Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique
UFR STMIA

École doctorale IAEM Lorraine

Outils et algorithmes pour gérer l'incertitude lors de l'ordonnancement d'application sur plateformes distribuées

THÈSE

date de soutenance envisagée le 18 octobre 2010

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Louis-Claude Canon

Composition du jury

Rapporteurs :	Bruno Gaujal, Directeur de recherche, Laboratoire d'Informatique de Grenoble
	Pierre Sens, Professeur, Université de Paris 6
Examinateurs :	Emmanuel Jeannot, Chargé de recherche, INRIA-Bordeaux (Directeur de thèse)
	Arnold Rosenberg, Professeur, Université du Colorado
	René Schott, Professeur, Université Henri Poincaré

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Ce mémoire de thèse représente l'aboutissement de trois années de recherche, de découvertes et d'égarement. Derrière les contributions scientifiques qui y sont présentées, se dissimule un investissement humain qu'il m'aurait été impossible d'assumer sans la direction éclairée de mon directeur de thèse. Je lui exprime donc ma plus vive reconnaissance pour son encadrement d'exception. Je tiens en particulier à souligner sa disponibilité, la pertinence de ses critiques ainsi que son soutien et l'en remercie sincèrement.

Je souhaiterai également adresser un remerciement particulier à mes parents qui ont eu un rôle significatif dans l'élaboration de ce document. Leur intérêt pour cette réalisation m'a spécialement touché.

Mes derniers remerciements vont à l'ensemble des personnes que j'ai côtoyé pendant ces dernières années, ce qui inclut les membres des équipes AlGorille et Runtime qui m'ont accueilli pendant ma thèse, les membres de ma famille et enfin mes amis.

Table des matières

Introduction

Cette thèse traite de l'ordonnancement dans les systèmes distribués. Notre objectif est d'étudier l'impact de l'incertitude sur les ordonnancements et de proposer des techniques pour en réduire les effets sur les critères à optimiser.

Dans cette introduction, nous présentons le contexte du calcul distribué, quelques concepts relatifs à l'incertitude et le domaine d'étude lié à l'ordonnancement. Nous poursuivons en présentant le plan et les principaux axes autour desquelles s'articule cette thèse, ainsi que les collaborations que nous avons menées.

Contexte

Calcul distribué

Une plateforme de calcul distribué est composée d'un ensemble de processeurs (ou plus généralement de ressources calculatoires) interconnectés par un réseau de communication. Comme ces systèmes sont complexes, leur étude requiert des modèles puissants et utiles. Si les ressources de calcul sont séquentielles, un modèle couramment utilisé est l'architecture de type SISD (Single Instruction, Single Data) de la taxonomie de Flynn [67], ce qui correspond à l'architecture de Von Neumann. Les modèles des réseaux de communication ont quant à eux considérablement évolués depuis le modèle PRAM [110] avec par exemple le modèle un-port [22] ou encore les modèles de communications point à point comme LogP [46].

Du coté logiciel, un modèle d'application spécifie l'ensemble des calculs à réaliser. Une application peut se décomposer en plusieurs tâches, le calcul de chaque tâche produisant alors un résultat. Le calcul d'une tâche peut par ailleurs être conditionné par l'accessibilité et la disponibilité des données produites par d'autres tâches. La structure de l'application définit ainsi dans quel ordre les tâches peuvent être exécutées.

Nous nous intéressons à l'exécution efficace d'une application parallèle sur une plateforme de calcul distribué. À cet égard, l'ensemble des modèles de calcul distribué mentionnés plus haut sont déterministes. Or l'incertitude est une composante essentielle à toute science. En particulier, l'échelle et la complexité importante (et grandissante) des systèmes distribués sont des sources d'indétermination. Par exemple, la disponibilité d'une ressource particulière à un moment donné ou bien encore la nature précise des calculs parallèles ne sont pas toujours caractérisées. Ainsi, cette thèse consiste en l'étude systématique de l'incertitude dans les systèmes distribués.

Incertitude

Nous distinguons plusieurs aspects de l'incertitude en considérant successivement l'objet qu'elle touche, sa nature, puis son origine.

Relativement au calcul distribué, l'incertitude touche le déroulement du calcul des tâches et plus précisément les trois caractéristiques suivantes :

Durée des calculs Une ressource ne garantie pas à quelle date un calcul sera finalisé. Exemple : opération de maintenance automatique ayant lieu lors du calcul d'une tâche et retardant son exécution.

