permet d'éviter que les perturbations dans la chaîne non critique se propagent dans la chaîne critique.

Dans la version originale, la longueur d'un tampon de temps correspond à la moitié de la longueur de la chaîne qui le précède. C'est-à-dire qu'il est égal à la moitié de la durée de la chaîne qu'il protège. Leach (2005), Newbold (2008), Tukel et al. (2006) ainsi que Zhang et al. (2011) proposent des procédures alternatives pour déterminer la longueur des tampons de temps. Entre autres, Long et Ohsato (2008) proposent une méthode pour déterminer la grandeur du tampon de temps à la fin du projet à l'aide des nombres flous (*fuzzy numbers*).

Il existe aussi des tampons pour les ressources (*resource buffers*). Ces tampons sont habituellement sous forme d'avertissements préalables. Ils sont placés lorsqu'une activité de la chaîne critique utilise une ressource différente de son prédécesseur. Cela permet d'assurer que la ressource est disponible lorsqu'elle est nécessitée par une activité de la chaîne critique. Lors de l'exécution du projet, les gestionnaires suivent l'évolution de la consommation des tampons de temps. Il y a un avertissement lorsqu'un tampon de temps a été consumé jusqu'à une certaine quantité. Si la situation se détériore au-delà d'un point critique, une action corrective est prise.

Les jalons ne débutent pas avant leur date de début planifiée, mais les autres activités débutent le plus tôt possible (*roadrunner mentality*).

Plusieurs logiciels utilisant cette méthode ont été commercialisés, notamment *Prochain*®. Bien que la méthode de la chaîne critique donne de bons résultats, Herroelen et Leus (2001) expliquent que la simplicité de la méthode est dangereuse. Ils affirment que de trouver la solution déterministe à l'aide d'un meilleur algorithme, comme la méthode *branch-and-bound*, améliore le résultat. De plus, ils indiquent que la méthode peut engendrer des tampons de temps inutilement grands qui ne protègent pas bien la date de fin du projet. Finalement, ils proposent une mise à jour régulière de l'ordonnancement afin d'obtenir une durée totale plus courte et d'avoir de meilleurs estimés intermédiaires de la date de fin du projet.

Zhang et al. (2011) ont aussi étudié l'interaction entre la grandeur des tampons de temps et le niveau de service. Ils en arrivent à la conclusion que les tampons de temps n'ont pas à être aussi grands que ce qui est suggéré dans la littérature. De plus, bien que cette technique protège la date de fin du projet, elle ne permet pas de connaître a priori la date de début des différentes activités du projet. La stabilité des dates des activités est nécessaire dans certains environnements. Par exemple, il se peut que du matériel doive être livré exactement au commencement d'une activité. C'est pourquoi il est nécessaire de développer des algorithmes d'insertion de tampon de temps qui offre une stabilité lors de l'exécution du projet.

1.3.2.2 Méthode d'insertion de tampon de fiabilité

Park et Pena-Mora (2004) présentent une méthode d'insertion de tampon de temps qu'ils ont testée sur un projet de construction de pont. La méthode d'insertion de tampons de fiabilité (*reliability buffering*) fonctionne de la manière suivante. D'abord, la contingence dans la durée des activités est ignorée. La grandeur du tampon de temps est déterminée selon les résultats de simulation, les caractéristiques des activités (type de production, fiabilité et sensibilité) et les politiques de contrôle du projet. Les tampons de fiabilité sont insérés entre les activités, au début du successeur. Le tampon de temps est caractérisé comme étant un temps pour rassembler les ressources de l'activité et pour trouver les problèmes survenus lors des activités précédentes. Cela permet d'éviter le syndrome de la dernière minute. La localisation et la taille des tampons de temps sont mises à jour dynamiquement durant le projet.

1.3.2.3 Méthodes visant la robustesse de solution

Van de Vonder et al. (2005) concluent qu'une heuristique visant la robustesse de solution est souvent préférable. Ces méthodes permettent de protéger la date de début de toutes les activités d'un projet.

Ces heuristiques utilisent le concept du réseau de flux de ressources (*resource flow network-RFN*) développé par Artigues et al. (2003). Lors de l'exécution d'un projet, chaque ressource est transférée d'une activité à une autre. Un RFN est un réseau du flux des ressources entre les activités pour un ordonnancement déterministe donné. Il peut exister plusieurs RFN pour un même ordonnancement. L'idée est de sélectionner un RFN quelconque, qui contient une contrainte de précédence entre deux activités qui utilisent la même ressource l'une après l'autre. Avec ces nouvelles contraintes de précédence, il est possible de déplacer les dates des activités tout en conservant la faisabilité de l'ordonnancement. Bien que l'espace des solutions soit réduit, cela permet d'insérer des tampons de temps entre les activités sans avoir à résoudre de nouveau le RCPSP. Leus et Herroelen (2004) présentent une méthode de *Branch-and-Bound* permettant de trouver tous les RFN d'un échéancier dans le cas où n'y a qu'un seul type de ressource. Cela permet de choisir le RFN le plus robuste, mais nécessite un important temps de calcul. Klimek et Lebkowski (2011) résument les principes d'allocation des ressources qui influencent la robustesse d'un RFN.

Van de Vonder et al. (2006) introduisent et comparent différentes heuristiques permettant d'insérer des tampons de temps dans un ordonnancement. L'heuristique RFDFF (*resource flow-dependent float factor*) considère le poids des activités. Cette méthode permet d'insérer des tampons de temps plus longs devant les activités qui engendreraient un coût élevé si elles étaient effectuées plus tôt ou plus tard que prévu. L'heuristique VADE *(virtual activity duration extension)* augmente virtuellement la durée prévue des activités selon la variance de la durée des activités. L'heuristique STC (*starting time critical)* exploite à la fois les informations sur le poids et sur la variance de la durée de l'activité. Un tampon de temps est inséré devant les activités jugées les plus critiques de façon à protéger leur date de début. Les auteurs introduisent aussi une procédure de recherche taboue. La solution de départ est un ordonnancement qui contient déjà des tampons de temps. L'amélioration de la solution se fait en augmentant ou en diminuant d'une unité la grandeur des tampons de temps devant les activités. La valeur des solutions est estimée par simulation.

Van de Vonder et al. (2008) ont effectué une analyse approfondie, à l'aide de la simulation, de la performance de ces heuristiques. L'expérience révèle que l'heuristique STC est généralement la meilleure parmi les heuristiques qui ne comportent pas de phase d'amélioration. Pour ceux qui reposent sur une phase d'amélioration, c'est la recherche taboue qui donne les meilleurs résultats. Cependant, la recherche taboue demande beaucoup de temps de calcul. Un ordonnancement construit par l'heuristique STC et amélioré par une descente (STC-D) donne presque le même coût de stabilité espéré que la procédure de recherche taboue, mais requiert beaucoup moins d'efforts de calculs.

Schatteman et al. (2008) ont développé un outil basé sur l'heuristique STC. C'est une méthode intégrée de gestion des risques et d'ordonnancement proactif pour ordonnancer des projets de construction en contexte incertain. Un ordonnancement proactif est construit par l'heuristique STC en utilisant les informations générées lors de l'évaluation des risques. Les résultats obtenus lors de l'implémentation de cette méthode pour de vrais projets sont très prometteurs.

Deblaere et al. (2011a) introduisent une heuristique basée sur la simulation. Une solution est représentée par une politique d'exécution et un vecteur des dates de début prévues des activités. La simulation de la réalisation du projet permet de calculer la robustesse de la solution. Lors de la réalisation du projet, une activité ne peut pas commencer avant sa date de début prévue. Des tampons de temps sont ajoutés aux endroits qui permettent d'augmenter le plus la robustesse de la solution. La robustesse est évaluée par simulation et une unité de temps tampon est ajoutée devant une activité si cela réduit le coût simulé de la somme pour chaque activité du nombre de jours de retards multiplié par le poids de l'activité. Ensuite, une phase d'amélioration permet d'augmenter ou de diminuer la grandeur des tampons de temps devant les activités et de modifier la liste de priorité des activités. Leurs expérimentations montrent que leur procédure *Simulation-based Descent* (SBD) est plus performante que la procédure STC-D, mais avec un temps de calcul plus grand.

1.3.2.4 Variantes du problème

Il existe des variantes du RCPSP avec durées d'activités incertaines. Le problème d'allocation robuste des ressources traite le cas où la disponibilité des ressources est incertaine. Une autre variante traite le cas où il y a des jalons intermédiaires.

La première variante est le problème d'allocation robuste des ressources. Ce problème s'intéresse au cas où il y a des ruptures dans la disponibilité des ressources. Xiuming et al. (2010) présentent

trois approches permettant de rendre plus robuste un ordonnancement dans ce cas. D'abord, un ordonnancement de base peut être construit en utilisant une heuristique qui tient compte de la demande en ressources de chaque activité et de la disponibilité des ressources. Ensuite, des ressources supplémentaires peuvent être ajoutées pour prendre la relève lorsqu'une ressource n'est plus disponible. Finalement, il est possible d'ajouter des tampons de temps devant les activités dont la durée augmentera en cas de rupture de ressource. Lambrechts et al. (2011) ont analysé l'augmentation de la durée d'une activité due à un manque de ressource. Ils ont utilisé ces résultats pour créer des algorithmes permettant d'ajouter des tampons de temps à un ordonnancement afin de le protéger contre la propagation des perturbations dues à un manque de ressources. L'insertion de tampons de temps basée sur la simulation, l'insertion de tampons de temps utilisant des mesures substituts et l'insertion de tampons de temps utilisant l'heuristique STC ont été comparées. C'est l'insertion de tampons de temps basée sur la simulation qui donne les meilleurs résultats, mais cela demande beaucoup de temps de calcul. Les auteurs suggèrent plutôt d'implémenter l'insertion de tampons de temps basée sur leur première mesure substitut (*Surr*1) ou sur l'heuristique STC. L'heuristique STC a l'avantage d'être aussi une bonne stratégie lorsque la variance dans la durée des activités est prise en compte. Ce mémoire se concentre sur l'incertitude au niveau de la durée des activités et suppose que les ressources sont toujours disponibles.

La deuxième variante est le problème comportant des jalons intermédiaires. Ces jalons sont des dates qu'il est important de respecter durant le projet. Klimek et Łebkowski (2009) ont développé un algorithme d'insertion de tampons de temps pour le cas où il y a plusieurs jalons à respecter au cours d'un projet. Ces jalons peuvent représenter, par exemple, une échéance contractuelle pour la fin de chaque phase du projet. L'auteur suggère que la définition de jalons intermédiaires permet de réduire le risque de retard du projet pour des projets de grande envergure. Il introduit une nouvelle mesure de robustesse permettant de protéger les différents jalons. Cependant, cette méthode ne protège que la date des jalons.

1.4 Limitations

Il est préférable d'avoir un bon ordonnancement prédictif auquel se référer lors de l'exécution du projet. Cela permet, entre autre, de mesurer la progression du projet. Un ordonnancement produit au début du projet procure un avantage certain au niveau du contrôle de projet, mais seulement

s'il est possible de le réaliser. En effet, si l'ordonnancement est trop optimiste ou trop pessimiste, le projet dévie de l'ordonnancement de base et celui-ci perd de son utilité. C'est pourquoi il est nécessaire que cet ordonnancement soit robuste.

Ordonnancer les activités en considérant une durée déterministe permet d'utiliser les méthodes d'optimisation les plus efficaces pour minimiser la durée totale du projet. Il est ensuite possible de choisir l'extension permise de la durée du projet. C'est pourquoi une méthode ajoutant des tampons de temps dans un ordonnancement de durée minimale déterministe est privilégiée. Des applications pratiques de cette technique montrent qu'elle permet de réduire le nombre de projets en retard et par le fait même, de réduire les dépassements de coûts. La méthode de la chaine critique n'est pas utilisée car elle ne permet pas de protéger la date de début des activités du projet. Elle protège seulement la date de fin. Une méthode visant la robustesse de solution est préférable.

Dans la littérature, les méthodes qui utilisent la simulation pour évaluer la robustesse sont celles qui obtiennent les meilleurs résultats. Cependant, la simulation demande beaucoup de temps de calcul pour produire une valeur de robustesse précise. En effet, il y a beaucoup d'aléatoire introduit par la variation de la durée des activités. Un nombre important de répétitions doit donc être effectué afin d'obtenir une valeur simulée moyenne stable, ce qui requiert beaucoup de calculs. Cette approche n'est donc pas souhaitable dans ce mémoire puisqu'il s'intéresse à l'ordonnancement de projets d'envergure. Le nombre important d'activités nécessite le développement d'une méthode demandant peu de temps de calcul. Deblaere et al. (2011a) ont effectué l'évaluation de la robustesse d'une solution d'un réseau de 120 activités en simulant 1000 répétitions de l'exécution du projet. Ils obtiennent une erreur relative moyenne de 0,95%. L'ajout de tampons de temps dans une solution déterministe est effectué itérativement en augmentant ou en diminuant d'une unité de temps la grandeur du tampon de temps d'une des activités du projet. Pour un projet de 120 activités ayant une durée de plus de 100 jours, il est légitime de penser qu'une telle modification n'aura pas toujours un impact de plus de 1% sur la valeur de robustesse de la solution. Ainsi, il ne sera pas toujours possible d'évaluer lors d'une itération s'il s'agit d'une amélioration ou d'une dégradation de la solution.

Une autre méthode, l'heuristique STC, utilise une fonction objective approchée pour évaluer la robustesse d'une solution. Comme cette méthode n'est pas basée sur la simulation, les temps de

calcul sont beaucoup plus petits. Cependant, bien que cette méthode donne de bons résultats, elle n'est pas aussi efficace qu'une méthode basée sur la simulation.

1.5 Objectif de la recherche

En regard de la revue de littérature, les questions de recherche suivantes apparaissent :

- Quelle est la meilleure façon d'évaluer la robustesse d'une solution sans toutefois que les temps de calcul soient trop grands ?
- Comment bénéficier de la rapidité de l'heuristique STC tout en obtenant une solution d'une qualité comparable à celle obtenue à l'aide de la simulation?

Compte tenu de l'importance de ces deux questions de recherche, l'objectif général de ce mémoire consiste à développer une méthode permettant d'insérer des tampons de temps dans un ordonnancement déterministe d'un projet d'envergure afin d'obtenir une solution robuste. Cette solution doit être obtenue en quelques heures de calcul. Durant la phase de planification du projet, il est acceptable d'utiliser un algorithme qui nécessite quelques heures de calcul. Au contraire, lorsque l'échéancier doit être réparé au cours de l'exécution du projet, les planificateurs n'allouent pas autant de temps à la résolution du problème. Ce mémoire se concentre sur l'ordonnancement initial effectué durant la phase de planification du projet.

Pour atteindre cet objectif général, deux objectifs spécifiques ont été formulés :

1- Définir une méthode d'évaluation de la robustesse d'un ordonnancement ;
2- Développer une heuristique permettant de trouver efficacement une solution robuste pour un problème de grande taille.

Suivant la revue de l'état de la recherche pour les méthodes d'ordonnancement en contexte incertain, le prochain chapitre présente la méthode proposée. Cette méthode part d'une solution déterministe et insère des tampons de temps permettant à l'ordonnancement de résister aux variations dans la durée des activités.

CHAPITRE 2 MODÈLE D'ORDONNANCEMENT ROBUSTE

Ce chapitre présente le modèle d'insertion de tampon de temps proposé. La première section décrit le problème étudié et définit les variables utilisées. La seconde section explique la méthode proposée pour rendre un ordonnancement robuste.

2.1 Définition du problème

Un projet est un ensemble d'activités exécutées à l'aide de ressources renouvelables. Le problème d'ordonnancement avec contraintes de ressource peut être représenté par un graphe connexe, orienté et acyclique $G(N, A)$. Les nœuds $N = \{0,1,...,n+1\}$ représentent les activités. Ce réseau d'activités sur les nœuds (*ActivityOnNode - AON*) est appelé *operation network*. Il existe deux activités factices (*dummy activities*) représentant le début et la fin du projet, 0 et $n+1$ respectivement. Ces jalons ont une durée déterministe de 0 unité de temps. La durée d'une activité non factice i est représentée par une variable entière stochastique $\mathbf{d}_i$ dont la distribution discrète est connue. La date de début réalisée d'une activité i est notée $\mathbf{s}_i$. Les arcs $(i, j) \in A$ représentent les relations de précédence entre les activités, où l'activité i précède l'activité j. Ces relations sont des relations fin-début sans décalage. Il en résulte la contrainte suivante:

$$\mathbf{s}_i + \mathbf{d}_i \leq \mathbf{s}_j \qquad \forall (i, j) \in A \tag{2.1}$$

Il existe un ensemble de ressources renouvelables K dont la disponibilité par période est $a_k, \forall k \in K$. Chaque activité i nécessite une quantité de ressources par période r_{ik}, où $i \in N$ et $k \in K$. Cette quantité est constante durant toute la durée de l'activité. Ainsi, la contrainte suivante doit être respectée :

$$\sum_{i \in S_t} r_{ik} \leq a_k \qquad \forall t \in T, \forall k \in K \tag{2.2}$$

où S_t est l'ensemble des activités en cours à la période t, T est l'ensemble des dates, discrétisées en terme de jours, entre le début du projet $t = 0$ et la fin du projet $t = \mathbf{s}_{n+1}$.

Il existe un ordonnancement prédictif construit avant d'amorcer le projet. Cet ordonnancement est un ensemble de la date de début prévue s_i de chaque activité i. Un poids w_i est associé à

chaque activité i. Ce poids représente le coût engendré si l'activité débute avant ou après sa date de début prévue s_i. Ce coût peut inclure les coûts d'entreposage, les coûts reliés au changement, les coûts reliés aux arrangements contractuels avec les sous-traitants, etc. Un bon ordonnancement prédictif est un ordonnancement qui résiste aux perturbations lors de l'exécution du projet. Au chapitre 1, plusieurs façons d'évaluer la robustesse d'une solution ont été présentées. La mesure de robustesse sélectionnée dans ce mémoire est celle présentée par Van de Vonder et al. (2007b). Ainsi, la robustesse d'un ordonnancement x est définie par la somme des différences pondérées entre la date de début prévue et la date de début réalisée de chaque activité:

$$f(x) = \sum_{i \in N} w_i E\,|\,\mathbf{s}_i - s_i\,|. \qquad (2.3)$$

La date de début réalisée des activités n'étant pas connue à priori, cette fonction sera évaluée à l'aide de la simulation. Aussi, afin d'alléger le texte, le mot robustesse sera utilisé pour désigner cette fonction objectif dans la suite de ce mémoire.

2.2 Méthode proposée

La réalisation d'un projet est considérée comme un processus dynamique qui suit une politique d'exécution. Cette politique consiste à considérer les activités en ordre de priorité et à les programmer selon leur date de début prévue. Un exemple d'une politique d'exécution serait l'algorithme parallèle. Ainsi, il est possible de représenter un ordonnancement x par deux ensembles $x = (\Pi, S)$, où $\Pi = (\pi_1, \pi_2, ..., \pi_n)$ est l'ensemble des priorités des activités et $S = (s_1, s_2, ..., s_n)$ est l'ensemble des dates de début prévues des activités. Il n'est pas nécessaire de représenter les activités factices dans ces ensembles. En effet, l'activité 0 commence (et finit) toujours au temps 0. La date de début (et de fin) prévue de l'activité $n+1$, notée δ, est une donnée du problème. Lors de la réalisation du projet, l'activité $n+1$ commence (et finit) lorsque toutes les activités sont terminées. Une priorité est superflue puisque, par définition, aucune autre activité ne peut être faite en parallèle.

Une méthode est proposée afin d'insérer des tampons de temps dans un ordonnancement déterministe. L'idée est de trouver une méthode permettant d'obtenir rapidement une solution robuste pour des instances comportant un grand nombre d'activités.

2.2.1 Construction de la solution

L'objectif est de construire une solution x robuste. Pour ce faire, il faut choisir l'ensemble des priorités des activités Π et l'ensemble des dates de début prévues des activités S. Puisque le nombre d'activités est grand, il faut trouver une méthode qui donne une bonne solution sans toutefois nécessiter un long temps de calcul.

En premier lieu, les dates de début prévues des activités S sont trouvées par l'heuristique STC (S*tarting Time Critical)*, développé par Van de Vonder et al. (2008). Cette heuristique est la méthode obtenant les solutions les plus robustes parmi les méthodes qui ne sont pas basées sur la simulation. Une légère modification est apportée à l'heuristique, ce qui permet d'améliorer la qualité de la solution trouvée.

Cette méthode exploite l'information sur le poids et sur la variance de la durée des activités. Elle commence d'abord avec un ordonnancement déterministe de durée minimale. La date de fin du projet est repoussée de façon à ajouter un tampon de temps total au projet. Ensuite, l'heuristique ajoute itérativement une période devant l'activité la plus critique de la solution courante, tout en gardant la date de fin prévue du projet fixe. La procédure termine lorsque la robustesse ne peut plus être augmentée. La criticité d'une activité j est définie par la probabilité que sa date de début réalisée soit plus grande que sa date de début prévue, le tout pondéré par le poids de l'activité :

$$stc_j = P\left(\mathbf{s}_j > s_j\right) \cdot w_j \tag{2.4}$$

La probabilité du retard d'une activité ne pouvant pas être calculé à priori, l'approximation suivante est utilisée:

$$stc_j \approx \sum_{i \in P_j} P\left(s_j \geq s_i + \mathbf{d}_i + LPL(i, j)\right) \cdot w_j \tag{2.5}$$

où P_j est l'ensemble de tous les prédécesseurs immédiats et transitifs de j dans le réseau de flux de ressources (RFN). $LPL(i, j)$ est la longueur du chemin le plus long entre l'activité i et l'activité j dans ce réseau. Lors du calcul, la durée des activités se trouvant entre i et j est leur durée déterministe. L'heuristique STC évalue la robustesse d'un ordonnancement par la somme des STC :

$$f(x) = \sum_{j \in N} stc_j = \sum_{j \in N} w_j \sum_{i \in P_j} P\left(\delta_i > s_j - s_i - LPL(i,j)\right) \qquad (2.6)$$

Une valeur près de zéro indique une solution robuste alors qu'une valeur élevée signifie que la solution est fragile face aux situations imprévues. L'algorithme est décrit à la figure 2.1, où $buff_j$ est la grandeur du tampon de temps devant l'activité j :

Algorithme *STC*

1. Pour $\forall j \in N$, poser $buff_j = 0$
2. Pour $\forall j \in N$, calculer stc_j
3. Trier les activités j en ordre décroissant de leur stc_j
4. Tant qu'il n'y a pas d'amélioration, faire :
    a. Prendre la prochaine activité j de la liste
    b. Si $stc_j = 0$: FIN
    c. Sinon :
        i. $buff_j = buff_j + 1$
        ii. Mise à jour de la date de début de l'activité j et celle de ces successeurs immédiats et transitifs
        iii. Pour $\forall j \in N$, calculer stc_j'
        iv. Si $\sum_{j \in N} stc_j' < \sum_{j \in N} stc_j$ et $s_{n+1} \leq \delta$:
            1. Enregistrer les nouvelles dates de début des activités
            2. Retourner à l'étape 3
        v. Sinon :
            1. $buff_j = buff_j - 1$
            2. Reprendre les dernières dates de début enregistrées

Figure 2.1 : Algorithme STC

Lors de la mise à jour des dates de débuts, à l'étape 3.c.ii, la date de début de l'activité j est augmentée d'une unité de temps. De plus, afin de conserver un ordonnancement admissible, la date de début de tous les successeurs immédiats de l'activité j pour lesquels la marge libre est nulle est augmentée d'une unité de temps. C'est-à-dire que pour tous les successeurs l tels que $s_j' + E(\delta_j) = s_l$, $s_l' = s_l + 1$. Les successeurs de ces successeurs pour lesquels la marge libre est nulle sont également programmés une journée plus tard. Une modification est apportée à cette étape, ce qui permet d'agrandir l'espace des solutions visitées. Si le tampon de temps d'un des successeurs de l'activité j est plus grand que zéro, alors il est diminué d'une unité de temps et la

date de début du successeur ne change pas. La même idée est répétée pour les successeurs transitifs. Un exemple numérique de l'application de la méthode STC se trouve à l'annexe C.

Le réseau de flux de ressources (RFN) est défini sur l'ordonnancement déterministe de durée minimale. Ce réseau $G(N, A \cup A_K)$ contient les arcs représentant les relations de précédence entre les activités ainsi que les arcs représentant le transfert des ressources. Il existe un arc $(i, j) \in A_K$ si une ressource est utilisée par l'activité i puis transférée à l'activité j et que $(i, j) \notin A$. Il faut noter que $A \cap A_k = 0$, car il est inutile d'ajouter une contrainte s'il y en a déjà une. De plus, il peut y exister plusieurs allocations des ressources possibles pour un ordonnancement donné. Klimek et Lebkowski (2011) indiquent que les ressources devraient tout d'abord être transférées d'un prédécesseur à un successeur immédiat, c'est-à-dire entre deux activités $(i, j) \in A$. En effet, cela diminue le nombre de relations de précédence ajoutées au réseau puisque cela évite d'ajouter une contrainte à l'ensemble A_k. Un réseau plus flexible est sujet à une moins grande propagation des perturbations. Ces auteurs présentent la mesure suivante pour qualifier la flexibilité d'un réseau :

$$flex = 1 - \frac{\#(A \cup A_k)}{\frac{N(N-1)}{2}} \tag{2.7}$$

Cette mesure se calcule comme un moins le ratio du nombre de contraintes totales $\#(A \cup A_k)$ du réseau par rapport au nombre théoriquement maximal de contraintes qu'il pourrait y avoir dans le réseau $\frac{N(N-1)}{2}$. Ainsi, *flex* prend une valeur entre zéro et un, où zéro désigne un réseau inflexible et un désigne un réseau qui ne contient aucune contrainte de précédence. Bien que l'objectif ne soit pas de trouver un ordonnancement flexible, mais bien un ordonnancement robuste, cette mesure est la plus appropriée pour comparer différents RFN.

Une des allocations de ressources possibles peut être générée grâce à l'algorithme *Genflow* développée par Artigues et al. (2003). Une petite modification est apportée lors de la génération du réseau afin de favoriser les relations de précédence déjà existantes dans l'ensemble A, suivant la proposition de Klimek et Lebkowski (2011). Lorsqu'une nouvelle activité j est ajoutée au réseau, les premières ressources considérées pour exécuter l'activité j, à l'étape 4.b, sont celles

libérées par les prédécesseurs immédiats de j. L'algorithme original et l'algorithme modifié sont décrits à la figure 2.2, avec les variables suivantes :

0 : activité factice du début

$n+1$: activité factice de la fin

f_{ijk} : flux de ressource de type k transféré de l'activité i à l'activité j

req_k : quantité de ressources de type k encore requise par l'activité j

V_j : ensemble des activités i telles que $s_i + E(\mathbf{d}_j) \leq s_j$, triées en ordre lexicographique

$Pred_j$: ensemble des prédécesseurs de base immédiats i de l'activité j tels que $(i, j) \subset A$

Algorithme *Genflow*

1. Pour $\forall k \in K$, poser $f_{0,n+1,k} = a_k$
2. Pour $\forall k \in K, \forall i, j \in N \mid (i, j) \neq (0, n+1)$, poser $f_{ijk} = 0$
3. Trier les activités $j \in N \setminus \{0, n+1\}$ en ordre non-décroissant de leur date de début s_j
4. Pour chaque activité j de la liste faire :
   a. Pour $\forall k \in K$, poser $req_k = r_{jk}$
   b. Pour $\forall k \in K$ faire :
      i. Pour $\forall i \in V_j$ faire:
         1. $q = \min(req_k, f_{i,n+1,k})$
         2. $req_k = req_k - q$
         3. $f_{i,n+1,k} = f_{i,n+1,k} - q$
         4. $f_{ijk} = f_{ijk} + q$
   c. Pour $\forall k \in K$, poser $f_{j,n+1,k} = r_{jk}$

Algorithme *Genflow Modifié*

5. Pour $\forall k \in K$, poser $f_{0,n+1,k} = a_k$
6. Pour $\forall k \in K, \forall i, j \in N \mid (i, j) \neq (0, n+1)$, poser $f_{ijk} = 0$
1. Trier les activités $j \in N \setminus \{0, n+1\}$ en ordre non-décroissant de leur date de début s_j
2. Pour chaque activité j de la liste faire :
   a. Pour $\forall k \in K$, poser $req_k = r_{jk}$
   b. Pour $\forall k \in K$ faire :
      i. Pour $\forall i \in Pred_j$ faire:
         1. $q = \min(req_k, f_{i,n+1,k})$
         2. $req_k = req_k - q$
         3. $f_{i,n+1,k} = f_{i,n+1,k} - q$
         4. $f_{ijk} = f_{ijk} + q$
      ii. Si $req_k > 0$, pour $\forall i \in V_j$ faire :
         1. $q = \min(req_k, f_{i,n+1,k})$
         2. $req_k = req_k - q$
         3. $f_{i,n+1,k} = f_{i,n+1,k} - q$
         4. $f_{ijk} = f_{ijk} + q$
   c. Pour $\forall k \in K$, poser $f_{j,n+1,k} = r_{jk}$

Figure 2.2 : Algorithmes *Genflow* et *Genflow Modifié*

En deuxième lieu, les priorités des activités Π sont déterminées. La règle de priorité EPST1 (*earliest projected starting times*) est utilisée. Van de Vonder et al. (2007a) ont montré que c'est celle qui donne les solutions les plus robustes. Elle est aussi parmi les plus rapides. Selon cette règle, les activités sont ordonnées en ordre croissant de leur date de début prévue s_i. S'il y a égalité, les activités sont ordonnées en ordre décroissant de leur poids. Un exemple numérique de l'application des algorithmes *Genflow* et *Genflow Modifié* se trouve en annexe C.

2.2.2 Amélioration de la solution

Van de Vonder et al. (2008) montrent qu'une phase d'amélioration où la simulation est utilisée pour évaluer la valeur de la solution courante permet d'améliorer la solution obtenue par la méthode STC. De plus, la méthode SBD (*Simulation-based Descent*) introduite par Deblaere et al. (2011a) est une méthode basée sur la simulation qui supplante la méthode STC. Cependant, les temps de calcul de ces méthodes sont substantiellement plus grands dû à l'utilisation de la simulation. Dans le cadre de ce mémoire, les algorithmes ont été codés en VBA, un langage nécessitant des temps de calcul substantiellement plus grands que le langage C++ utilisé dans la littérature. Ainsi, les temps de calcul nécessaire pour effectuer la simulation sont jugés trop grands pour que celle-ci soit utilisée durant l'algorithme. À titre d'exemple, simuler 1000 répétitions de l'exécution d'un projet de 120 activités sur l'ordinateur utilisé nécessite environ 20 secondes de calcul. Ceci est trop long pour pouvoir évaluer un ordonnancement à de multiples reprises durant l'algorithme. En effet, pour une instance de 120 activités, évaluer les 240 voisins du voisinage restreint des dates de début des activités nécessiterait 1,3 heure de calcul. Le voisinage restreint est composé des ordonnancements où la date de début d'une activité est avancée ou reculée d'une période.

En conclusion, la simulation n'est pas utilisée pour évaluer la robustesse d'une solution intermédiaire. Il n'y a pas de phase d'amélioration dans la méthode proposée. Bien que l'abandon de cette phase d'amélioration nuise à la qualité des solutions trouvées, cela permet d'obtenir des solutions dans un temps de calcul beaucoup plus petit. Il faut rappeler que ce mémoire s'intéresse aux projets d'envergure, qui comportent un très grand nombre d'activités. L'ordonnancement final est donc celui construit par la méthode proposée, c'est-à-dire l'heuristique STC où la construction du RFN et la mise à jour de l'ordonnancement ont été modifiées. Seule la robustesse

d’une solution finale est évaluée par simulation afin de juger de la performance de la méthode proposée. Le chapitre suivant présente les résultats numériques obtenus pour cette méthode.

CHAPITRE 3 RÉSULTATS NUMÉRIQUES

Ce chapitre vise à évaluer la performance du modèle présenté précédemment en présentant les résultats numériques obtenus suite à une série de tests. Ainsi, les instances utilisées pour les tests sont d'abord présentées. Ensuite, les mesures de robustesse sont analysées. Par la suite, la performance de la méthode proposée est évaluée. Finalement, l'impact de certains paramètres est examiné.

3.1 Instances tests

Les tests sont effectués sur des instances tirées de PSPLIB (Kolisch et Sprecher 1997). Ce choix est motivé par le fait que ces instances sont couramment utilisées dans la littérature. Comme ce mémoire s'intéresse aux projets de grande envergure, les instances utilisées sont celles comportant le plus grand nombre d'activités, c'est-à-dire les instances de 120 activités. Dans la pratique, les projets d'envergure comportent beaucoup plus d'activités. Ces instances de 120 activités sont quand même assez importantes, particulièrement si les phases du projet sont ordonnancées individuellement. Afin d'obtenir des résultats comparables, les paramètres choisis sont les mêmes que ceux utilisés dans Deblaere et al. (2011a) et Van de Vonder et al. (2008).

Dix instances de 120 activités sont choisies. Les instances de la base de données de PSPLIB ont été générées de façon aléatoire à partir de trois paramètres. Le premier est le nombre moyen de relation de précédence par activité dans le réseau, NC. Il y a trois valeurs possibles : 1.5, 1.8 et 2.1. Le réseau contient peu de connexions pour 1.5 alors qu'il y en a beaucoup pour 2.1. Le deuxième est le nombre moyen de types de ressources différents requis par les activités, RF_R. Il y a quatre valeurs possibles : 0.25, 0.5, 0.75 et 1. Plus le nombre est grand, plus les activités requièrent des ressources de type différent. Le dernier désigne la sévérité des contraintes de ressources, RS_R. Il y a 5 valeurs possibles, de 0.1 à 0.5. Les contraintes sont très sévères pour 0.1 alors qu'elles le sont moins pour 0.5. Le tableau 3.1 montre les paramètres utilisés pour générer les dix instances utilisées.

Tableau 3.1 : Paramètres de génération des instances déterministes

	NC	RF_R	RS_R
120054	1.5	0.25	0.5
120101	1.5	0.50	0.5
120158	1.5	0.75	0.5
120301	1.8	0.50	0.5
120303	1.8	0.50	0.5
120025	1.5	0.25	0.2
120061	1.5	0.50	0.1
120011	1.5	0.50	0.5
1202110	1.8	0.25	0.1
1200110	1.5	0.50	0.5

Un ordonnancement initial déterministe est trouvé en prenant la durée moyenne $E(\mathbf{d}_i)$ des activités comme durée. L'objectif est alors de minimiser la durée totale du projet $s_{\max}$. Les instances sont résolues à l'aide de la méthode présentée par Bieber (2012). La solution optimale est trouvée pour les cinq premières instances. Les cinq dernières instances semblent plus difficiles à résoudre à cause des contraintes de ressources qui sont plus sévères. Ainsi, la moitié des instances ont une solution initiale de durée minimale, alors que les autres commencent avec la meilleure solution trouvée.

Dans le cas stochastique, la durée d'une activité $\mathbf{d}_i$ est une variable aléatoire. Les durées peuvent suivre des fonctions statistiques diverses dans la pratique. La loi de probabilité Bêta est toutefois couramment utilisée en gestion de projet pour modéliser la durée d'une activité. Cette loi convient bien car elle nécessite peu de points d'estimation. La loi Bêta discrétisée de paramètre $\alpha = 2$ et $\beta = 5$ est utilisée, suivant les hypothèses établies dans la littérature. La courbe asymétrique représente le fait qu'une activité dure souvent plus longtemps que prévu. En effet,

l'accroissement possible de la durée d'une activité par rapport à la valeur espérée est plus grand que la réduction possible. La durée des activités factices est nulle avec une variance également nulle. Les autres activités ont une variance caractérisée comme petite, moyenne ou grande. La loi Bêta nécessite de définir les valeurs extrêmes de la variable. Les durées minimales et maximales d'une activité dépendent de sa variance, comme le montre le tableau 3.1 ci-dessous :

Tableau 3.2**:** Durée minimale et maximale des activités selon la variance

Variance	Durée minimale	Durée maximale
Petite	$0{,}75 \cdot E(\mathbf{d}_i)$	$1{,}625 \cdot E(\mathbf{d}_i)$
Moyenne	$0{,}5 \cdot E(\mathbf{d}_i)$	$2{,}25 \cdot E(\mathbf{d}_i)$
Grande	$0{,}25 \cdot E(\mathbf{d}_i)$	$2{,}875 \cdot E(\mathbf{d}_i)$

Un poids est également associé à chaque activité. Toutes les activités, sauf les activités factices, ont 50% de chance d'être souples. C'est-à-dire d'avoir un poids nul car il n'y a pas de coût engendré par un décalage dans la date de début de l'activité. Pour les autres activités non factices, le poids suit une loi triangulaire discrète où $P(w_i = q) = (21 - 2q)\%$ avec $q \in \{1,2,...,10\}$. Ces poids représentent les coûts engendrés si l'activité est commencée avant ou après sa date de début prévue. L'activité $n+1$ a un poids $w_{n+1} = \lfloor 10 \cdot \overline{w} \rfloor = 38$. Ce poids représente l'importance de ne pas dépasser la date de fin prévue du projet. L'activité 0 a un poids nul et commence toujours au temps $t = 0$. Pour chaque instance déterministe, trois ensembles de poids et de variances différents sont générés. Cela permet de construire trente instances différentes pour le cas stochastique.

L'ordonnancement est rendu robuste en insérant des tampons de temps. Le tampon de temps total concédé, nommé b, représente un certain pourcentage de l'espérance de la durée du projet $E(\mathbf{s}_{n+1})$ lorsque les dates de début prévues des activités sont celles de l'ordonnancement initial déterministe. Ce tampon de temps est d'abord ajouté à la fin du projet. Ainsi, la date de fin prévue du projet devient $\delta = ent[(1+b) \cdot E(\mathbf{s}_{n+1})]$. Dans la pratique, b pourrait être déterminé selon la variabilité de la date de fin simulée du projet en exécutant le projet selon

l'ordonnancement initial. En effet, la simulation permet d'obtenir la loi de distribution empirique de la date de fin d'un projet pour un ordonnancement donné. Bien que la distribution de l'ordonnancement final ne soit pas la même que celle de l'ordonnancement initial, une analyse de la variabilité de la date de fin selon l'ordonnancement initial amènerait des informations supplémentaires pour choisir le tampon de temps total b à ajouter au projet. Pour obtenir des résultats comparables, les tests sont effectués en posant $b = 0{,}01$ pour chaque instance. Ainsi, le tampon de temps total est 1% de l'espérance de la durée du projet. Cela représente environ 5% de la durée minimale du projet dans le cas déterministe.

Les algorithmes sont codés en VBA. Le code se trouve à l'annexe B. Ce langage permet d'exécuter le programme avec Excel®, un logiciel présent dans la majorité des entreprises. Les tests ont été effectués sur un ordinateur personnel (AMD Athlon 64 X2 Dual Core Processor 4200+, 2,22 GHz, 3 Go de RAM).

3.2 Évaluation de la robustesse

Pour évaluer la robustesse d'un ordonnancement, il faut trouver la date de début réelle de chaque activité $\mathbf{s}_i$. La durée réelle d'une activité $\mathbf{d}_i$ est définie comme une variable aléatoire dont la distribution est connue. L'idée est de simuler plusieurs réalisations du projet. La date de début réalisée de chaque activité $\mathbf{s}_i$ dépend alors de la politique d'exécution choisie lors de la réalisation du projet. Pour chaque répétition du projet simulé, il est possible de calculer l'écart entre la date de début prévue et la date de début réalisée d'une activité $|\mathbf{s}_i - s_i|$. La somme de ces écarts pour toutes les activités donne le coût simulé de l'ordonnancement, soit la mesure de robustesse. Le coût simulé moyen converge lorsque le nombre de répétitions est élevé. Il suffit donc de déterminer la politique d'exécution du projet ainsi que le nombre de simulations qu'il est nécessaire d'effectuer. Il faut aussi s'assurer que la somme des STC_i offre une bonne approximation de la valeur obtenue par simulation.

3.2.1 Politique d'exécution

Le projet est réalisé suivant une politique d'exécution. Dans ce mémoire, la politique d'exécution choisie est celle utilisée par Deblaere et al. (2011a). C'est une politique basée sur les ressources

(*resource-based policy*). Le projet est exécuté en suivant le principe de l'algorithme parallèle. Cet algorithme est utilisé comme règle d'ordonnancement dynamique.

Il y a un point de décision au temps $t = 0$ et à chaque période t où une activité termine. Une activité i est éligible à commencer au temps t si tous ces prédécesseurs sont terminés et si sa date de début prévue est atteinte, c'est-à-dire que $s_i \leq t$. Lors d'un point de décision, les activités éligibles sont considérées selon l'ordre de priorité. L'activité considérée débute si la quantité de ressources disponibles est suffisante. Cette politique ne permet pas qu'une activité débute avant sa date de début prévue, donc les coûts engendrés par une activité commençant plus tôt n'ont pas à être pris en compte.

3.2.2 Nombre de simulations nécessaire

Le nombre de simulations nécessaire pour obtenir un résultat précis est déterminé selon les résultats numériques obtenus. Les solutions utilisées pour effectuer les tests sont obtenues en appliquant la méthode proposée. Ainsi, pour chacune des trente instances, un RFN est trouvé selon l'algorithme *Genflow* modifié, puis la solution est rendue robuste en appliquant l'algorithme STC modifié.

L'instance 120054 est utilisée comme exemple dans le graphique ci-dessous. L'ensemble des poids et des variances associé aux activités se trouve en annexe A. La robustesse de la solution est évaluée en effectuant de 1 à 15 000 répétitions de l'exécution du projet. Une courbe représente la valeur moyenne du coût simulé obtenu pour chaque répétition. Le graphique de la figure 3.1 montre les courbes de 30 essais pour lesquels 15 000 répétitions ont été effectuées.