Succès des calculs Cette caractéristique est liée à la capacité d'une ressource calculatoire à délivrer un résultat. Exemple : un individu décide d'interrompre un calcul.

Exactitude des résultats Une ressource renvoie un résultat potentiellement incorrect pour une tâche donnée. Exemple : perturbation sur le réseau lors de la transmission d'un résultat.

Nous proposons par ailleurs une taxonomie sur la nature des incertitudes inspirée de celle exposée par Haimes [78] :

Méthodologique L'incertitude de ce type est liée aux limites des méthodes employées. La simplification des modèles est parfois une nécessité pour leur permettre d'être manipulable en pratique. Il arrive aussi que des hypothèses favorables soient utilisées en dehors des situations où elles sont justifiées. Exemple : modèle de communication ou d'exécution imparfait.

Épistémique Celle-ci se rapporte à l'ignorance portée sur les mécanismes en jeu dans un système donné. En d'autres termes, elle provient de notre inaptitude à connaître les interactions ayant lieu. Il peut s'agir d'une connaissance qui est incomplète et/ou inexacte. L'incertitude de ce type peut toucher des aspects connus et délimités ou bien être totale. Dans tous les cas, cela signifie que la modélisation parfaite d'un système est hors de portée. Exemple : soumission de tâches en ligne dans une machine parallèle.

Aléatoire L'incertitude de ce type concerne la variabilité aléatoire qui est inhérente aux phénomènes physiques. Bien que l'ensemble des paramètres soit déterminé par des probabilités, il est dans ce cas impossible de prédire précisément le résultat d'une expérience donnée. Exemple : panne matérielle.

Enfin, nous classons les problématiques posées en fonction de l'origine de l'incertitude :

Matérielle Les caractéristiques de la plateforme ne lui permettent pas de garantir tous les aspects de son mode de fonctionnement.

Logicielle La nature des applications complique la prédictibilité du système. Exemple : erreur logicielle entraînant l'échec d'un calcul donné.

Humaine Des individus malveillants, malhonnêtes ou imprévisibles sont présents sur une plateforme donnée. Exemple : utilisateur d'une grille de calcul.

Ordonnancement

Exécuter une application sur une plateforme de calcul distribué de manière efficace lève un problème d'ordonnancement. En toute généralité, l'ordonnancement est l'étape qui réalise une association ordonnée entre des requêtes (dans notre cas, des tâches) et des ressources (dans notre cas, des processeurs). Le problème posé possède donc deux dimensions : l'allocation des tâches aux processeurs ; et, l'agencement temporel de leurs calculs sur chaque processeur. L'objectif est de réaliser cette association de manière à optimiser des critères d'efficacité tout en respectant les contraintes définies.

Nous distinguons deux stratégies d'ordonnancement suivant le moment auquel chaque décision d'ordonnancement est prise. Un ordonnancement *statique* spécifie entièrement la façon dont les calculs se déroulent. Cette approche postule que toutes les informations liées aux tâches et aux processeurs sont connues à priori. Un ordonnancement *dynamique* définit des règles qui sont appliquées en cours d'exécution pour déterminer sur quel processeur calculer chaque tâche. Dans ce dernier cas, si une tâche est prête à être exécutée et qu'un processeur est libre à un instant

donné, alors une décision d'ordonnancement peut est prise. Ces deux approches sont parfois complémentaires, les règles dynamiques venant alors ajuster un ordonnancement statique en cas d'imprévu.

La notation $\alpha|\beta|\gamma$ [76] offre une manière concise d'exprimer les paramètres d'un problème d'ordonnancement. Le champ α dénote les caractéristiques de la plateforme de calcul comme le nombre de processeurs et son hétérogénéité. Les contraintes sont spécifiées à la place du symbole β. Par exemple, les tâches sont parfois soumises à des contraintes de dépendance, c'est-à-dire que l'exécution d'une tâche donnée ne peut commencer tant que les tâches dont celle-ci dépend n'ont pas fini les leurs. Enfin, le champ γ indique le critère à optimiser. Un exemple commun est la durée totale de l'ordonnancement, à savoir la période séparant le début de l'exécution de la première tâche et la fin de l'exécution de la dernière tâche.