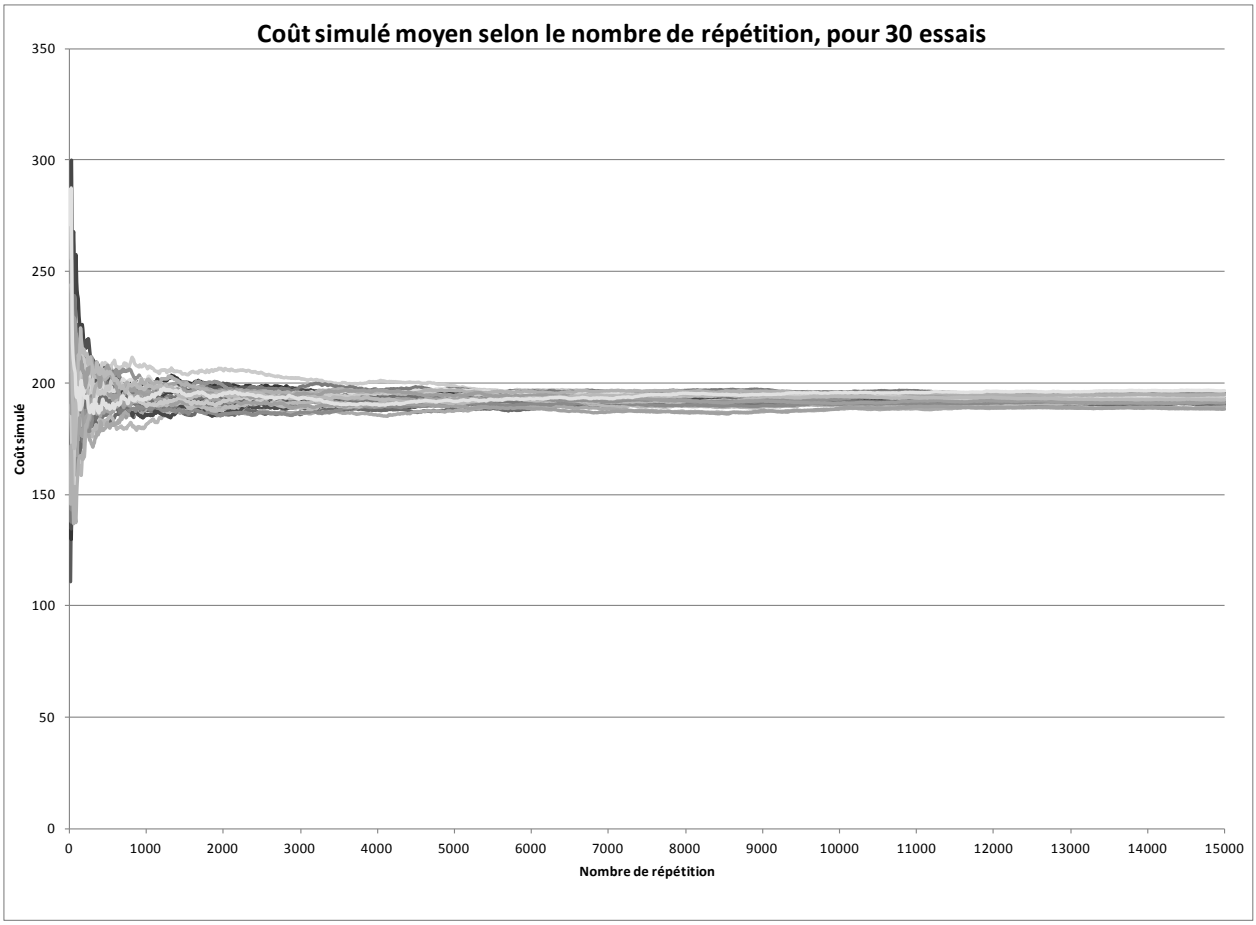

Figure 3.1**:** Coût simulé moyen selon le nombre de répétitions, pour 30 essais

Pour chacune des instances, le coefficient de variation du coût simulé moyen des 30 essais selon différents nombres de répétitions est calculé. Le tableau 3.2 montre la moyenne des coefficients de variation des 30 instances selon le nombre de répétitions.

Tableau 3.3 : Coefficient de variation selon le nombre de répétitions

Nombre de répétitions	Coefficient de variation
1 000	2,9%
5 000	1,3%
10 000	0,9%
15 000	0,8%

La valeur de robustesse obtenue par la simulation n'est pas précise. En effet, il y a beaucoup d'aléatoire introduit par la variation de la durée des activités. Deblaere et al. (2011a) jugent que mille répétitions sont suffisantes pour évaluer la robustesse d'une instance de 120 activités. Le coefficient de variation moyen est tout de même de 3% pour mille répétitions. Dans ce mémoire, la simulation n'est utilisée que pour évaluer une solution finale. Puisque le temps de calcul nécessaire n'est pas un problème dans ce cas, une solution est évaluée en effectuant dix mille répétitions de façon à obtenir un résultat plus précis. Le coefficient de variation moyen est alors de moins de 1%.

3.2.3 Mesure de robustesse approchée

C'est la simulation qui permet de mesurer la robustesse de la façon la plus exacte. Cependant, elle requiert un long temps de calcul. Les algorithmes permettant de rendre un ordonnancement robuste nécessitent d'évaluer la robustesse de l'ordonnancement à de nombreuses reprises. Dans le cas d'un projet d'envergure, les temps de calcul deviennent trop grands. C'est pourquoi la méthode proposée utilise une mesure approchée nécessitant moins de temps de calcul. L'heuristique STC mesure la robustesse par la somme des STC_i de chaque activité. La corrélation entre cette somme et le coût obtenu par simulation est étudiée afin de s'assurer que cette mesure offre une bonne approximation de la robustesse. Les valeurs de robustesse des solutions utilisées à la section 3.3.2 sont étudiées. Dans cette section, les versions originales et modifiées des méthodes *Genflow* et STC sont testées en comparant les quatre possibilités pour trente instances, ce qui donne cent-vingt solutions différentes. Dans la section actuelle, la valeur de la robustesse d'une solution évaluée par la somme des STC_i est comparée à la valeur obtenue par le coût simulé. Le graphique de la figure 3.2 montre le coût simulé en fonction de la somme des STC_i pour chacune de ces cent-vingt solutions.

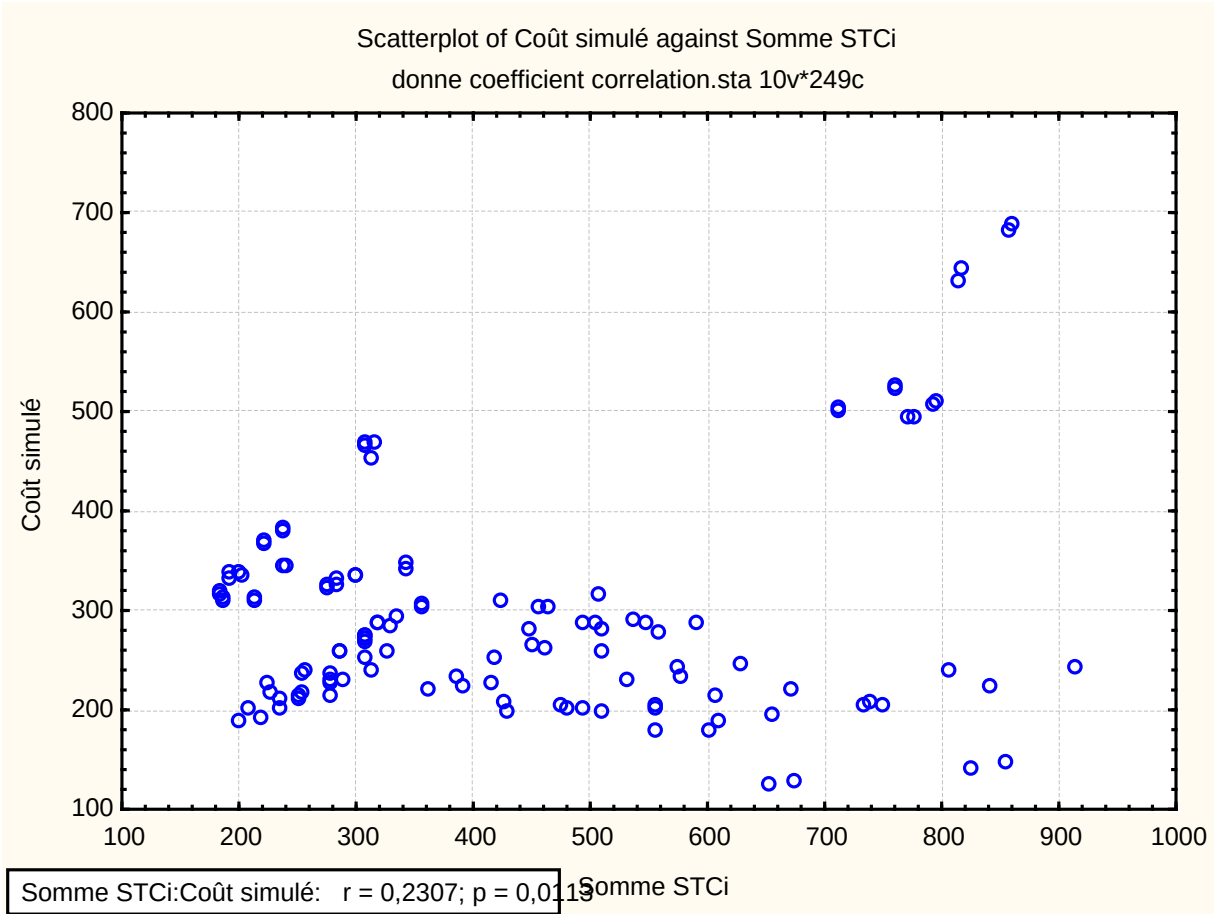

Figure 3.2 : Coût simulé selon la somme des STC_i, pour les 120 solutions.

Le coefficient de corrélation entre les deux mesures est 0,23 ($P < 0{,}05$). Il existe donc une faible corrélation entre les deux valeurs de robustesse. Cela implique un certain décalage entre les deux mesures. Une diminution de la somme des STC n'entraine pas obligatoirement une diminution du coût simulé.

3.3 Performance de la méthode proposée

Le gain obtenu grâce aux deux modifications proposées est étudié. D'abord, la flexibilité des RFN obtenus par les méthodes *Genflow* originale et modifiée est examinée. Ensuite, la robustesse des solutions obtenues est comparée. Finalement, les temps de calcul des différentes méthodes sont examinés.

3.3.1 Flexibilité du RFN

Un RFN est trouvé pour chacune des dix instances déterministes selon les méthodes *Genflow* originale et modifiée. Le RFN est défini sur l'ordonnancement déterministe, il est donc indépendant de l'ensemble des poids et des variances ou de la date de fin δ. La modification apportée à la méthode *Genflow* vise à réduire le nombre de prédécesseurs totaux du RFN en favorisant les prédécesseurs déjà existants lors de la construction du flux de ressource. La mesure *flex* est utilisée pour qualifier la flexibilité d'un réseau selon le nombre de prédécesseurs totaux du RFN. Le tableau 3.3 montre les résultats obtenus. Les différences sont reportées en soustrayant la valeur obtenue par la méthode originale à la valeur obtenue pour la méthode modifiée.

Tableau 3.4 : Comparaison du RFN obtenu par les méthodes *Genflow* originale et modifiée

	Nombre de prédécesseurs		**Flexibilité**		
Instances :	**Différence** $\#A_k$	**Différence** $\#A \cup A_k$	***flex* pour *Genflow* originale**	***flex* pour *Genflow* modifiée**	**Différence** *flex*
120054	-2	-15	0,9489	0,9510	0,20%
120101	5	-23	0,9312	0,9343	0,31%
120158	-39	-73	0,9188	0,9287	0,99%
120301	0	-26	0,9263	0,9298	0,35%
120303	-15	-40	0,9271	0,9325	0,54%
120025	2	-5	0,9512	0,9519	0,07%
120061	2	-5	0,9312	0,9319	0,07%
120011	2	-1	0,9499	0,9500	0,01%
1202110	-4	-6	0,9443	0,9451	0,08%
1200110	2	-5	0,9495	0,9501	0,07%

Les résultats montrent que même si l'ensemble des contraintes de précédence pour les ressources A_k est parfois plus grand avec la méthode modifiée, le nombre de prédécesseurs totaux est toujours strictement plus petit. La valeur de *flex* est toujours plus grande grâce à la modification. Cela signifie que le RFN est plus flexible car il comporte moins de contraintes de précédence. Cependant, pour les cinq dernières instances, l'amélioration est minime. Ces dernières sont des instances pour lesquelles la solution optimale n'a pas été trouvée. Ce phénomène pourrait être expliqué par le fait que les contraintes de ressource sont plus sévères et qu'il n'est pas possible de trouver un réseau de flux de ressources sans ajouter plusieurs autres relations de précédence. Le tableau 3.4 résume l'avantage de la modification apportée.

Tableau 3.5 : Réduction de la valeur de *flex* de la méthode *Genflow* modifiée par rapport à la version originale

	Moyenne	Écart-type	Intervalle de confiance à 95%
Instances 1 à 15 (solutions optimales)	0,48%	0,31%	0,27%
Instances 16 à 30 (solutions non-optimales)	0,06%	0,03%	0,02%
Toutes les instances	0,27%	0,30%	0,19%

La méthode *Genflow* modifiée permet de trouver un RFN significativement plus flexible. L'amélioration est notable pour les instances où la solution pour le cas déterministe est optimale.

3.3.2 Robustesse de l'ordonnancement

Les versions originales et modifiées pour les méthodes *Genflow* et STC sont comparées pour les 30 instances, avec $b = 0{,}01$. Le graphique de la figure 3.3 montre la somme des STC_i selon les quatre combinaisons de méthode.

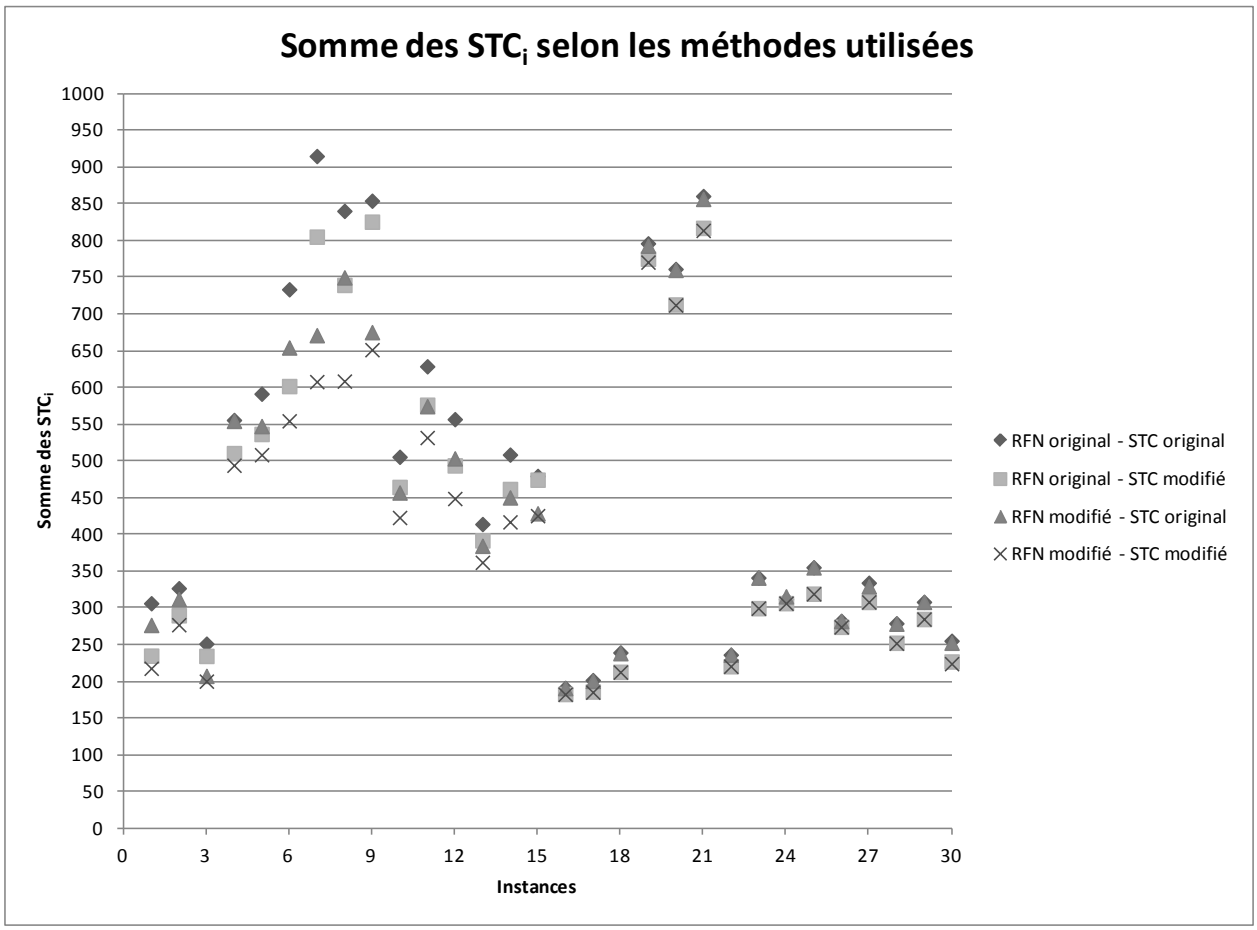

Figure 3.3 : Somme des STC_i selon les méthodes utilisées

Les modifications apportées amènent une amélioration dans tous les cas. Pour les instances 16 à 30, la version de la méthode *Genflow* utilisée pour trouver le RFN influence faiblement la valeur de robustesse obtenue. Il a été montré à la section précédente que la méthode *Genflow* modifiée ne trouve pas un RFN beaucoup plus flexible pour les instances non-optimales. Cependant, pour les instances 1 à 15, la méthode *Genflow* modifiée trouve des solutions plus robustes que la version originale. La modification apportée à la méthode STC donne des solutions plus robustes dans tous les cas.

Le graphique de la figure 3.4 montre le coût simulé selon les quatre combinaisons de méthode.

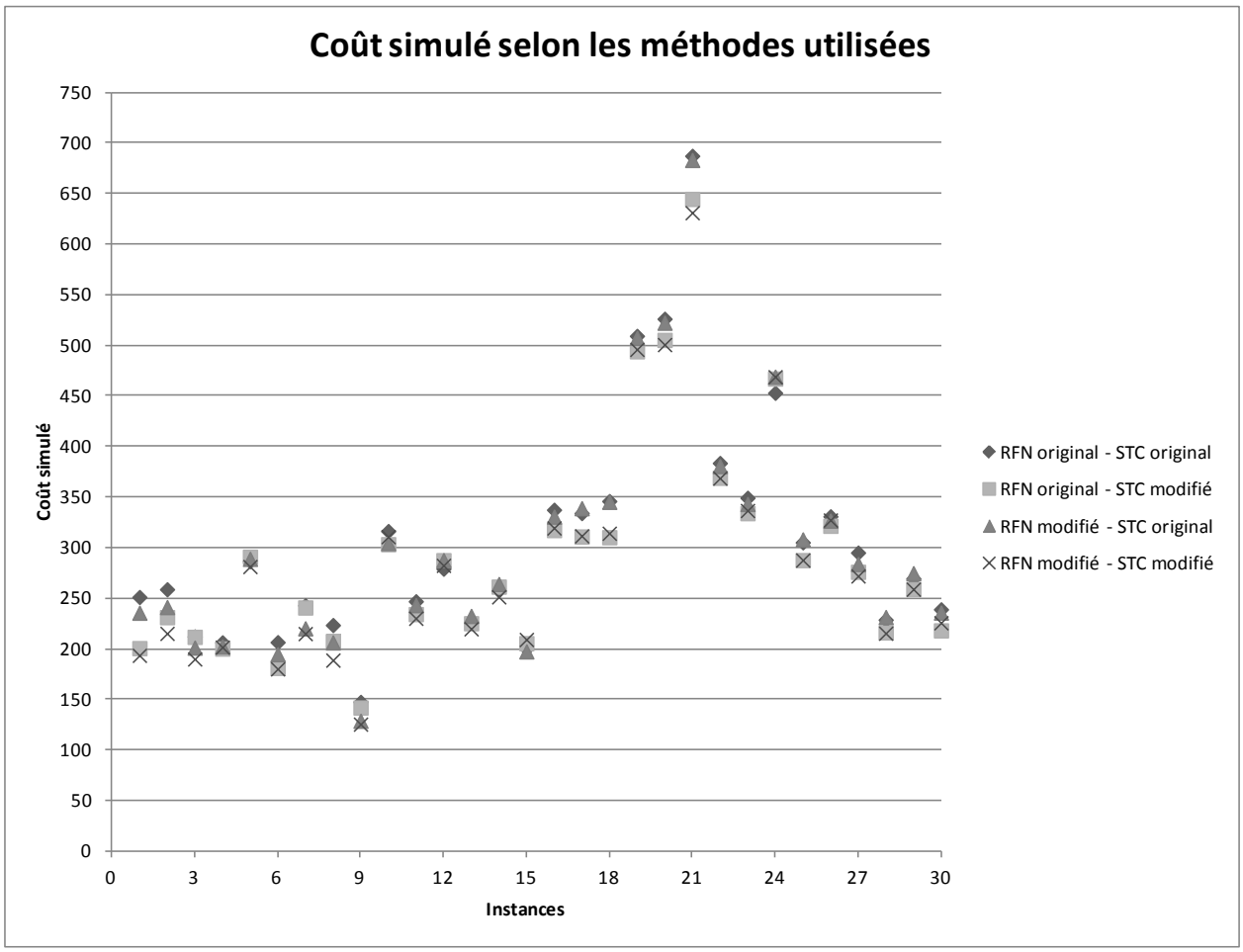

Figure 3.4 : Coût simulé selon les méthodes utilisées

Le coût simulé ne montre pas une amélioration pour tous les cas. Cela est dû au manque de corrélation entre la valeur de robustesse obtenue par la somme des STC_i et par le coût simulé.

Le tableau 3.5 montre le pourcentage d'amélioration obtenu par les méthodes modifiées par rapport aux méthodes *Genflow* et STC originales.

Tableau 3.6 : Coût moyen obtenu par les méthodes modifiées
par rapport à celui obtenu par la méthode originale

	Somme des STC_i			Coût simulé		
Méthodes utilisées :	**RFN modifié**	**STC modifié**	**RFN et STC modifiés**	**RFN modifié**	**STC modifié**	**RFN et STC modifiés**
Instances 1 à 15 (solutions optimales)	-11,0%	-9,8%	-19,4%	-3,6%	-4,0%	-7,8%
Instances 16 à 30 (solutions non-optimales)	-0,2%	-7,1%	-7,3%	-0,3%	-5,0%	-4,7%
Toutes les instances	-5,6%	-8,5%	-13,4%	-1,9%	-4,5%	-6,3%

Les résultats montrent que la méthode *Genflow* modifiée n'amène pas une grande amélioration par rapport à la méthode originale pour les instances non-optimales. En effet, la section 3.3.1 montre que les RFN obtenus dans ces cas ne sont pas très différents. Le tableau 3.6 résume l'amélioration obtenue par les méthodes modifiées.

Tableau 3.7 : Amélioration des méthodes modifiées par rapport aux méthodes originales

	Moyenne	Écart-type	Intervalle de confiance à 95%
Somme des STC_i	-13,4%	8,1%	2,9%
Coût simulé	-6,3%	6,0%	2,2%

Les méthodes modifiées amènent une amélioration significative. Même si l'amélioration du coût simulé est moins grande, l'amélioration est significative.

3.3.3 Temps de calcul

Les temps de calcul pour les méthodes *Genflow* originale et modifiée sont de l'ordre de la seconde. Ils sont donc négligeables. Les temps de calcul pour les deux versions de la méthode STC sont présentés dans le tableau 3.7.

Tableau 3.8 : Temps de calcul (secondes) pour les méthodes STC originale et modifiée

Méthode STC :	Moyenne	Écart-type	Intervalle de confiance à 95%
Originale	43 s	37 s	9 s
Modifiée	151 s	58 s	15 s

Pour la version originale, le temps de calcul est de l'ordre de quarante secondes. La modification apportée engendre des temps de calcul plus grands. Le temps de calcul est de l'ordre de deux minutes et demie. Cette augmentation du temps de calcul est due au fait qu'un voisinage plus large est visité. Dans la version originale, le tampon de temps d'une activité ne peut que s'accroître. Dans la version modifiée, il peut aussi être réduit.

3.4 Variation des paramètres

L'impact de certains paramètres est étudié. D'abord, l'amélioration de la robustesse lorsqu'aucun tampon de temps total n'est ajouté est étudiée. Ensuite, la robustesse obtenue en partant d'une solution initiale différente est examinée.

3.4.1 Tampon de temps total nul

Il est intéressant d'étudier l'utilité de la méthode si la date de fin du projet n'est pas repoussée. Dans ce cas-ci, le tampon de temps total b ajouté est nul et la date de fin du projet est $\delta = s_{\max}$. Trois ordonnancements sont comparés. D'abord, l'ordonnancement initial qui est composé des dates de début de la solution déterministe de durée minimale. Ensuite, l'ordonnancement rendu robuste où la date de fin n'a pas été reculée. De plus, à titre référentiel, l'ordonnancement rendu robuste en appliquant la méthode proposée où la date de fin a été reculée de 1% de l'espérance de

la durée du projet, c'est-à-dire les résultats obtenus à la section 3.3. Le graphique de la figure 3.5 montre la somme des STC_i pour ces trois ordonnancements.

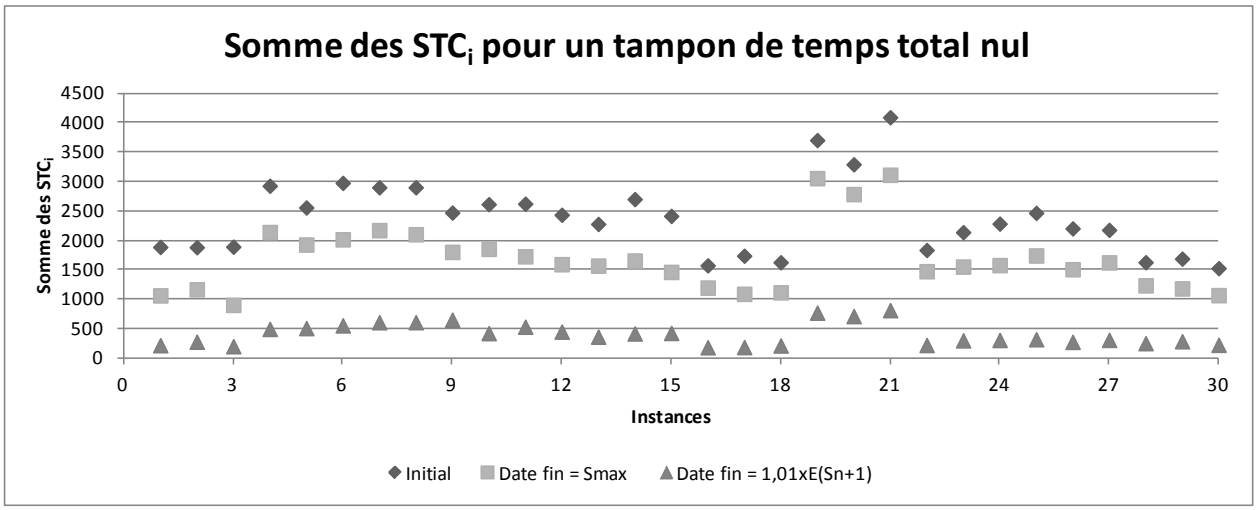

Figure 3.5 : Somme des STC_i pour un tampon de temps total nul

Le graphique de la figure 3.6 montre le coût simulé pour ces trois ordonnancements.

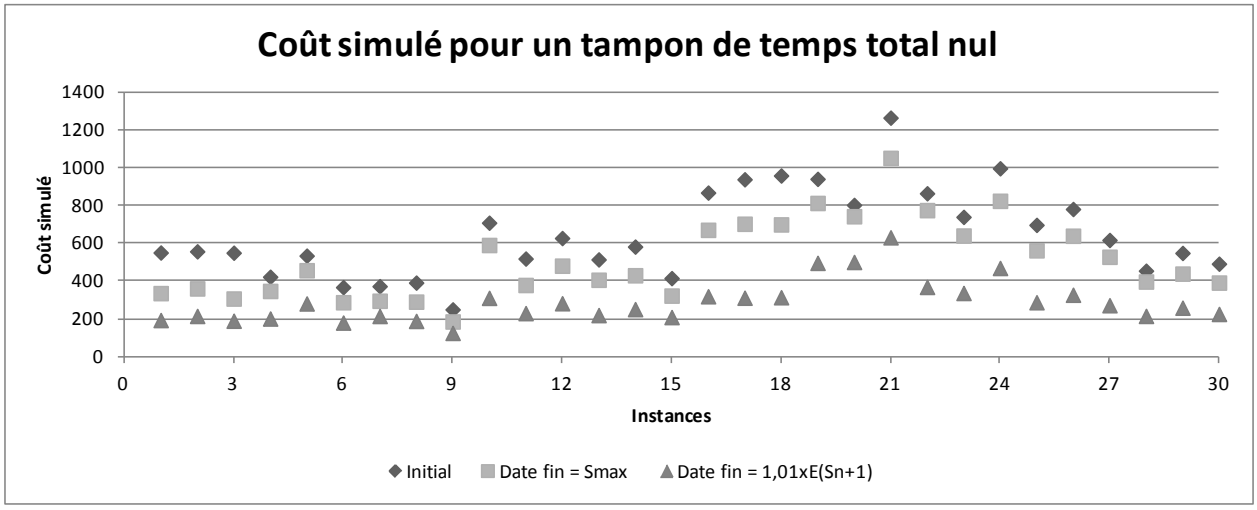

Figure 3.6 : Coût simulé pour un tampon de temps total nul

La méthode permet de rendre chacune des instances plus robustes que la solution initiale, que la robustesse soit calculée par la somme des STC_i ou par le coût simulé. Cela même si la date de fin n'est pas reculée. Évidemment, les ordonnancements sont encore plus robustes lorsque la date de fin est reculée. Le tableau 3.8 montre la diminution de la somme des STC_i et du coût simulé de l'ordonnancement où la date de fin n'est pas reculée par rapport à l'ordonnancement initial.

Tableau 3.9 : Diminution des coûts pour un tampon de temps total nul

Diminution du coût :	Moyenne	Écart-type	Intervalle de confiance à 95%
Somme des STC_i	-30%	8%	3%
Coût simulé	-21%	8%	3%

En appliquant l'heuristique tout en n'ajoutant aucun tampon de temps total, la robustesse augmente significativement de 30% selon la somme des STC_i et de 20 % selon le coût simulé. Ces résultats montrent que la robustesse d'un ordonnancement peut être grandement améliorée tout en conservant une durée totale de projet minimale. Ceci peut être expliqué par le fait que le problème déterministe est résolu en optimisant seulement la durée totale du projet. L'heuristique STC permet par la suite d'optimiser la robustesse de l'ordonnancement en déplaçant les activités dans leur marge totale.

3.4.2 Solution initiale de plus courte durée

L'importance de la solution initiale est étudiée. Cela permet de juger de l'intérêt à accorder aux méthodes visant la réduction de la durée totale du projet. Pour les cinq dernières instances déterministes dont la solution initiale était non-optimale, une deuxième solution déterministe de plus courte durée est trouvée grâce à la méthode proposée par Berthaut (2013). Le premier ordonnancement initial est nommé Ordo1 et le deuxième, d'une plus courte durée, est nommé Ordo2. Le tableau 3.9 montre, pour le cas déterministe, la date de fin des deux ordonnancements trouvés ainsi que la date de fin optimale ou sinon un intervalle dans lequel elle se trouve si elle n'est pas connue. Le tableau présente aussi, pour le cas stochastique, la date de fin lorsque celle-ci est retardée de 1% de l'espérance de la durée du projet. Les résultats sont regroupés selon les dix instances déterministes. L'espérance de fin varie selon les ensembles de poids et de variances choisis.

Tableau 3.10 : Date de fin des ordonnancements

	Ordo1		Ordo2		Solution optimale
Instances :	$s_{\max Ordo1}$	$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo1})$	$s_{\max Ordo2}$	$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo2})$	$s_{\max Optmiale}$
120025 (16 à 18)	114	122	103	114-116	103
120061 (19 à 21)	191	197-198	151	167-168	132-144
120011 (22 à 24)	119	126-128	106	115-117	104-105
1202110 (25 à 27)	111	117-119	103	112-113	102
1200110 (28 à 30)	123	129	108	117-118	108

Les ordonnancements déterministes obtenus par la méthode de Berthaut (2013) sont visiblement d'une meilleure qualité que ceux obtenus par la méthode de Bieber (2012). La date de fin des Ordo2 pour le cas déterministe est très près de l'optimal. Leur date de fin retardée de 1% est environ la même que la date de fin des Ordo1 dans le cas déterministe.

Pour chacune des quinze instances, trois cas sont comparés. D'abord, le premier ordonnancement avec une date de fin $\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo1})$, c'est-à-dire l'ordonnancement obtenu dans la section 3.3 avec la méthode modifiée. Ensuite, le deuxième ordonnancement avec une date de fin $\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo2})$ et puis avec une date de fin $\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo1})$, c'est-à-dire la même date de fin que pour le premier ordonnancement.

Le tableau 3.10 montre la flexibilité des RFN obtenus. Les différences sont reportées en soustrayant la valeur obtenue pour le deuxième ordonnancement (Ordo2) à la valeur obtenue pour le premier ordonnancement (Ordo1). Il faut noter que le RFN est indépendant de la date de fin δ puisqu'il est défini sur la solution déterministe.

Tableau 3.11 : Comparaison du RFN obtenue pour les deux ordonnancements

	Nombre de prédécesseurs		Flexibilité		
Instances :	**Différence** $\#A_k$	**Différence** $\#A \cup A_k$	***flex* pour Ordo1**	***flex* pour Ordo2**	**Différence** *flex*
120025	-1	-2	0,9519	0,9522	0,03%
120061	-29	-22	0,9319	0,9348	0,30%
120011	-3	-3	0,9500	0,9504	0,04%
1202110	-4	-10	0,9451	0,9465	0,14%
1200110	-6	0	0,9501	0,9501	0,00%

Les résultats montrent que le nombre de contraintes de précédence ajoutées est inférieur pour le deuxième ordonnancement, c'est-à-dire avec une solution initiale d'une plus courte durée. Ceci pourrait être expliqué par le fait que les activités ont une moins grande marge. En étant plus proches, il serait plus facile de transférer les ressources en suivant les contraintes de précédence déjà existantes. Ce phénomène pourrait aussi être dû à la méthode utilisée pour trouver l'ordonnancement déterministe. Il se peut que la méthode proposée par Berthaut (2013) trouve des ordonnancements intrinsèquement plus flexibles, c'est-à-dire des ordonnancements où les ressources peuvent être plus facilement transférées d'un prédécesseur à un successeur appartenant à l'ensemble A. Le tableau 3.11 résume l'amélioration de la flexibilité pour un ordonnancement initial plus court.

Tableau 3.12 : Réduction de la valeur de *flex* de l'Ordo2 par rapport à l'Ordo1

	Moyenne	**Écart-type**	**Intervalle de confiance à 95%**
Cinq dernières instances	0,10%	0,12%	0,11%

L'amélioration moyenne de la flexibilité est de 0,10%. Cependant, selon l'instance, la réduction varie de 0 à 0,30%. Il s'agit tout de même d'une amélioration puisque la flexibilité ne se dégrade dans aucun cas. Le graphique de la figure 3.7 suivant montre la somme des STC_i selon les trois combinaisons d'ordonnancement initial.

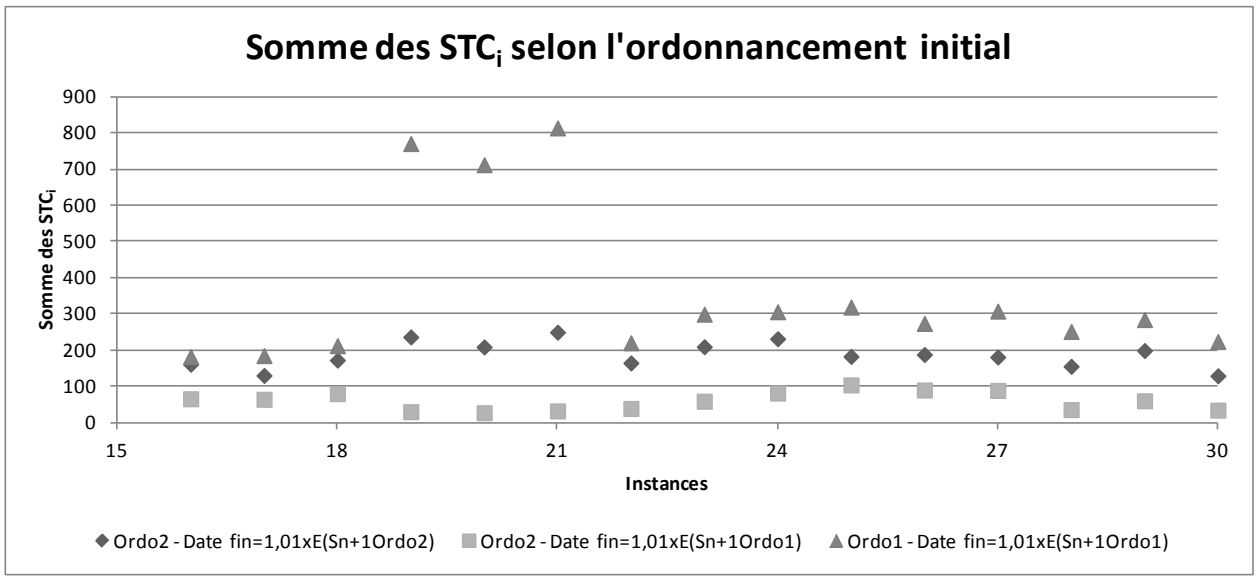

Figure 3.7 : Somme des STC_i selon l'ordonnancement initial

Un ordonnancement initial plus près de l'optimal montre une amélioration de la robustesse dans tous les cas, lorsque celle-ci est évaluée par la somme des STC_i. Le deuxième ordonnancement est particulièrement plus robuste que le premier lorsque ceux-ci ont la même date de fin planifiée. Le graphique de la figure 3.8 montre le coût simulé selon les trois combinaisons d'ordonnancement initial.

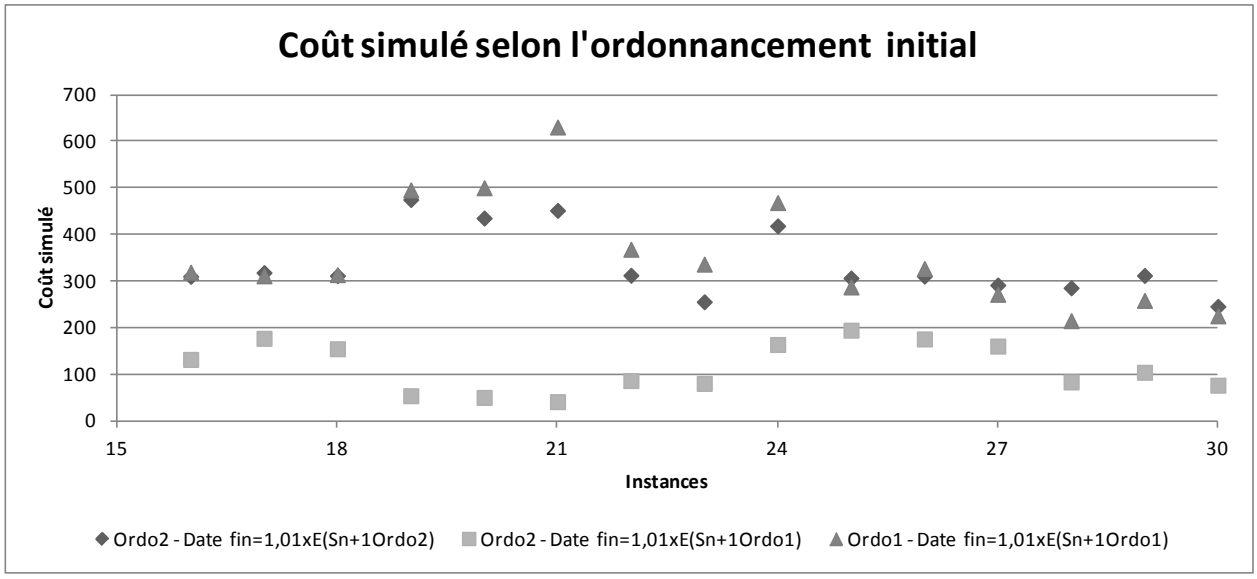

Figure 3.8 : Coût simulé selon l'ordonnancement initial

Le coût simulé du deuxième ordonnancement avec la date de fin reculée du premier est toujours plus bas. Pour les deux autres cas, où la date de fin est 1% de plus que l'espérance de la durée totale des ordonnancements respectifs, le coût simulé n'est pas toujours meilleur. Cela pourrait être dû au manque de corrélation entre la valeur de robustesse obtenue par la somme des STC_i et par le coût simulé.

Le tableau 3.12 montre le pourcentage d'amélioration de la somme des STC_i obtenue par le deuxième ordonnancement par rapport au premier, pour les deux dates de fin étudiées.

Tableau 3.13 : Amélioration moyenne de la somme des STC_i de l'Ordo2 par rapport à l'Ordo1 pour les deux dates de fin étudiées

	Moyenne	Écart-type	Intervalle de confiance à 95%
$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo2})$	-38%	18%	9%
$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo1})$	-78%	12%	6%

Les résultats montrent qu'un ordonnancement initial de plus courte durée donne toujours une solution plus robuste selon la somme des STC_i. L'amélioration serait en partie due au meilleur RFN trouvé. Le tableau 3.13 montre le pourcentage d'amélioration du coût simulé obtenue par le deuxième ordonnancement par rapport au premier, pour les deux dates de fin étudiées.

Tableau 3.14 : Amélioration moyenne du coût simulé de l'Ordo2 par rapport à l'Ordo1 pour les deux dates de fin étudiées

	Moyenne	Écart-type	Intervalle de confiance à 95%
$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo2})$	-2%	16%	8%
$\delta = 1{,}01 \times E(\mathbf{s}_{n+1Ordo1})$	-63%	19%	10%

Les résultats montrent qu'un ordonnancement initial de plus courte durée ne donne pas une solution significativement plus robuste, selon le coût simulé, lorsque la date de fin est retardée de

1% de l’espérance de la durée de cet ordonnancement. Cependant, lorsque les deux ordonnancements ont la même date de fin, la robustesse est augmentée d’une manière significative.

En résumé, les deux ordonnancements initiaux ont une robustesse plus ou moins égale si la date de fin est reculée de 1% de l’espérance de fin de l’ordonnancement considéré. Cela même si un des ordonnancements a une durée totale plus courte que l’autre. Cependant, si les deux ordonnancements ont la même date de fin, l’ordonnancement qui était plus court dans le cas déterministe sera significativement plus robuste. Cela démontre l’importance de trouver une solution initiale déterministe de bonne qualité.

Pour conclure, la performance de la méthode proposée a été comparée à la méthode originale. L’impact du tampon de temps total ajouté ainsi que l’impact de l’ordonnancement déterministe initial ont aussi été testés. Les résultats sont discutés dans le chapitre suivant.

CHAPITRE 4 DISCUSSION

Ce chapitre discute des résultats obtenus au chapitre précédent. Pour y arriver, les résultats sont d'abord analysés. Ensuite, les limitations de ces constatations sont expliquées. Finalement, les futures directions de recherche sont recensées.

4.1 Analyse des résultats

La mesure de robustesse la plus exacte est le coût simulé. La valeur d'une solution est simulée en effectuant 10 000 répétitions, ce qui nécessite un temps de calcul non négligeable. La méthode proposée part d'un ordonnancement optimisé pour le cas déterministe puis le rend robuste en ajoutant itérativement des tampons de temps. Lorsque les tampons de temps sont ajoutés, la robustesse de l'ordonnancement intermédiaire est évaluée en calculant la somme des STC_i. Cette mesure de robustesse n'est pas parfaitement corrélée avec le coût simulé. Ainsi, l'amélioration de la robustesse d'une solution évaluée en calculant le coût simulé n'est pas aussi grande que lorsque la robustesse est évaluée par la somme des STC_i. Les tests montrent tout de même que les modifications apportées à la méthode permettent d'obtenir une solution plus robuste.

De plus, il est possible d'améliorer considérablement la robustesse d'une solution sans reculer la date de fin du projet. Cela signifie que la méthode devrait être appliquée même si les gestionnaires ne souhaitent pas allonger la durée totale du projet.

Les tests ont aussi mis de l'avant l'importance d'avoir une solution initiale déterministe de bonne qualité. Les efforts déployés pour minimiser la durée totale du projet dans le cas déterministe sont pertinents même lorsque le cas stochastique est considéré.

4.2 Limitations

Il existe plusieurs limitations à la méthode proposée. Elles émanent d'abord des accommodements pris pour effectuer les tests, mais aussi de la définition propre du problème.

Les conclusions sont premièrement limitées aux paramètres étudiés. Les résultats pourraient varier pour des ensembles de poids et de variances générés différemment. Aussi, la politique d'exécution choisie influence le coût simulé. Une autre politique d'exécution pourrait trouver un coût simulé ayant une corrélation plus ou moins grande avec la somme des STC_i. Finalement, les

instances étudiées sont des projets fictifs comportant un nombre d'activités moindre que les projets d'envergure. Le nombre d'instances étudiées est aussi relativement petit.

Deuxièmement, la définition du problème est réductrice lorsqu'un regard est porté d'un point de vu plus global. En effet, il est possible que certains paramètres considérés comme connus ne le soient pas dans la pratique. Par exemple, la détermination d'un poids pour chaque activité peut s'avérer être une tâche ardue, voire impossible, lors de la phase de planification du projet. De plus, le problème étudié considère que l'incertitude ne provient que de la durée des activités. Dans la pratique, la disponibilité des ressources peut aussi être incertaine. Finalement, le problème, tel que formulé, ne prend pas en compte le chevauchement ou la compression possible des activités.

Bref, plusieurs facteurs limitent la portée de la méthode proposée. Il y a donc encore place à amélioration.

4.3 Améliorations possibles

Les perspectives de recherche dans ce domaine sont multiples. D'abord, il serait intéressant d'étudier d'autres politiques d'exécution afin de mesurer leur impact sur le coût simulé. Il serait ainsi être possible de trouver une politique d'exécution mieux corrélée avec la somme des STC_i. Subséquemment, les algorithmes pourraient être codés dans un langage permettant de réduire les temps de calcul, comme le langage C++. Cela permettrait d'envisager l'utilisation de la simulation pour évaluer la robustesse d'une solution intermédiaire durant l'algorithme. Une phase d'amélioration pourrait être ajoutée. Ensuite, l'impact de la solution initiale déterministe sur la robustesse de la solution finale justifie la recherche d'une méthode intégrée permettant de minimiser la durée du projet tout en maximisant la robustesse de l'ordonnancement. La prise en compte des deux objectifs simultanément pourrait améliorer les performances de la méthode. Finalement, le modèle pourrait être bonifié pour permettre de considérer le chevauchement et la compression d'activités. L'ajout d'une politique réactive pourrait aussi renforcer la méthode.

Ce chapitre conclut l'objet de ce mémoire. D'abord, les résultats obtenus ont été analysés. Ensuite, les limites associées ont été soulignées puis les opportunités d'amélioration ont été

identifiées. La méthode proposée se révèle performante en théorie et une application pratique serait la prochaine étape.

CONCLUSION

Ce mémoire propose une nouvelle approche qui permet de trouver un échéancier robuste pour des projets d'envergure. La solution est un ordonnancement qui résiste aux perturbations dans la durée des activités pour des projets comportant un grand nombre d'activités. Cette solution est obtenue en quelques minutes de calcul.

L'avantage de cette méthode est qu'elle part d'une solution initiale admissible pour le cas déterministe, c'est-à-dire en considérant que les durées des activités sont connues. Ainsi, il est possible d'utiliser les méthodes développées pour le cas déterministe qui permettent de trouver un ordonnancement initial minimisant la durée totale du projet. Ensuite, des tampons de temps sont ajoutés devant les activités les plus critiques afin de rendre l'ordonnancement robuste face à des changements dans la durée des activités. La mesure de robustesse utilisée lors de l'ajout des tampons de temps ne recourt pas à la simulation. Bien que la valeur de robustesse obtenue par cette mesure ne soit pas exacte, elle permet de trouver une solution rapidement, même lorsque le nombre d'activités est très grand.

Plusieurs étapes ont été nécessaire afin d'aboutir à une méthode efficace. D'abord, les différentes mesures de robustesse ainsi que les différentes méthodes présentes dans la littérature ont été étudiées. Les mesures qui semblaient les plus appropriées ont été choisies. Ensuite, la méthode la plus pertinente de la littérature actuelle, l'heuristique STC, a été améliorée en proposant deux modifications. La première consiste à favoriser les relations de précédence lors de la génération du réseau de flux de ressources, alors que la deuxième consiste à explorer un voisinage plus large lors de l'ajout des tampons de temps. Les tests numériques ont permis de quantifier cette amélioration. Ils ont aussi permis de montrer que la méthode augmente la robustesse d'un ordonnancement même lorsque la durée totale du projet n'est pas augmentée. Aussi, les tests ont mis en évidence l'importance d'une solution de bonne qualité pour le cas déterministe.

En conclusion, la méthode proposée dans ce mémoire constitue un réel avancement dans la recherche d'ordonnancements robustes pour les projets d'envergure. La contribution à la recherche de cette étude réside dans le développement d'une méthode d'ordonnancement de projet en contexte incertain, et tout particulièrement dans le fait que cette méthode est applicable dans la pratique pour des projets comportant un grand nombre d'activités.

BIBLIOGRAPHIE

Abbasi, B., S. Shadrokh, and J. Arkat. 2006. Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Applied Mathematics and Computation* 180 (1).

Al-Fawzan, M. A., and M. Haouari. 2005. A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics* 96 (2).

Aloulou, M. A., and C. Artigues. 2010. Flexible solutions in disjunctive scheduling: General formulation and study of the flow-shop case. *Computers & Operations Research* 37 (5).

Alsakini, Wafa, Kim Wikström, and Juhani Kiiras. 2004. Proactive schedule management of industrial turnkey projects in developing countries. *International Journal of Project Management* 22 (1).

Artigues, C., P. Michelon, and S. Reusser. 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research* 149 (2).

Aytug, Haldun, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161 (1).

Baradaran, S., S. M. T. F. Ghomi, M. Mobini, and S. S. Hashemin. 2010. A hybrid scatter search approach for resource-constrained project scheduling problem in PERT-type networks. *Advances in Engineering Software* 41 (7-8).

Barraza, G. A. 2011. Probabilistic Estimation and Allocation of Project Time Contingency. *Journal of Construction Engineering and Management-Asce* 137 (4).

Bendavid, I., and B. Golany. 2009. Setting gates for activities in the stochastic project scheduling problem through the cross entropy methodology. *Annals of Operations Research* 172 (1).

Bendavid, I., and B. Golany. 2011. Predetermined intervals for start times of activities in the stochastic project scheduling problem. *Annals of Operations Research* 186 (1).

Berthaut, François. 2013. A modified Scatter Search algorithm for the RCPSP with overlapping. *À soumettre*.

Bhaskar, T., M. N. Pal, and A. K. Pal. 2011. A heuristic method for RCPSP with fuzzy activity times. *European Journal of Operational Research* 208 (1).

Bieber, Jean-Mathieu. 2012. Méthode itérative de résolution de problèmes d'ordonnancement de projets avec contraintes de ressources basée sur des bornes évolutives. In *Département de génie industriel*, 33. Montréal: École Polytechnique de Montréal.

Bruni, M. E., P. Beraldi, F. Guerriero, and E. Pinto. 2011. A heuristic approach for resource constrained project scheduling with uncertain activity durations. *Computers & Operations Research* 38 (9).

Chtourou, H., and M. Haouari. 2008. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & Industrial Engineering* 55 (1).

Deblaere, F., E. Demeulemeester, and W. Herroelen. 2011a. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research* 214 (2).

Deblaere, F., E. Demeulemeester, and W. Herroelen. 2011b. Reactive scheduling in the multi-mode RCPSP. *Computers & Operations Research* 38 (1).

Ghezail, F., H. Pierreval, and S. Hajri-Gabouj. 2010. Analysis of robustness in proactive scheduling: A graphical approach. *Computers & Industrial Engineering* 58 (2).

Goldratt, Eliyahu M. 1997. *Critical chain*. Great Barrington, Mass.: North River Press.

Herman, Steyn. 2001. An investigation into the fundamentals of critical chain project scheduling. *International Journal of Project Management* 19 (6).

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

Herroelen, W., and R. Leus. 2001. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management* 19 (5).

Jensen, M. T. 2003. Generating robust and flexible job shop schedules using genetic algorithms. *Evolutionary Computation, IEEE Transactions on* 7 (3).

Klimek, M., and P. Lebkowski. 2011. Resource allocation for robust project scheduling. *Bulletin of the Polish Academy of Sciences-Technical Sciences* 59 (1).

Klimek, Marcin, and Piotr Łebkowski. 2009. Robust Buffer Allocation for Scheduling of a Project with Predefined Milestones. *Decision Making in Manufacturing and Services* 3 (1-2).

Kolisch, R., and A. Sprecher. 1997. PSPLIB - A project scheduling problem library. *European Journal of Operational Research* 96 (1).

Kolisch, Rainer. 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research* 90 (2).

Lambrechts, O., E. Demeulemeester, and W. Herroelen. 2011. Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research* 186 (1).

Leach, Lawrence P. 2005. *Critical chain project management*. Boston, Mass.: Artech House.

Leus, R., and W. Herroelen. 2004. Stability and resource allocation in project planning. *Iie Transactions* 36 (7).

Long, Luong Duc, and Ario Ohsato. 2008. Fuzzy critical chain method for project scheduling under resource constraints and uncertainty. *International Journal of Project Management* 26 (6).

Newbold, Robert C. 2008. *The billion dollar solution : secrets of prochain project management*. Lake Ridge, VA: ProChain Press.

Park, M., and F. Pena-Mora. 2004. Reliability buffering for construction projects. *Journal of Construction Engineering and Management-Asce* 130 (5).

Policella, N., A. Cesta, A. Oddi, and S. F. Smith. 2009. Solve-and-robustify Synthesizing partial order schedules by chaining. *Journal of Scheduling* 12 (3).

Rabbani, M., S. M. T. F. Ghomi, F. Jolai, and N. S. Lahiji. 2007. A new heuristic for resource-constrained project scheduling in stochastic networks using critical chain concept. *European Journal of Operational Research* 176 (2).

Raz, Tzvi, and Erdal Erel. 2000. Optimal timing of project control points. *European Journal of Operational Research* 127 (2).

Schatteman, D., W. Herroelen, S. Van de Vonder, and A. Boone. 2008. Methodology for Integrated Risk Management and Proactive Scheduling of Construction Projects. *Journal of Construction Engineering and Management-Asce* 134 (11).

Tukel, O. I., W. O. Rom, and S. D. Eksioglu. 2006. An investigation of buffer sizing techniques in critical chain scheduling. *European Journal of Operational Research* 172 (2).

Van de Vonder, S., F. Ballestin, E. Demeulemeester, and W. Herroelen. 2007a. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering* 52 (1).

Van de Vonder, S., E. Demeulemeester, and W. Herroelen. 2007b. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling* 10 (3).

Van de Vonder, S., E. Demeulemeester, and W. Herroelen. 2008. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research* 189 (3).

Van de Vonder, S., E. Demeulemeester, W. Herroelen, and R. Leus. 2005. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics* 97 (2).

Van de Vonder, S., E. Demeulemeester, R. Leus, and W. Herroelen. 2006. Proactive-Reactive Project Scheduling Trade-offs and Procedures. In *Perspectives in modern project scheduling*, Ed. Joanna Jozefowska, 25-57. New York: Springer.

Vieira, GuilhermeE, JeffreyW Herrmann, and Edward Lin. 2003. Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods. *Journal of Scheduling* 6 (1).

Winch, Graham M., and John Kelsey. 2005. What do construction project planners do? *International Journal of Project Management* 23 (2).

Xiuming, Li, Wang Wenjun, and Zeng Xiong. 2010. The study on resource constraint project scheduling problem under stochastic circumstances. In *Environmental Science and Information Application Technology (ESIAT), 2010 International Conference on*, 648-651.

Zhang, X., N. Cui, and Y. Chai. 2011. Timely Project Completin Probability and Stability Cost on the Interaction among Uncertainty of Random Duration, Service Level and Feeding Buffer in a RCPSP Environment. In *International Conference on Mangegement and Service Science (MASS)*. Wuhan: IEEE.

ANNEXE A – PARAMÈTRES UTILISÉS POUR L'EXEMPLE

Le tableau A.1 présente le poids et la variance associés à chaque activité de l'instance 120054 utilisée comme exemple dans la section 3.2.2.

Tableau A.1 : Ensemble des pois et variances pour l'instance utilisée en exemple

i	w_i	var_i
0	0	0
1	5	3
2	6	1
3	0	2
4	0	3
5	3	1
6	10	1
7	0	2
8	6	2
9	8	3
10	0	1
11	2	2
12	3	3
13	2	2
14	8	3
15	0	3
16	0	3
17	0	2
18	0	3
19	0	3
20	1	3
21	0	3
22	9	2
23	0	3
24	3	1
25	8	2
26	3	3
27	0	1
28	6	1
29	0	2
30	3	3
31	0	1
32	0	1
33	4	3
34	6	2
35	0	3
36	0	2
37	5	3
38	0	1
39	0	2
40	3	1
41	2	3
42	2	1
43	0	2
44	0	1
45	4	3
46	5	1
47	1	1
48	6	2
49	0	3
50	0	2
51	3	2
52	0	1
53	0	2
54	4	2
55	3	1
56	0	1
57	5	3
58	8	1
59	2	3
60	2	3
61	4	1
62	0	1
63	7	2
64	0	3
65	2	2
66	3	2
67	2	1
68	1	1
69	0	1
70	0	2
71	1	1
72	5	1
73	7	1
74	0	2
75	0	2
76	0	3
77	3	2
78	0	3
79	0	1
80	2	2
81	7	2
82	1	1
83	3	1
84	4	1
85	2	2
86	0	3
87	3	1
88	0	1
89	0	3
90	3	1
91	9	3
92	0	2
93	3	2
94	0	2
95	3	2
96	3	3
97	6	2
98	0	1
99	3	3
100	8	2
101	0	1
102	1	1
103	2	3
104	0	3
105	0	3
106	1	1
107	4	3
108	0	3
109	2	1
110	0	2
111	0	2
112	0	1
113	1	1
114	0	3
115	7	3
116	0	3
117	0	2
118	1	2
119	0	1
120	0	2
121	38	0

ANNEXE B – CODE UTILISÉ POUR EFFECTUER LES TESTS

Code permettant de générer un réseau de flux de ressources (RFN) :

Algorithme *Genflow* original et modifié

```
Option Explicit
Option Base 1

Dim phase As String

Dim nbActivites As Integer
Dim DonneesSuccesseurs As Range

Dim NbSuccBase() As Integer, SuccBase() As Integer, NbSuccBaseMax As Integer
Dim NbPredBase() As Integer, PredBase() As Integer, NbPredBaseMax As Integer

Dim NbR As Integer, R() As Integer, R_limite() As Integer
Dim Flow() As Integer, requis() As Integer
Dim j_ordre_croissant_ST() As Integer
Dim d() As Integer, ST() As Integer

Dim nbpredRFN() As Integer, predRFN() As Integer

Dim i As Integer, j As Integer, k As Integer

Dim Modifie As Boolean, reponse As VbMsgBoxStyle

Sub Predecesseurs_et_RFN()

    Call parametres

    'Predecesseurs
    Call DetruireEtCreer("PredBase", "PredBase-" & phase)
    Call TableauDonneesSansEntete(DonneesSuccesseurs, "A5", "Successeurs-" & phase)
    Call Initialisation_pred
    Call Inscrire_Predbase

    'RFN
    Call nbressources
    Call DetruireEtCreer("PredBase", "PredRFN-" & phase)
    Call Initialisation_RFN
    Call Tri_Croissant_ST
    Call calcul_RFN
    Call Inscrire_PredRFN
```

```
End Sub

Sub parametres()
   phase = InputBox("Quel phase (P1, P2, ...)?", "Pi") 'La variable reçoit la valeur entrée dans l'InputBox
   reponse = MsgBox("RFN Modifié ou non?", vbYesNo) 'La variable reçoit la valeur entrée dans l'InputBox
   If reponse = vbNo Then
      Modifie = False
   Else: Modifie = True
   End If
End Sub
Sub nbressources()
   NbR = InputBox("Combien de ressources?", "Ressources") 'La variable reçoit la valeur entrée dans l'InputBox
End Sub

Sub DetruireEtCreer(nomCopie As String, nomDetruire As String)
'   Cette sous-routine :
'      1- détruit la feuille Excel dont le nom est nomDetruire si cette feuille existe;
'      2- crée une copie de la feuille dont le nom est nomCopie après cette dernière;
'      3- et attribue le nom nomDetruire à la nouvelle feuille.
'

Dim ws As Worksheet

Application.DisplayAlerts = False
For Each ws In Worksheets
   If ws.Name = nomDetruire Then ws.Delete
Next
Application.DisplayAlerts = True

Worksheets(nomCopie).Copy after:=Worksheets(nomCopie)
ActiveSheet.Name = nomDetruire
End Sub

Sub TableauDonneesSansEntete(Tableau As Range, PointDepart As String, Feuille As String)
'   Cette sous-routine crée un objet Range appelé Tableau avec le tableau de donnees dont le coin supérieur
'   gauche est situé sur la cellule PointDepart sur la feuille Excel appelée Feuille
'

With Worksheets(Feuille).Range(PointDepart)
   Set Tableau = Range(.Offset(0, 0), .End(xlDown).End(xlToRight))
End With

End Sub
```

```
Sub Initialisation_pred()
'   Cette sous-routine calcule le nombre d'activités.
'   Elle redimensionne tous les vecteurs utilisés
'   Elle donne aux vectuers les valeurs connues, contenues dans les feuilles excels.

    nbActivites = DonneesSuccesseurs.Rows.Count


    'Successeurs de base
    ReDim NbSuccBase(nbActivites)
    ReDim NbPredBase(nbActivites)

    NbSuccBaseMax = 0
    NbSuccBaseMax = 0

    With Worksheets("Successeurs-" & phase).Range("C4")
       For j = 1 To nbActivites
          NbSuccBase(j) = .Offset(j, 0).Value
          If NbSuccBase(j) > NbSuccBaseMax Then
             NbSuccBaseMax = NbSuccBase(j)
          End If
       Next

    NbPredBaseMax = 2 * NbSuccBaseMax

    ReDim SuccBase(nbActivites, NbSuccBaseMax)
    ReDim PredBase(nbActivites, NbPredBaseMax)

       For j = 1 To nbActivites
          For i = 1 To NbSuccBase(j)
             SuccBase(j, i) = .Offset(j, i).Value

             NbPredBase(SuccBase(j, i)) = NbPredBase(SuccBase(j, i)) + 1
             PredBase(SuccBase(j, i), NbPredBase(SuccBase(j, i))) = j
          Next
       Next
    End With


End Sub

Sub Inscrire_Predbase()
    With Worksheets("PredBase-" & phase).Range("B4")

       For j = 1 To nbActivites
          .Offset(j, 0) = j
```

```
            .Offset(j, 1) = NbPredBase(j)

            For i = 1 To NbPredBase(j)
                .Offset(j, 1 + i) = PredBase(j, i)
            Next
        Next

    End With
End Sub

Sub Initialisation_RFN()

    ReDim d(nbActivites)
    ReDim ST(nbActivites)
    ReDim R(nbActivites, NbR)
    ReDim R_limite(NbR)
    ReDim Flow(nbActivites, nbActivites, NbR)
    ReDim j_ordre_croissant_ST(nbActivites)
    ReDim requis(NbR)

    ReDim nbpredRFN(nbActivites)
    ReDim predRFN(nbActivites, nbActivites)

    With Worksheets("Donnes-" & phase).Range("A6")
        For j = 1 To nbActivites
            d(j) = .Offset(j, 4).Value
            ST(j) = .Offset(j, 1).Value

            For k = 1 To NbR
                R(j, k) = .Offset(j, 4 + k).Value
            Next
        Next

        For k = 1 To NbR
                R_limite(k) = .Offset(-2, 4 + k).Value
        Next
    End With

    For i = 1 To nbActivites
        j_ordre_croissant_ST(i) = i

        For j = 1 To nbActivites
            For k = 1 To NbR
                Flow(i, j, k) = 0
            Next
        Next
    Next
```

```
   For k = 1 To NbR
      Flow(1, nbActivites, k) = R_limite(k)
   Next

End Sub

Sub Tri_Croissant_ST()
   Dim temporaire, t As Integer
   Dim permut As Boolean

For j = 2 To nbActivites
   If ST(j_ordre_croissant_ST(j)) < ST(j_ordre_croissant_ST(j - 1)) Then
      i = j + 1
      temporaire = j_ordre_croissant_ST(j)
      permut = False
      t = 0
      Do
         i = i - 1
         j_ordre_croissant_ST(i) = j_ordre_croissant_ST(i - 1)
         If i = 2 Then
            permut = True
            t = 1
         End If
      Loop While ST(temporaire) < ST(j_ordre_croissant_ST(i - 2 + t)) And permut = False
      j_ordre_croissant_ST(i - 1) = temporaire
   End If
Next
End Sub

Sub calcul_RFN()

   Dim a, q As Integer
   Dim p As Integer
   Dim deja_pred As Boolean

For a = 2 To nbActivites
   j = j_ordre_croissant_ST(a)

   'Algorithme UPDATEFLOW(j,F) de artigues (2003) modifié pour favoriser les prédécesseurs

      For k = 1 To NbR
         requis(k) = R(j, k)
      Next

     If Modifie = True Then
      For i = 1 To NbPredBase(j)
```

```
            For k = 1 To NbR

               'q = Min(requis(k), Flow(i, nbActivites, k))
               q = Flow(PredBase(j, i), nbActivites, k)
               If requis(k) < q Then
                  q = requis(k)
               End If

               If q > 0 Then
                  requis(k) = requis(k) - q
                  Flow(PredBase(j, i), nbActivites, k) = Flow(PredBase(j, i), nbActivites, k) - q
                  Flow(PredBase(j, i), j, k) = Flow(PredBase(j, i), j, k) + q
               End If
            Next
         Next
        End If

        For k = 1 To NbR
           If requis(k) > 0 Then
              i = 1
              Do
                  If ST(i) + d(i) <= ST(j) Then
                    'q = Min(requis(k), Flow(i, nbActivites, k))
                    q = Flow(i, nbActivites, k)
                    If requis(k) < q Then
                       q = requis(k)
                    End If

                    If q > 0 Then
                       requis(k) = requis(k) - q
                       Flow(i, nbActivites, k) = Flow(i, nbActivites, k) - q
                       Flow(i, j, k) = Flow(i, j, k) + q
                    End If
                  End If
                    i = i + 1
              Loop While requis(k) > 0
           End If

           Flow(j, nbActivites, k) = R(j, k)
        Next
Next

For j = 1 To nbActivites
   nbpredRFN(j) = 0
   For i = 1 To nbActivites
      For k = 1 To NbR
      If Flow(i, j, k) > 0 Then
```

```
            'Vérifier si i n'est pas déjà prédécesseur de j
            deja_pred = False
            For p = 1 To nbpredRFN(j)
               If predRFN(j, p) = i Then
                  deja_pred = True
               End If
            Next
            If deja_pred = False Then
               nbpredRFN(j) = nbpredRFN(j) + 1
               predRFN(j, nbpredRFN(j)) = i
            End If
         End If
         Next
      Next
   Next
End Sub

Sub Inscrire_PredRFN()
   With Worksheets("PredRFN-" & phase).Range("B4")

      For j = 1 To nbActivites
         .Offset(j, 0) = j
         .Offset(j, 1) = nbpredRFN(j)

         For i = 1 To nbpredRFN(j)
            .Offset(j, 1 + i) = predRFN(j, i)
         Next
      Next

   End With
End Sub
```

Code permettant d'insérer des tampons de temps :

Algorithme STC original et modifié

```
Option Explicit
Option Base 1

Dim Horloge As Single, Horloge_debut As Single

Dim Phase As String, PhasePred As String

Dim nbactivites As Integer
Dim DonneesActivites As Range

Dim s() As Integer, d() As Integer, dbuffer() As Integer, w() As Double
Dim activites_particulieres() As Integer, nbAct_part As Integer

Dim NbPredBase() As Integer, PredBase() As Integer, NbPredBaseMax As Integer
Dim NbPredRFN() As Integer, PredRFN() As Integer, NbPredRFNMax As Integer
Dim PredTotal() As Integer, NbPredTotalMax As Integer, NbPredTotal() As Integer
Dim Successeurs() As Integer, NbSuccesseurs() As Integer
Dim NbSuccesseurs_restant() As Integer, fermer() As Boolean
Dim Pred As Boolean

Dim LPL() As Integer, LPLb() As Boolean

Dim Xvalue As Double
Dim var() As Integer, Mind As Double, Maxd As Double
Const alpha = 2, beta = 5
Dim STC() As Double, y() As Double

Dim permutation() As Integer
Dim permut As Boolean
Dim tempo As Integer, t As Integer

Dim updated_s() As Integer
Dim changement As Boolean

Dim SommeSTC As Double, UpdatedSommeSTC As Double
Dim modification() As Double

Dim BufferAjouter As Boolean
Dim Faisable As Boolean, amelioration As Boolean

Dim debut_au_plus_tot, debut_au_plus_tard As Integer
```

```
Dim i As Integer, j As Integer, n As Integer, p As Integer

Dim nombre_Phase As Integer

Sub HeuristiqueSTC()

Phase = InputBox("Quel phase (P1, P2, ...)?", "Pi") 'La variable reçoit la valeur entrée dans l'InputBox

   Horloge_debut = Timer

   Call Initialisation
   Call CalculLPL
   Call CalculerSTC

   Horloge = Timer - Horloge_debut
   Call Ecrire_solution("K6")

   Do

      Call Tri_Decroissant_STC    'Trier les activites en ordre décroissant de leur STC
      Call Ajouter_un_buffer      'Ajouter, si c'est possible, un buffer devant l'activite j ayant le STC Max, si STC(j)>0

   Loop While BufferAjouter = True

   Horloge = Timer - Horloge_debut

   Call Ecrire_solution("L6")

End Sub

Sub DetruireEtCreer(nomCopie As String, nomDetruire As String)
'
'   Cette sous-routine :
'       1- détruit la feuille Excel dont le nom est nomDetruire si cette feuille existe;
'       2- crée une copie de la feuille dont le nom est nomCopie après cette dernière;
'       3- et attribue le nom nomDetruire à la nouvelle feuille.
'
Dim ws As Worksheet

Application.DisplayAlerts = False
For Each ws In Worksheets
   If ws.Name = nomDetruire Then ws.Delete
Next
Application.DisplayAlerts = True
```

```
Worksheets(nomCopie).Copy after:=Worksheets(nomCopie)
ActiveSheet.Name = nomDetruire
End Sub
Sub TableauDonneesSansEntete(Tableau As Range, PointDepart As String, Feuille As String)
'
'   Cette sous-routine crée un objet Range appelé Tableau avec le tableau de donnees dont le coin
supérieur
'   gauche est situé sur la cellule PointDepart sur la feuille Excel appelée Feuille
'

With Worksheets(Feuille).Range(PointDepart)
   Set Tableau = Range(.Offset(0, 0), .End(xlDown).End(xlToRight))
End With
End Sub

Sub Initialisation()
'
'   Cette sous-routine crée les nouveaux onglets et calcule le nombre d'activités.
'   Elle indique également la taille de tous les vecteurs utilisés et les valeurs de départ.

   PhasePred = Left(Phase, 3)

   Call DetruireEtCreer("PredBase-" & PhasePred, "Predecesseurs-" & PhasePred)

   Call DetruireEtCreer("Donnes-" & Phase, "Ordonnancement-" & Phase)
   Call TableauDonneesSansEntete(DonneesActivites, "A7", "Ordonnancement-" & Phase)


   nbactivites = DonneesActivites.Rows.count

   ReDim s(nbactivites)
   ReDim updated_s(nbactivites)
   ReDim LPL(nbactivites, nbactivites)
   ReDim LPLb(nbactivites, nbactivites)
   ReDim STC(nbactivites)
   ReDim y(nbactivites)
   ReDim modification(nbactivites)

   ReDim NbSuccesseurs(nbactivites)
   ReDim NbSuccesseurs_restant(nbactivites)
   ReDim fermer(nbactivites)

   ReDim d(nbactivites)
   ReDim dbuffer(nbactivites)
   ReDim w(nbactivites)
   nbAct_part = 0
   ReDim activites_particulieres(nbactivites)
```

```
ReDim var(nbactivites)

ReDim permutation(nbactivites)

For j = 1 To nbactivites
   d(j) = DonneesActivites(j, 5)
   dbuffer(j) = 0

   w(j) = DonneesActivites(j, 3)

   var(j) = DonneesActivites(j, 4)

   s(j) = DonneesActivites(j, 2)
   updated_s(j) = s(j)

   NbSuccesseurs(j) = 0

   permutation(j) = j
Next

'Prédécesseurs de base
ReDim NbPredBase(nbactivites)
NbPredBaseMax = 0
With Worksheets("PredBase-" & PhasePred).Range("C4")
   For j = 1 To nbactivites
      NbPredBase(j) = .Offset(j, 0).Value
      If NbPredBase(j) > NbPredBaseMax Then
         NbPredBaseMax = NbPredBase(j)
      End If
   Next

   ReDim PredBase(nbactivites, NbPredBaseMax)
   For j = 1 To nbactivites
      For i = 1 To NbPredBase(j)
         PredBase(j, i) = .Offset(j, i).Value - 1        "-1: pour que les activités commencent à 0
      Next
   Next
End With

'Prédécesseurs dans le RFN
ReDim NbPredRFN(nbactivites)
NbPredRFNMax = 0
With Worksheets("PredRFN-" & PhasePred).Range("C4")
   For j = 1 To nbactivites
      NbPredRFN(j) = .Offset(j, 0).Value
      If NbPredRFN(j) > NbPredRFNMax Then
         NbPredRFNMax = NbPredRFN(j)
```

```
        End If
    Next

    ReDim PredRFN(nbactivites, NbPredRFNMax)
    For j = 1 To nbactivites
        For i = 1 To NbPredRFN(j)
            PredRFN(j, i) = .Offset(j, i).Value - 1          "'-1: pour que les activités commencent à 0
        Next
    Next
End With

'Déterminer le total des prédécesseurs et des successeurs de chaque activite
NbPredTotalMax = NbPredBaseMax + NbPredRFNMax
ReDim NbPredTotal(nbactivites)
ReDim PredTotal(nbactivites, NbPredTotalMax)

ReDim Successeurs(nbactivites, nbactivites)
    'Mettre les predécesseurs de base
For j = 1 To nbactivites
    NbPredTotal(j) = NbPredBase(j)
    For i = 1 To NbPredBase(j)
        PredTotal(j, i) = PredBase(j, i)

        NbSuccesseurs(PredTotal(j, i) + 1) = NbSuccesseurs(PredTotal(j, i) + 1) + 1
        Successeurs(PredTotal(j, i) + 1, NbSuccesseurs(PredTotal(j, i) + 1)) = j - 1
    Next
Next
    'Ajouter les prédécesseurs du RFN
For j = 1 To nbactivites
    For i = 1 To NbPredRFN(j)
        Pred = True
        For n = 1 To NbPredBase(j)
            If PredBase(j, n) = PredRFN(j, i) Then
                Pred = False
            End If
        Next
        If Pred = True Then
            NbPredTotal(j) = NbPredTotal(j) + 1
            PredTotal(j, NbPredTotal(j)) = PredRFN(j, i)

            NbSuccesseurs(PredRFN(j, i) + 1) = NbSuccesseurs(PredRFN(j, i) + 1) + 1
            Successeurs(PredRFN(j, i) + 1, NbSuccesseurs(PredRFN(j, i) + 1)) = j - 1
        End If
    Next
Next

'Écrire tous les Prédécesseurs
```

```
With ThisWorkbook.Worksheets("Predecesseurs-" & PhasePred).Range("B4")
   For j = 1 To nbactivites
      .Offset(j, 0).Value = j - 1
      .Offset(j, 1).Value = NbPredTotal(j)
       For i = 1 To NbPredTotal(j)
          .Offset(j, i + 1).Value = PredTotal(j, i)
       Next
    Next
 End With
End Sub

Sub CalculLPL()

For j = 1 To nbactivites
   fermer(j) = False
   NbSuccesseurs_restant(j) = NbSuccesseurs(j)
   For i = 1 To nbactivites
      LPL(i, j) = 0
      LPLb(i, j) = False
   Next
Next

j = nbactivites

Do
   'Calculer les LPL entre les prédécesseurs de j et tous leurs successeurs directes et transitifs
   For i = 1 To NbPredTotal(j)

      NbSuccesseurs_restant(PredTotal(j, i) + 1) = NbSuccesseurs_restant(PredTotal(j, i) + 1) - 1
'ôter un successeur au prédécesseur

      'Calcul des LPL pour les succésseurs directes
      LPLb(PredTotal(j, i) + 1, j) = True

      'Calcul des LPL pour les succésseurs transitifs
      For n = 1 To nbactivites
         If LPLb(j, n) = True Then
            LPLb(PredTotal(j, i) + 1, n) = True
            If LPL(PredTotal(j, i) + 1, n) < (d(j) + LPL(j, n)) Then
               LPL(PredTotal(j, i) + 1, n) = (d(j) + LPL(j, n))
            End If
         End If
      Next
   Next

   fermer(j) = True
```

```
   j = 0
   m = nbactivites
   Do
      If fermer(m) = False And NbSuccesseurs_restant(m) = 0 Then
         j = m
      End If
      m = m - 1
   Loop While j = 0 And m > 0

Loop While m > 0
End Sub

Sub CalculerSTC()

For j = 1 To nbactivites
   STC(j) = 0
   y(j) = 0
Next

SommeSTC = 0

For j = 1 To nbactivites
   If s(j) > 0 Then    ' Par définition, les STC des activites qui commencent à 0 égalent 0
      For i = 1 To nbactivites

         'calculer y(j) = Somme pour tous les prédécesseurs directes et transitifs i de j de: P(di >
Sj-Si-LPL(i,j))
         If LPLb(i, j) = True Then
            Call Distribution_Beta(y(j), i, j, s())
         End If
      Next
      STC(j) = y(j) * w(j)
      SommeSTC = SommeSTC + STC(j)
   End If
Next
End Sub

Sub Tri_Decroissant_STC()

For j = 2 To nbactivites
   If STC(permutation(j)) > STC(permutation(j - 1)) Then
      i = j + 1
      tempo = permutation(j)
      permut = False
      t = 0
      Do
         i = i - 1
```

```
            permutation(i) = permutation(i - 1)
            If i = 2 Then
               permut = True
               t = 1
            End If
         Loop While STC(tempo) > STC(permutation(i - 2 + t)) And permut = False
         permutation(i - 1) = tempo
      End If
   Next
   End Sub

   Sub Ajouter_un_buffer()

   BufferAjouter = False

   p = 1
   j = permutation(p)      'Prendre l'activite j avec le STC max

   Do While STC(j) > 0 And BufferAjouter = False

      Faisable = True
      amelioration = False

      dbuffer(j) = dbuffer(j) + 1

      'Updater les s(n) des activités successeurs directes et transitifs de j
         Call Updater_s_modified 'ôter «_modified» pour la version originale

         If updated_s(nbactivites) > s(nbactivites) Then
            Faisable = False
         End If

      If Faisable = True Then

         'Vérifier s'il y a une amélioration, c'est-à-dire une diminution de la somme des STC
         Call Updater_STC
         If UpdatedSommeSTC < SommeSTC Then     'Il faut améliorer la solution d'au moins 0,01,
   sinon ça compte pas.
            amelioration = True
         End If

      End If

      If amelioration = True Then

         BufferAjouter = True
```

```
        'Enregistrer les nouvelles valeurs de STC et de temps de début s(n)
        SommeSTC = UpdatedSommeSTC
        For n = 1 To nbactivites
            y(n) = modification(n)
            STC(n) = w(n) * y(n)

            s(n) = updated_s(n)
        Next

    Else

        dbuffer(j) = dbuffer(j) - 1

        'Restaurer les temps de débuts des activités
        For n = 1 To nbactivites
            updated_s(n) = s(n)
        Next

    End If

    p = p + 1
    j = permutation(p)

Loop
End Sub

Sub Updater_s()

'Version originale

    updated_s(j) = updated_s(j) + 1

    changement = False
    For i = 1 To NbSuccesseurs(j)
        If updated_s(j) + d(j) > updated_s(Successeurs(j, i) + 1) - dbuffer(Successeurs(j, i) + 1)
Then
            updated_s(Successeurs(j, i) + 1) = updated_s(Successeurs(j, i) + 1) + 1
            fermer(Successeurs(j, i) + 1) = False
            changement = True
        End If
    Next

    Do While changement = True
        changement = False
        For n = 1 To nbactivites
            If fermer(n) = False Then
                For i = 1 To NbSuccesseurs(n)
```

```
            If updated_s(n) + d(n) > updated_s(Successeurs(n, i) + 1) - dbuffer(Successeurs(n, i)
+ 1) Then
               updated_s(Successeurs(n, i) + 1) = updated_s(Successeurs(n, i) + 1) + 1
               fermer(Successeurs(n, i) + 1) = False
               changement = True
            End If
         Next
         fermer(n) = True
      End If
   Next
Loop
End Sub

Sub Updater_s_modified()

'Permettre de diminuer un buffer déjà placé

   updated_s(j) = updated_s(j) + 1

   changement = False
   For i = 1 To NbSuccesseurs(j)
      If updated_s(j) + d(j) > updated_s(Successeurs(j, i) + 1) - dbuffer(Successeurs(j, i) + 1)
Then
         If dbuffer(Successeurs(j, i) + 1) > 0 Then
            dbuffer(Successeurs(j, i) + 1) = dbuffer(Successeurs(j, i) + 1) - 1
         Else
            updated_s(Successeurs(j, i) + 1) = updated_s(Successeurs(j, i) + 1) + 1
            fermer(Successeurs(j, i) + 1) = False
            changement = True
         End If
      End If
   Next

   Do While changement = True
      changement = False
      For n = 1 To nbactivites
         If fermer(n) = False Then
            For i = 1 To NbSuccesseurs(n)
               If updated_s(n) + d(n) > updated_s(Successeurs(n, i) + 1) - dbuffer(Successeurs(n, i)
+ 1) Then
                  If dbuffer(Successeurs(n, i) + 1) > 0 Then
                     dbuffer(Successeurs(n, i) + 1) = dbuffer(Successeurs(n, i) + 1) - 1
                  Else
                     updated_s(Successeurs(n, i) + 1) = updated_s(Successeurs(n, i) + 1) + 1
                     fermer(Successeurs(n, i) + 1) = False
                     changement = True
                  End If
```

```
            End If
         Next
         fermer(n) = True
      End If
   Next
 Loop
End Sub

Sub Updater_STC()

UpdatedSommeSTC = 0

'Recalculer les y() de j et de ces successeurs
For n = 1 To nbactivites
 If LPLb(j, n) = True Or n = j Then
   modification(n) = 0
   For i = 1 To nbactivites - 1
     If LPLb(i, n) = True Then
       Call Distribution_Beta(modification(n), i, n, updated_s())
     End If
   Next
 Else
   modification(n) = y(n)
 End If
 UpdatedSommeSTC = UpdatedSommeSTC + w(n) * modification(n)
Next

End Sub

Sub Distribution_Beta(Resultat As Double, predecesseur As Integer, successeur As Integer, Vecteur_s() As Integer)

      If var(predecesseur) > 0 Then
          Select Case var(predecesseur)
            Case 1     'Low
                  Mind = 0.75 * d(predecesseur)
                  Maxd = 1.625 * d(predecesseur)
            Case 2     'Medium
                  Mind = 0.5 * d(predecesseur)
                  Maxd = 2.25 * d(predecesseur)
            Case 3      'High
                  Mind = 0.25 * d(predecesseur)
                  Maxd = 2.875 * d(predecesseur)
          End Select
          Xvalue = Vecteur_s(successeur) - Vecteur_s(predecesseur) - LPL(predecesseur, successeur) + 0.5   'X-value = durée de i + marge entre i et j
          If Xvalue <= Maxd Then
```

```
                If Xvalue <= Mind Then
                Resultat = Resultat + 1
                Else
                   Resultat = Resultat + 1 - Application.WorksheetFunction.BetaDist(Xvalue,
alpha, beta, Mind, Maxd)
                End If
             End If
          End If
End Sub

Sub Ecrire_solution(Range As String)

With ThisWorkbook.Worksheets("Ordonnancement-" & Phase).Range(Range)
   For j = 1 To nbactivites
      .Offset(j, 0).Value = s(j)
   Next
   .Offset(-2, 0).Value = SommeSTC
   .Offset(-3, 0).Value = Horloge
End With
End Sub
```

Code permettant de mesurer la robustesse d'une solution par simulation:

Algorithme du coût simulé

```
Option Explicit
Option Base 1

Dim Phase As String, PhasePred As String

Dim nbactivites As Integer
Dim DonneesActivites As Range

Dim d() As Integer, w() As Double
Dim activites_particulieres() As Integer, nbAct_part As Integer

Dim NbPredBase() As Integer, PredBase() As Integer, NbPredBaseMax As Integer

Dim Xvalue As Double
Dim var() As Integer, Mind As Double, Maxd As Double
Const alpha = 2, beta = 5

Dim i As Integer, j As Integer

Dim NbR As Integer, r() As Integer, R_limite() As Integer

Dim nbpredRFN() As Integer, predRFN() As Integer, NbPredRFNMax As Integer

Dim k As Integer

Dim Mois As String
Dim Periode As Integer

Dim debut_periode As Integer, fin_periode As Integer
Dim dans_periode() As Integer, nbdans_periode As Integer
Dim apres_periode() As Integer, nbapres_periode As Integer
Dim deplacable() As Boolean
Dim s_jalon_bidon As Integer
Dim nbpredjalon As Integer, predjalon() As Integer
Dim retard_jalon As Boolean

Dim s_prevu() As Integer, s_realise() As Integer
Dim d_realise() As Integer
Dim j_fini() As Boolean

Dim R_dates_en_cours() As Integer
```

```
Dim nbsimulations As Integer
Dim Date_en_cours As Integer
Dim cout_periode As Double, cout_simulations As Double, cout_meilleur As Double

Dim nb_Retards_dans() As Integer, nb_Retards_apres() As Integer

Dim repet As Integer
Dim a As Integer, s As Integer

Dim tous_periode() As Integer
Dim nbtous_periode As Integer

Dim Tri_Pire_activite() As Integer
Dim nb_retards() As Integer
Dim cout_essai_meilleur As Double

Dim phase_suivante As String
Dim m As Integer

Dim esperance_fin As Single
Dim Horloge As Single, Horloge_debut As Single

Dim fois_rendu As Integer
Dim nombre_Phase As Integer

'_____________________________________________________
Const longueur = 500 'longueur d'une période en nombre de jours
Const nombre = 1000 'nb de simulations
Const fois = 10 'nb de fois que les simulations sont faites

Sub Evaluation_par_Simulation()

   Randomize

Phase = InputBox("Quel phase (P1, P2, ...)?", "Pi") 'La variable reçoit la valeur entrée dans l'InputBox

   Horloge_debut = Timer

   Call Initialisation_generale

   Mois = InputBox("Dans quelle rangée sont les temps de début?", "X0")

      'Inscrire les dates de début prévus                    ********
      With ThisWorkbook.Worksheets("Ordonnancement-" & Phase).Range(Mois)
         For j = 1 To nbactivites '                          ********
            s_prevu(j) = .Offset(j, 0).Value
```

```
        Next
    End With


    Periode = 1
    fois_rendu = 1
    Call DetruireEtCreer("Resultat", "Resultat-" & Phase & "-" & Mois)

    Do

        Call Construction_instance_Periode
        Call Tri_Croissant_EBST1    'Tri Croissant des s_Prevu, briser les égalités par w max

        'Donner les bonnes valeurs à tous_periode()
        For a = 1 To nbdans_periode
            tous_periode(a) = dans_periode(a)
        Next
        For a = 1 To nbapres_periode
            tous_periode(a + nbdans_periode) = apres_periode(a)
        Next

        '********************
        Call Simulation(nombre)
        '********************

        Horloge = Timer - Horloge_debut
        """""""""""
        With ThisWorkbook.Worksheets("Resultat-" & Phase & "-" & Mois).Range("B2")
            .Offset(3, fois_rendu + 1).Value = fois_rendu
            .Offset(4, fois_rendu + 1).Value = cout_simulations
            .Offset(5, fois_rendu + 1).Value = esperance_fin
            .Offset(6, fois_rendu + 1).Value = Horloge
        End With
        """""""""""""""


        fois_rendu = fois_rendu + 1
    Loop While fois_rendu <= fois

End Sub
Sub Simulation(nombre As Integer)
'Effectuer des simulations

    nbsimulations = nombre
    cout_simulations = 0
    esperance_fin = 0

    'Mettre le nombre de retards à 0
```

```
    For a = 1 To nbdans_periode
        nb_Retards_dans(a) = 0
    Next
    For a = 1 To nbapres_periode
        nb_Retards_apres(a) = 0
    Next


    For s = 1 To nbsimulations

        Date_en_cours = 0
        'Poser que toutes les activitées de la période ne sont pas finient
        For a = 1 To nbdans_periode
            j_fini(dans_periode(a)) = False
        Next

        'Trouver une durée réelle pour chaque activitées
        For a = 1 To nbdans_periode
            Call Distribution_Beta__Duree_Reelle(dans_periode(a))
        Next
        For a = 1 To nbapres_periode
            Call Distribution_Beta__Duree_Reelle(apres_periode(a))
        Next

        Call Politique__RP_SGS      'Robust Parallel - Schedule Generation Sheme

        Call Calculer_Cout_Simulation

    Next

    cout_simulations = cout_simulations / nbsimulations
    esperance_fin = esperance_fin / nbsimulations

End Sub

Sub DetruireEtCreer(nomCopie As String, nomDetruire As String)
'
'   Cette sous-routine :
'       1- détruit la feuille Excel dont le nom est nomDetruire si cette feuille existe;
'       2- crée une copie de la feuille dont le nom est nomCopie après cette dernière;
'       3- et attribue le nom nomDetruire à la nouvelle feuille.
'

Dim ws As Worksheet

Application.DisplayAlerts = False
For Each ws In Worksheets
```

```
    If ws.Name = nomDetruire Then ws.Delete
Next
Application.DisplayAlerts = True

Worksheets(nomCopie).Copy after:=Worksheets(nomCopie)
ActiveSheet.Name = nomDetruire
End Sub
Sub TableauDonneesSansEntete(Tableau As Range, PointDepart As String, Feuille As String)
'
'   Cette sous-routine crée un objet Range appelé Tableau avec le tableau de donnees dont le coin
supérieur
'   gauche est situé sur la cellule PointDepart sur la feuille Excel appelée Feuille
'


With Worksheets(Feuille).Range(PointDepart)
    Set Tableau = Range(.Offset(0, 0), .End(xlDown).End(xlToRight))
End With

End Sub
Sub Initialisation_generale()
'
'   Cette sous-routine crée les nouveaux onglets et calcule le nombre d'activités.
'   Elle indique également la taille de tous les vecteurs utilisés et les valeurs de départ.
'
    PhasePred = Left(Phase, 3)

    Call TableauDonneesSansEntete(DonneesActivites, "A7", "Ordonnancement-" & Phase)

    nbactivites = DonneesActivites.Rows.count

    ReDim d(nbactivites)
    ReDim w(nbactivites)
    ReDim var(nbactivites)

    'ReDim permutation(nbactivites)

    For j = 1 To nbactivites
        d(j) = DonneesActivites(j, 5)
        w(j) = DonneesActivites(j, 3)
        var(j) = DonneesActivites(j, 4)
    Next

    'Prédécesseurs de base
    ReDim NbPredBase(nbactivites)
    NbPredBaseMax = 0
    With Worksheets("PredBase-" & PhasePred).Range("C4")
        For j = 1 To nbactivites
```

```
            NbPredBase(j) = .Offset(j, 0).Value
            If NbPredBase(j) > NbPredBaseMax Then
               NbPredBaseMax = NbPredBase(j)
            End If
         Next

      ReDim PredBase(nbactivites, NbPredBaseMax)
         For j = 1 To nbactivites
            For i = 1 To NbPredBase(j)
               PredBase(j, i) = .Offset(j, i).Value - 1
            Next
         Next
      End With

      'Prédécesseurs dans le RFN
      ReDim nbpredRFN(nbactivites)
      NbPredRFNMax = 0
      With Worksheets("PredRFN-" & PhasePred).Range("C4")
         For j = 1 To nbactivites
            nbpredRFN(j) = .Offset(j, 0).Value
            If nbpredRFN(j) > NbPredRFNMax Then
               NbPredRFNMax = nbpredRFN(j)
            End If
         Next

      ReDim predRFN(nbactivites, nbactivites)
         For j = 1 To nbactivites
            For i = 1 To nbpredRFN(j)
               predRFN(j, i) = .Offset(j, i).Value - 1
            Next
         Next
      End With


ReDim s_prevu(nbactivites)
ReDim s_realise(nbactivites)
ReDim d_realise(nbactivites)

ReDim j_fini(nbactivites)
ReDim dans_periode(nbactivites)
ReDim apres_periode(nbactivites)
ReDim deplacable(nbactivites)

ReDim Pire_activite(2)  '1: Activité    2: Coût de l'activité

'----------
'Ressources
```

```
'----------

'Déterminr le nombre de ressources du projet

    Dim Tableau_R As Range

    With Worksheets("Ordonnancement-" & Phase).Range("F4")
        Set Tableau_R = Range(.Offset(0, 0), .End(xlToRight))
    End With

    NbR = Tableau_R.Columns.count

    ReDim R_dates_en_cours(NbR) As Integer

    ReDim r(nbactivites, NbR)
    ReDim R_limite(NbR)

    With Worksheets("Ordonnancement-" & Phase).Range("A6")
        For j = 1 To nbactivites
            For k = 1 To NbR
                r(j, k) = DonneesActivites(j, 5 + k)
            Next
        Next

        For k = 1 To NbR
                R_limite(k) = .Offset(-2, 4 + k).Value
        Next
    End With

End Sub
Sub Construction_instance_Periode()

'Dates de début de la période
debut_periode = longueur * (Periode - 1)
fin_periode = longueur * Periode
s_jalon_bidon = fin_periode + 7


'Déterminer les activiés qui sont dans la période et juste après la période
nbdans_periode = nbactivites
nbapres_periode = 0
For j = 1 To nbactivites
    dans_periode(j) = j
    deplacable(j) = True
Next

'Considérer que toutes les activitées sont finient
```

```
For j = 1 To nbactivites
   j_fini(j) = True
Next

'Initialiser le nombre de retards des activités
ReDim nb_Retards_dans(nbdans_periode)

nbtous_periode = nbdans_periode + nbapres_periode

ReDim nb_retards(nbtous_periode)

'Créer un vecteur avec toutes les activitées dans et après la période
ReDim tous_periode(nbtous_periode)

End Sub
Sub Distribution_Beta__Duree_Reelle(activite As Integer)
'Génère une durée aléatoire suivant la loi Betâ

Dim probabilite As Double

   If var(activite) > 0 Then
            Select Case var(activite)
               Case 1     'Low
                        Mind = 0.75 * d(activite)
                        Maxd = 1.625 * d(activite)
               Case 2     'Medium
                        Mind = 0.5 * d(activite)
                        Maxd = 2.25 * d(activite)
               Case 3      'High
                        Mind = 0.25 * d(activite)
                        Maxd = 2.875 * d(activite)
            End Select

      probabilite = Rnd()
      If probabilite = 0 Then
         probabilite = 0.5
      End If
      d_realise(activite) = Int(Application.WorksheetFunction.BetaInv(probabilite, alpha, beta,
Mind, Maxd))
   Else
      d_realise(activite) = d(activite)
   End If

End Sub
Sub Politique__RP_SGS()
'REF: Van de Vonder (2007), "Heuristic procedures for reactive project scheduling".
'Dès qu'il y a possibilité de commencer une activité,
```

```
' l'activité choisie est celle ayant la date de début prévue la plus petite,
' dont la date_prévu <= date_en_court et dont les prédécesseurs de base sont finies.
'Toutes les activitées dans la periode doivent être commencée avant de commencer une activité après la période.

Dim Peut_commencer As Boolean, Looper As Boolean

'Commencer les activitées dont la date de début prévu est le premier jour de la simulation
'Mettre la date de début réalisé =-1 pour les activitées qui ne sont pas commencées
For a = 1 To nbdans_periode
      If s_prevu(dans_periode(a)) = Date_en_cours Then
         s_realise(dans_periode(a)) = s_prevu(dans_periode(a))
      Else
         s_realise(dans_periode(a)) = -1
      End If
Next

For a = 1 To nbapres_periode
   s_realise(apres_periode(a)) = -1
Next

'Dans la période
Do

   'Calculer combien il y a de ressources utilisées par les activitées en cours
   For k = 1 To NbR              'Initialisation
      R_dates_en_cours(k) = 0
   Next

   For a = 1 To nbdans_periode     'Calcul
      If s_realise(dans_periode(a)) <> -1 And j_fini(dans_periode(a)) = False Then

         'Regarder si l'activité est fini
         If s_realise(dans_periode(a)) + d_realise(dans_periode(a)) <= Date_en_cours Then
            j_fini(dans_periode(a)) = True
         Else 'Si elle n'est pas fini, compter les ressources utilisées
            For k = 1 To NbR
               R_dates_en_cours(k) = R_dates_en_cours(k) + r(dans_periode(a), k)
            Next
         End If
      End If
   Next

   For k = 1 To NbR              'Vérification
      If R_dates_en_cours(k) > R_limite(k) Then
         MsgBox ("La limite de ressources est dépassée au jour: " & Date_en_cours)
      End If
```

```
    Next

    'Commencer les activitées qui sont possibles de commencer
    For a = 1 To nbdans_periode
        If s_realise(dans_periode(a)) = -1 Then
            j = dans_periode(a)
            Peut_commencer = True

            'Vérifier si la date de début prévue est atteinte
            If s_prevu(j) > Date_en_cours Then
                Peut_commencer = False
                GoTo line2
            End If

            'Vérifier si les prédécesseurs de base sont finient
            For i = 1 To NbPredBase(j)
                If j_fini(PredBase(j, i) + 1) = False Then
                    Peut_commencer = False
                    GoTo line2
                End If
            Next

            'Vérifier s'il y a assez de ressources
            For k = 1 To NbR
                If R_dates_en_cours(k) + r(j, k) > R_limite(k) Then
                    Peut_commencer = False
                    GoTo line2
                End If
            Next

            'Commencer l'activitée si c'est possible
            If Peut_commencer = True Then
                s_realise(j) = Date_en_cours
                For k = 1 To NbR
                    R_dates_en_cours(k) = R_dates_en_cours(k) + r(j, k)
                Next
            End If
        End If
line2:
    Next

    'Passer à la journée suivante
    Date_en_cours = Date_en_cours + 1

    'Regarder s'il reste des activitées à commencer
    Looper = False
    For a = 1 To nbdans_periode
```

```
    If j_fini(dans_periode(a)) = False Then
       Looper = True
    End If
  Next

Loop While Looper = True And Date_en_cours < fin_periode 'Répéter tant qu'il reste des activitées à effectuer et qu'on est dans la phase

'Après la période
Do

  'Calculer combien il y a de ressources utilisées par les activitées en cours
  For k = 1 To NbR              'Initialisation
    R_dates_en_cours(k) = 0
  Next

  For a = 1 To nbdans_periode     'Calcul dans Période
    If s_realise(dans_periode(a)) <> -1 And j_fini(dans_periode(a)) = False Then

      'Regarder si l'activité est fini
      If s_realise(dans_periode(a)) + d_realise(dans_periode(a)) <= Date_en_cours Then
        j_fini(dans_periode(a)) = True
      Else 'Si elle n'est pas fini, compter les ressources utilisées
        For k = 1 To NbR
          R_dates_en_cours(k) = R_dates_en_cours(k) + r(dans_periode(a), k)
        Next
      End If
    End If
  Next

  For a = 1 To nbapres_periode     'Calcul après Période
    If s_realise(apres_periode(a)) <> -1 And j_fini(apres_periode(a)) = False Then

      'Regarder si l'activité est fini
      If s_realise(apres_periode(a)) + d_realise(apres_periode(a)) <= Date_en_cours Then
        j_fini(apres_periode(a)) = True
      Else 'Si elle n'est pas fini, compter les ressources utilisées
        For k = 1 To NbR
          R_dates_en_cours(k) = R_dates_en_cours(k) + r(apres_periode(a), k)
        Next
      End If
    End If
  Next

  For k = 1 To NbR              'Vérification
    If R_dates_en_cours(k) > R_limite(k) Then
      MsgBox ("La limite de ressources est dépassée au jour: " & Date_en_cours)
```

```
        End If
    Next

    'Commencer les activitées qui sont dans la période et qui sont possibles de commencer
    For a = 1 To nbdans_periode
        If s_realise(dans_periode(a)) = -1 Then
            j = dans_periode(a)
            Peut_commencer = True

            'Vérifier si la date de début prévue est atteinte
            If s_prevu(j) > Date_en_cours Then
                Peut_commencer = False
                GoTo line3
            End If

            'Vérifier si les prédécesseurs de base sont finient
            For i = 1 To NbPredBase(j)
                If j_fini(PredBase(j, i) + 1) = False Then
                    Peut_commencer = False
                    GoTo line3
                End If
            Next

            'Vérifier s'il y a assez de ressources
            For k = 1 To NbR
                If R_dates_en_cours(k) + r(j, k) > R_limite(k) Then
                    Peut_commencer = False
                    GoTo line3
                End If
            Next

            'Commencer l'activitée si c'est possible
            If Peut_commencer = True Then
                s_realise(j) = Date_en_cours
                For k = 1 To NbR
                    R_dates_en_cours(k) = R_dates_en_cours(k) + r(j, k)
                Next
            End If
        End If
line3:
    Next

    'Commencer les activitées qui sont apres la période et qui sont possibles de commencer
    For a = 1 To nbapres_periode
        If s_realise(apres_periode(a)) = -1 Then
            j = apres_periode(a)
            Peut_commencer = True
```

```
        'Vérifier si la date de début prévue est atteinte
        If s_prevu(j) > Date_en_cours Then
           Peut_commencer = False
           GoTo line4
        End If

        'Vérifier si les prédécesseurs de base sont finient
        For i = 1 To NbPredBase(j)
           If j_fini(PredBase(j, i) + 1) = False Then
              Peut_commencer = False
              GoTo line4
           End If
        Next

        'Vérifier s'il y a assez de ressources
        For k = 1 To NbR
           If R_dates_en_cours(k) + r(j, k) > R_limite(k) Then
              Peut_commencer = False
              GoTo line4
           End If
        Next

        'Commencer l'activitée si c'est possible
        If Peut_commencer = True Then
           s_realise(j) = Date_en_cours
           For k = 1 To NbR
              R_dates_en_cours(k) = R_dates_en_cours(k) + r(j, k)
           Next
        End If
     End If
line4:
  Next

  'Passer à la journée suivante
  Date_en_cours = Date_en_cours + 1

  'Regarder s'il reste des activitées à commencer
  Looper = False
  For a = 1 To nbapres_periode
     If j_fini(apres_periode(a)) = False Then
        Looper = True
     End If
  Next
  If Looper = False Then
     For a = 1 To nbdans_periode
        If j_fini(dans_periode(a)) = False Then
```

```
            Looper = True
        End If
    Next
  End If

Loop While Looper = True 'Répéter tant qu'il reste des activitées à effectuer
End Sub
Sub Calculer_Cout_Simulation()


  cout_periode = 0
'Calculer le coût de cette réalisation
  For a = 1 To nbdans_periode
    If s_realise(dans_periode(a)) - s_prevu(dans_periode(a)) > 0 Then
      nb_Retards_dans(a) = nb_Retards_dans(a) + 1
      cout_periode = cout_periode + w(dans_periode(a)) * (s_realise(dans_periode(a)) -
s_prevu(dans_periode(a)))
    End If
  Next
  'Activitées après période
  For a = 1 To nbapres_periode
    If s_realise(apres_periode(a)) - s_prevu(apres_periode(a)) > 0 Then
      nb_Retards_apres(a) = nb_Retards_apres(a) + 1
      cout_periode = cout_periode + w(apres_periode(a)) * (s_realise(apres_periode(a)) -
s_prevu(apres_periode(a)))
    End If
  Next

cout_simulations = cout_simulations + cout_periode
esperance_fin = esperance_fin + s_realise(nbactivites)

End Sub
Sub Tri_Croissant_EBST1()
'Trier les activités dans la période en ordre croissant de leur s_prevu.
'Si égalité, prendre l'activité avec le plus grand w.

  Dim temporaire, t As Integer
  Dim permut As Boolean

For a = 2 To nbdans_periode
  If s_prevu(dans_periode(a)) < s_prevu(dans_periode(a - 1)) Or (s_prevu(dans_periode(a)) =
s_prevu(dans_periode(a - 1)) And w(dans_periode(a)) > w(dans_periode(a - 1))) Then
      i = a + 1
      temporaire = dans_periode(a)
      permut = False
      t = 0
      Do
```

```
            i = i - 1
            dans_periode(i) = dans_periode(i - 1)
            If i = 2 Then
               permut = True
               t = 1
            End If
         Loop While (s_prevu(temporaire) < s_prevu(dans_periode(i - 2 + t)) Or
(s_prevu(temporaire) = s_prevu(dans_periode(i - 2 + t)) And w(temporaire) > w(dans_periode(i -
2 + t)))) And permut = False
         dans_periode(i - 1) = temporaire
   End If
Next

For a = 2 To nbapres_periode
   If s_prevu(apres_periode(a)) < s_prevu(apres_periode(a - 1)) Or (s_prevu(apres_periode(a)) =
s_prevu(apres_periode(a - 1)) And w(apres_periode(a)) > w(apres_periode(a - 1))) Then
         i = a + 1
         temporaire = apres_periode(a)
         permut = False
         t = 0
         Do
            i = i - 1
            apres_periode(i) = apres_periode(i - 1)
            If i = 2 Then
               permut = True
               t = 1
            End If
         Loop While (s_prevu(temporaire) < s_prevu(apres_periode(i - 2 + t)) Or
(s_prevu(temporaire) = s_prevu(apres_periode(i - 2 + t)) And w(temporaire) > w(dans_periode(i
- 2 + t)))) And permut = False
         apres_periode(i - 1) = temporaire
   End If
Next

End Sub
```

ANNEXE C – EXEMPLE NUMÉRIQUE POUR LES ALGORITHMES

Cette annexe présente un exemple numérique de l'application de la méthode Genflow ainsi que de la méthode STC. Les instances déterministes ont été choisies pour faciliter la compréhension. Les paramètres pour le cas stochastique sont les mêmes que ceux présentés à la section 3.1.

Algorithme *Genflow* et *Genflow modifié*

L'application de l'algorithme de génération du réseau du flux des ressources RFN ainsi que la différence entre la méthode originale et la méthode modifiée sont expliquées à l'aide d'un exemple numérique. La figure C.1 montre les activités ainsi que les contraintes de précédence.

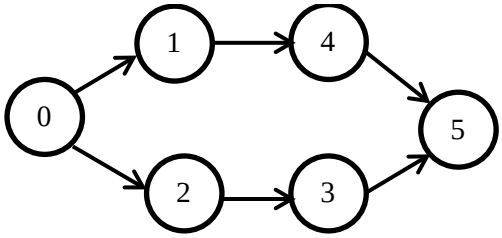

Figure C.1 : Réseau des activités

La durée des activités ainsi que le nombre de ressource requis par chaque activité se trouvent dans le tableau C.1. Pour des fins de simplification, il n'y a qu'un seul type de ressource.

Tableau C.1 : Caractéristiques des activités

Activité	Durée	Nombre de ressource
0	0	0
1	5	1
2	5	3
3	5	2
4	5	2
5	0	0

La figure C.2 montre le réseau de flux des ressources obtenu par l'algorithme *Genflow* original alors que la figure C.3 montre le flux des ressources obtenu par l'algorithme *Genflow modifié*. Les flèches pleines représentent les contraintes $\in A$ alors que les flèches pointillées représentent les contraintes $\in A_k$.

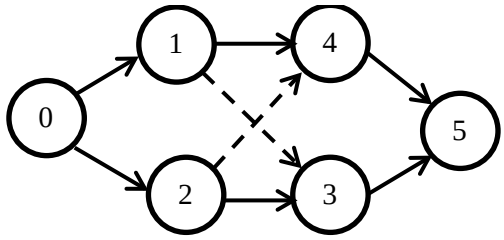

Figure C.2 : RFN obtenu par l'algorithme *Genflow*

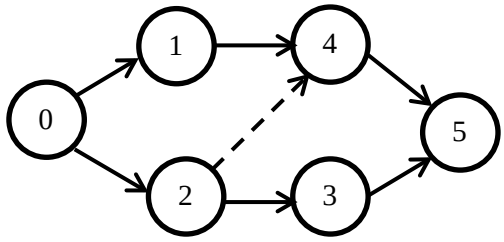

Figure C.3 : RFN obtenu par l'algorithme *Genflow modifié*

Il est possible de constater qu'il y a une contrainte de moins dans le RFN obtenu par l'algorithme modifié. L'étape 4 de l'algorithme, pour l'activité 3, est détaillée afin de montrer la différence entre les deux algorithmes. Au début de cette étape, le RFN courant est celui de la figure C.4, pour les deux méthodes. L'étape décrite permet d'ajouter l'activité 3 au RFN.

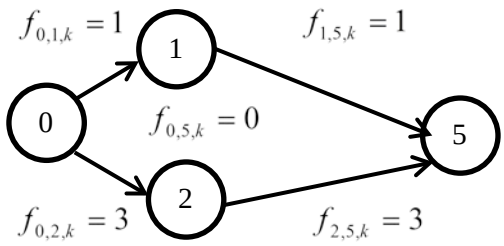

Figure C.4 : RFN au début de l'étape 4 pour l'activité 3

Algorithme *Genflow*

4. Pour l'activité 3 faire :

a. Pour la ressource k, poser $req_k = r_{3k} = 2$

b. Pour la ressource k faire :

$V_3 = \{0,1,2\}$

i. Pour $0 \in V_3$ faire :

1. $q = \min(req_k, f_{0,5,k}) = \min(2,0) = 0$
2. $req_k = req_k - q = 2 - 0 = 2$
3. $f_{0,5,k} = f_{0,5,k} - q = 0 - 0 = 0$
4. $f_{0,3,k} = f_{0,3,k} + q = 0 + 0 = 0$

i. Pour $1 \in V_3$ faire :

1. $q = \min(req_k, f_{1,5,k}) = \min(2,1) = 1$
2. $req_k = req_k - q = 2 - 1 = 1$
3. $f_{1,5,k} = f_{1,5,k} - q = 1 - 1 = 0$
4. $f_{1,3,k} = f_{1,3,k} + q = 0 + 1 = 1$

i. Pour $2 \in V_3$ faire :

1. $q = \min(req_k, f_{2,5,k}) = \min(1,3) = 1$
2. $req_k = req_k - q = 1 - 1 = 0$
3. $f_{2,5,k} = f_{2,5,k} - q = 3 - 1 = 2$
4. $f_{2,3,k} = f_{2,3,k} + q = 0 + 1 = 1$

c. Pour la ressource k, poser $f_{3,5,k} = r_{3k} = 2$

Algorithme *Genflow modifié*

4. Pour l'activité 3 faire :

a. Pour la ressource k, poser $req_k = r_{3k} = 2$

b. Pour la ressource k faire :

$Pred_3 = \{2\}$

i. Pour $2 \in Pred_3$ faire :

1. $q = \min(req_k, f_{2,5,k}) = \min(2,3) = 2$
2. $req_k = req_k - q = 2 - 2 = 0$
3. $f_{2,5,k} = f_{2,5,k} - q = 3 - 2 = 1$
4. $f_{2,3,k} = f_{2,3,k} + q = 0 + 2 = 2$

ii. $req_k = 0$, donc cette étape n'est pas faite

c. Pour la ressource k, poser $f_{3,5,k} = r_{3k} = 2$

Ainsi, il est possible d'éviter d'ajouter des contraintes au réseau en considérant en premier les ressources des prédécesseurs.

Heuristique STC, version originale et modifié

L'application de l'heuristique STC ainsi que la différence entre la méthode originale et la méthode modifiée sont expliquées à l'aide d'un exemple. Le tableau C.2 présente les caractéristiques des activités. Pour des fins de simplification, il n'y a qu'un seul type de ressource.

Tableau C.2 : Caractéristiques des activités

Activité	Durée	Nombre de ressource	Date de début prévue	Poids	Variance
0	0	0	0	0	0
1	6	1	0	0	3
2	2	1	0	0	3
3	2	1	2	1	2
4	2	2	6	1	1
5	2	2	8	8	1
6	0	0	10	38	0

La figure C.5 montre le diagramme de Gantt pour l'ordonnancement initial déterministe alors que la figure C.6 montre le réseau des flux de ressource RFN.

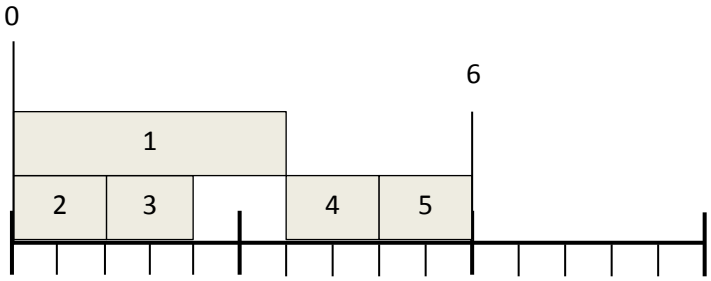

Figure C.5 : Diagramme de Gantt de l'ordonnancement initial déterministe

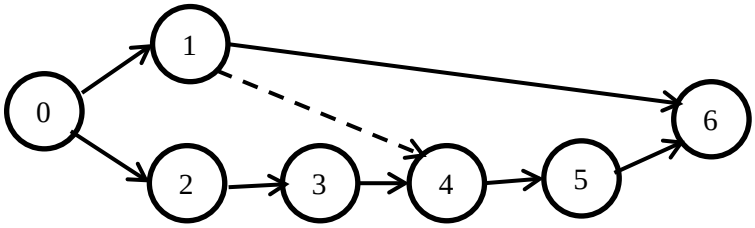

Figure C.6 : RFN

La date de fin est reculée de trois jours $\delta = 13$, afin de permettre l'ajout de plusieurs tampon de temps permettant de bien expliquer la méthode. La figure C.7 montre l'ordonnancement au début de l'heuristique STC.

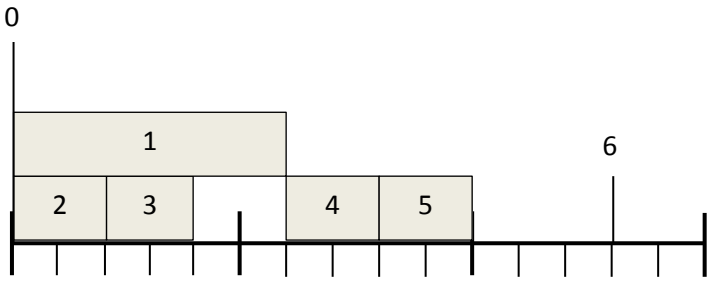

Figure C.7 : Diagramme de Gantt de l'ordonnancement initial stochastique

Algorithme STC

1. Poser tous les tampons de temps comme étant 0
2. Calculer les stc_j de chaque activité. Le calcul pour l'activité 4 est détaillé. Les autres valeurs se trouvent dans le tableau C.3.

$$stc_j = w_j \sum P\left(s_j < s_i + \mathbf{d}_i + LPL(i,j)\right)$$

$$stc_4 = w_4 \begin{pmatrix} P\left(s_4 < s_0 + \mathbf{d}_0 + LPL(0,4)\right) + \\ P\left(s_4 < s_1 + \mathbf{d}_1 + LPL(1,4)\right) + \\ P\left(s_4 < s_2 + \mathbf{d}_2 + LPL(2,4)\right) + \\ P\left(s_4 < s_3 + \mathbf{d}_3 + LPL(3,4)\right) \end{pmatrix}$$

$$= 1 \begin{pmatrix} P\left(6 < 0 + \mathbf{d}_0 + 4\right) + \\ P\left(6 < 0 + \mathbf{d}_1 + 0\right) + \\ P\left(6 < 0 + \mathbf{d}_2 + 2\right) + \\ P\left(6 < 2 + \mathbf{d}_3 + 0\right) \end{pmatrix}$$

$$= 1 \begin{pmatrix} P\left(2 < \mathbf{d}_0\right) + \\ P\left(6 < \mathbf{d}_1\right) + \\ P\left(4 < \mathbf{d}_2\right) + \\ P\left(4 < \mathbf{d}_3\right) \end{pmatrix}$$

$$= 1 \begin{pmatrix} 0 + \\ 0{,}131 + \\ 0{,}014 + \\ 0{,}002 \end{pmatrix}$$

$$= 0{,}147$$

Tableau C.3 : Somme des STC et stc_j

Activité	stc_j
0	0
1	0
2	0
3	0,320
4	0,147
5	3,818
6	2,155
Somme	6,440

3. Trie des activités en ordre décroissant de leur stc_j : 5,6,3,4,1,2,0
4. Tant qu'il n'y a pas d'amélioration, faire :
    a. L'activité 5 est prise
    b. $stc_5 > 0$
    c. Continuer :
        i. $buff_5 = buff_5 + 1 = 0 + 1 = 1$
        ii. $s_5 = s_5 + 1 = 8 + 1 = 9$

            Il y a une marge libre entre l'activité 5 et l'activité 6, alors l'activité 6 n'a pas à être reculée.
        iii. Tableau C.4 : Somme des STC et stc'_j

Activité	stc'_j
0	0
1	0
2	0
3	0,320
4	0,147
5	0,902
6	2,155
Somme	3,324

        iv. $\sum_{j \in N} stc'_j < \sum_{j \in N} stc_j$ et la date de fin n'a pas été reculée

            1. Les nouvelles dates sont enregistrées

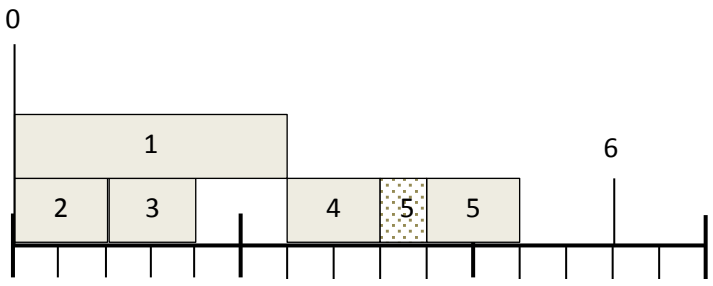

Figure C.8 : Diagramme de Gantt de l'ordonnancement courant

            2. Retour à l'étape 3

3. Trie des activités en ordre décroissant de leur stc_j : 6,5,3,4,1,2,0

4. Tant qu'il n'y a pas d'amélioration, faire :

a. L'activité 6 est prise

b. $stc_6 > 0$

c. La date de fin ne peut pas être reculée, alors il n'est pas possible d'ajouter un tampon de temps devant l'activité factice de fin.

a. L'activité 5 est prise

b. $stc_5 > 0$

c. L'ajout d'un tampon de temps devant l'activité 5 diminue la somme des STC sans reculer la date de fin. L'ordonnancement mis à jour est montré à la figure C.9.

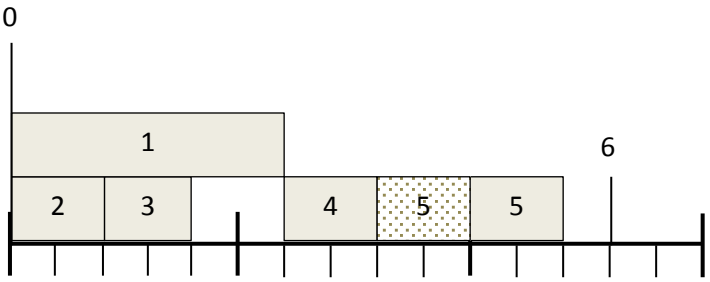

Figure C.9 : Diagramme de Gantt de l'ordonnancement courant

3. Trie des activités en ordre décroissant de leur stc_j : 6,5,3,4,1,2,0

4. Tant qu'il n'y a pas d'amélioration, faire :

a. L'activité 6 est prise

b. $stc_6 > 0$

c. La date de fin ne peut pas être reculée, alors il n'est pas possible d'ajouter un tampon de temps devant l'activité factice de fin.

a. L'activité 5 est prise

b. $stc_5 > 0$

c. Continuer :

i. $buff_5 = buff_5 + 1 = 2 + 1 = 3$

ii. $s_5 = s_5 + 1 = 10 + 1 = 11$

Il y a une marge libre entre l'activité 5 et l'activité 6, alors l'activité 6 n'a pas à être reculée.

iii. Tableau C.5 : Somme des STC et stc'_j

Activité	stc'_j
0	0
1	0
2	0
3	0,320
4	0,147
5	0,454
6	14,713
Somme	15,634

iv. $\sum_{j \in N} stc'_j > \sum_{j \in N} stc_j$, donc ce n'est pas une amélioration

v. Comme ce n'est pas une amélioration :

1. $buff_5 = buff_5 - 1 = 3 - 1 = 2$
2. $s_5 = 10$

a. L'activité 3 est prise

b. $stc_3 > 0$

c. L'ajout d'un tampon de temps devant l'activité 3 diminue la somme des STC sans reculer la date de fin. L'ordonnancement mis à jour est montré à la figure C.10.

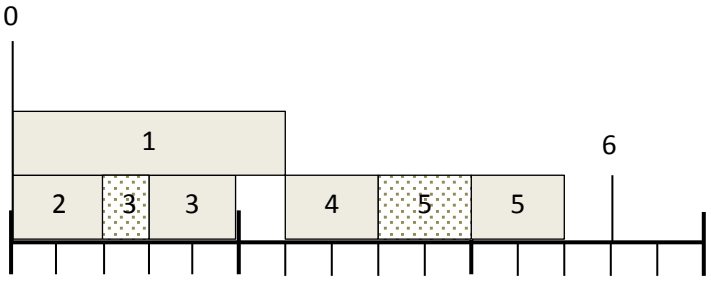

Figure C.10 : Diagramme de Gantt de l'ordonnancement courant

3. Trie des activités en ordre décroissant de leur stc_j : 6,5,4,3,1,2,0

4. Tant qu'il n'y a pas d'amélioration, faire :

a. L'activité 6 est prise

b. $stc_6 > 0$

c. La date de fin ne peut pas être reculée, alors il n'est pas possible d'ajouter un tampon de temps devant l'activité factice de fin.

a. L'activité 5 est prise

b. $stc_5 > 0$

c. L'ajout d'un tampon de temps devant l'activité 5 augmente la somme des STC, alors le déplacement est rejeté.

a. L'activité 4 est prise

b. $stc_4 > 0$

c. Continuer :

i. $buff_4 = buff_4 + 1 = 0 + 1 = 1$

ii. $s_4 = s_4 + 1 = 6 + 1 = 7$

Il n' a pas de marge entre l'activité 4 et l'activité 5 alors la date de début de l'activité 5 est mise à jour.

Selon la méthode originale :

Dans la version originale, les successeurs sont mis à jour sans changer la durée de leur tampon de temps. Ainsi, l'activité 5 est reculée.

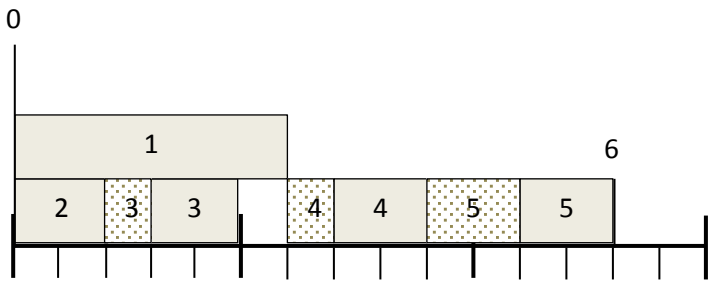

Figure C.11 : Diagramme de Gantt de l'ordonnancement courant, selon la méthode originale

Selon la méthode modifiée :

Dans la version modifiée, s'il y a un tampon de temps devant l'activité, celui-ci est diminué et la date de début de l'activité reste inchangée. Ainsi, l'activité 5 ne bouge pas et $buff_5 = buff_5 - 1$.

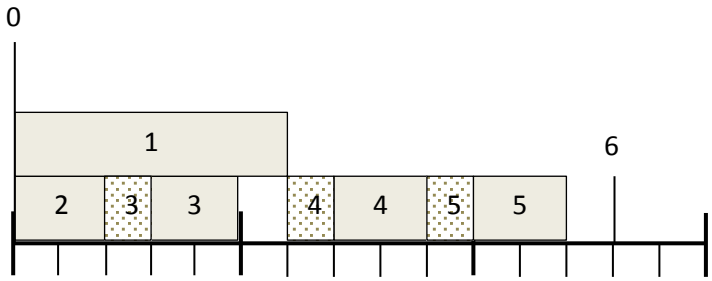

Figure C.12 : Diagramme de Gantt de l'ordonnancement courant, selon la méthode modifiée

iii. Tableau C.6 : Somme des STC et stc'_j, **selon la méthode originale**

Activité	stc'_j
0	0
1	0
2	0
3	0,110
4	0,108
5	0,454
6	14,713
Somme	15,385

Tableau C.7 : Somme des STC et stc'_j, **selon la méthode modifié**

Activité	stc'_j
0	0
1	0
2	0
3	0,110
4	0,108
5	0,645
6	2,155
Somme	3,019

iv. **Méthode originale :** $\sum_{j \in N} stc'_j > \sum_{j \in N} stc_j$, ce n'est pas une amélioration.

Méthode modifié : $\sum_{j \in N} stc'_j < \sum_{j \in N} stc_j$ et la date de fin n'est pas reculée.

1. Les nouvelles dates sont enregistrées
2. Retour à l'étape 3

v. **Méthode originale :** Comme ce n'est pas une amélioration, le déplacement n'est pas accepté.

1. $buff_4 = buff_4 - 1 = 1 - 1 = 0$
2. $s_4 = 6$ et $s_5 = 10$

Suite et fin de l'algorithme selon la méthode originale :

a. L'activité 3 est prise

b. $stc_3 > 0$

c. Continuer :

i. $buff_3 = buff_3 + 1 = 1 + 1 = 2$

ii. $s_3 = s_3 + 1 = 3 + 1 = 4$

Il y a une marge libre entre l'activité 3 et l'activité 4, alors l'activité 4 n'a pas à être reculée.

iii. Tableau C.8 : Somme des STC et stc'_j

Activité	stc'_j
0	0
1	0
2	0
3	0,014
4	0,574
5	0,645
6	2,155
Somme	3,389

iv. $\sum_{j \in N} stc_j^{'} = 3{,}389 > \sum_{j \in N} stc_j = 3{,}324$, donc ce n'est pas une amélioration

v. Comme ce n'est pas une amélioration :

3. $buff_3 = buff_3 - 1 = 2 - 1 = 1$

4. $s_3 = 3$

a. L'activité 1 est prise

b. $stc_1 = 0$, FIN

Suite et fin de l'algorithme selon la méthode modifiée :

3. Trie des activités en ordre décroissant de leur stc_j : 6,5,3,4,1,2,0

4. Tant qu'il n'y a pas d'amélioration, faire :

a. L'activité 6 est prise

b. $stc_6 > 0$

c. La date de fin ne peut pas être reculée, alors il n'est pas possible d'ajouter un tampon de temps devant l'activité factice de fin.

a. L'activité 5 est prise

b. $stc_5 > 0$

c. L'ajout d'un tampon de temps devant l'activité 5 augmente la somme des STC, alors le déplacement est rejeté.

a. L'activité 3 est prise

b. $stc_3 > 0$

c. Continuer :

i. $buff_3 = buff_3 + 1 = 1 + 1 = 2$

ii. $s_3 = s_3 + 1 = 3 + 1 = 4$

Il y a une marge libre entre l'activité 3 et l'activité 4, alors l'activité 4 n'est pas déplacée.

iii. Tableau C.9 : Somme des STC et stc'_j

Activité	stc'_j
0	0
1	0
2	0
3	0,014
4	0,199
5	0,645
6	2,155
Somme	3,014

iv. $\sum_{j \in N} stc'_j = 3{,}014 < \sum_{j \in N} stc_j = 3{,}019$ et la date de fin n'a pas été reculée

1. Les nouvelles dates sont enregistrées

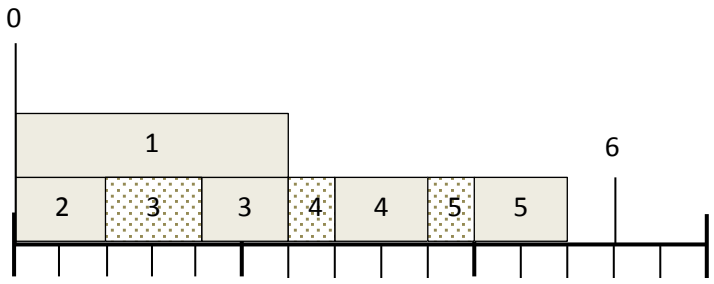

Figure C.13 : Diagramme de Gantt de l'ordonnancement courant

2. Retour à l'étape 3

3. Trie des activités en ordre décroissant de leur stc_j : 6,5,4,3,1,2,0

4. Tant qu'il n'y a pas d'amélioration, faire :

a. L'activité 6 est prise

b. $stc_6 > 0$

c. La date de fin ne peut pas être reculée, alors il n'est pas possible d'ajouter un tampon de temps devant l'activité factice de fin.

a. L'activité 5 est prise

b. $stc_5 > 0$

c. L'ajout d'un tampon de temps devant l'activité 5 augmente la somme des STC, alors le déplacement est rejeté.

a. L'activité 4 est prise

b. $stc_4 > 0$

c. L'ajout d'un tampon de temps devant l'activité 4 augmente la somme des STC, alors le déplacement est rejeté.

a. L'activité 3 est prise

b. $stc_3 > 0$

c. L'ajout d'un tampon de temps devant l'activité 3 augmente la somme des STC, alors le déplacement est rejeté.

a. L'activité 1 est prise

b. $stc_1 = 0$, FIN

La solution finale obtenue par la méthode STC modifié est la suivante :

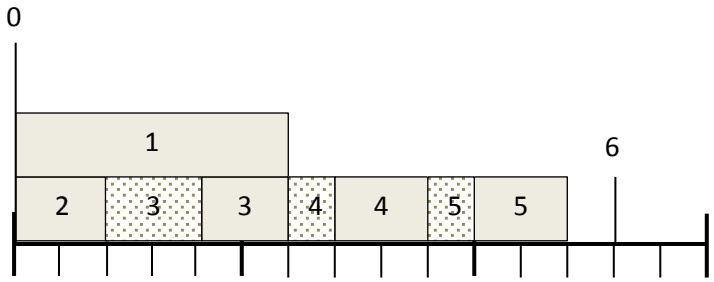

Figure C.14 : Diagramme de Gantt de l'ordonnancement final

La solution finale obtenue par la méthode STC originale est la suivante :

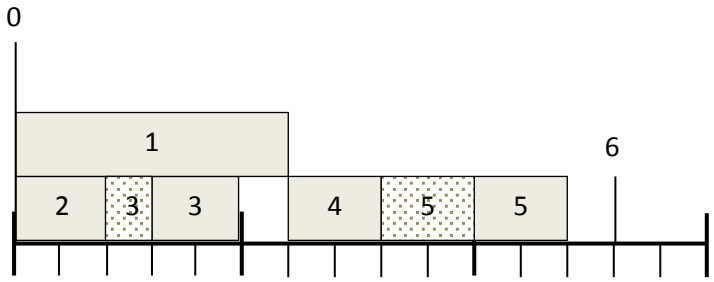

Figure C.15 : Diagramme de Gantt de l'ordonnancement final

Ainsi, le fait de réduire le tampon de temps d'un successeur permet de visiter plus de solutions et donc de trouver une solution plus robuste.