Organisation du document

L'objectif de cette thèse est d'analyser un ensemble de problèmes d'ordonnancement en présence d'incertitude. De l'étude de ces problèmes, se dégagent deux problématiques qui sont indépendantes d'un problème d'ordonnancement spécifique et qui revêtent un caractère générique. Les deux problématiques générales font l'objet de la première partie. La seconde partie de la thèse concerne les problèmes d'ordonnancement eux-mêmes.

Méthodes génériques

La première partie présente des méthodes utiles pour la gestion de l'incertitude dans les problèmes d'ordonnancement. Nous postulons que l'incertitude enrichit l'ordonnancement d'une dimension probabiliste et d'une dimension multicritère qui font chacune l'objet d'un chapitre.

Le chapitre 1 présente comment estimer la distribution d'une variable aléatoire qui possède une représentation issue d'un graphe stochastique. Pour simplifier, le problème consiste à évaluer des opérations de maximum et de somme sur un ensemble de variables aléatoires. Deux heuristiques sont proposées et se montrent favorables à l'utilisation que nous visons.

Le chapitre 2 décrit l'état de l'art en terme d'optimisation multicritère. Quelques incréments de recherche sont par ailleurs réalisés et détaillés.

Nous pensons que ces outils probabilistes et multicritères sont pertinents et montrons la mise en pratique des résultats de ces deux premiers chapitres dans le chapitre 3 en seconde partie.

Problèmes d'ordonnancement

La seconde partie s'attaque à des problèmes d'ordonnancement connus dans la littérature qui sont revisités en y considérant l'incertitude.

Dans le chapitre 3, nous instancions un problème d'ordonnancement classique qui consiste à ordonnancer un graphe de tâches sur une plateforme hétérogène. Nous y spécifions que les durées d'exécution sont des variables aléatoires. Cela se motive par la complexité des applications et des plateformes. Les durées des calculs sont alors seulement estimées, ce qui induit une incertitude de type *méthodologique*. Un nouveau critère lié à la stabilité de l'ordonnancement intervient alors : la robustesse. La structure du problème est analysée et des méthodes d'ordonnancement multicritère sont proposées et empiriquement validées.

Le chapitre 4 se base aussi sur l'ordonnancement d'un graphe de tâches sur une plateforme hétérogène. Alors que les durées sont fixes, chaque calcul peut échouer suite à une panne matérielle. Bien qu'en pratique une telle panne ne survienne que très rarement sur un seul processeur, l'échelle croissante des systèmes distribués rend cet événement plus critique. Nous considérons

		Incertitude			Problème d'ordonnancement		
		Objet	Nature	Origine	Type	Critère	Notation $\alpha\|\beta\|\gamma$
Chapitre	3	durée des calculs	méthodologique	matérielle, logicielle	optimisation	robustesse	$R\|prec\|C_{\max}$
	4	succès des calculs	aléatoire	matérielle	évaluation	fiabilité	$R\|prec\|C_{\max}$
	5	exactitude des résultats	épistémique	logicielle, humaine	caractérisation	précision	$R\|online - time{-}nclv\|\sum C_i$

TABLE 1 – Résumé des modalités des chapitres de la seconde partie

que ces pannes sont par nature stochastiques et rentrent dans la catégorie de l'incertitude *aléatoire*. Déterminer la probabilité de succès de l'ordonnancement, autrement dit évaluer sa fiabilité, accapare l'essentiel du chapitre.

Bien que le sujet du chapitre 5 diffère de ceux des deux autres chapitres de cette partie, il élargit l'étude à un autre type d'incertitude. Nous y considérons un problème d'ordonnancement d'un ensemble de tâches indépendantes. L'exactitude de chaque résultat obtenu après exécution n'est pas garantie car les utilisateurs peuvent tricher et le système n'a pas accès à un mécanisme de vérification. Comme le système ignore quels processeurs sont susceptibles de retourner des résultats erronés, il est soumis à un incertitude *épistémique*. Le problème consiste donc à caractériser les propriétés de la plateforme. Cette caractérisation pourrait ultimement servir à un mécanisme qui certifierait les résultats corrects.

La seconde partie s'articule donc autour de plusieurs modalités qui sont résumées dans le tableau 1. D'abord, l'objet sur lequel s'applique l'incertitude diffère dans chaque cas. Ces objets déterminent directement les critères qui se rajoutent aux problèmes. Nous avons déjà remarqué que chaque type d'incertitude défini dans notre taxinomie est représenté. Plusieurs types de problèmes sont également considérés. Enfin, l'origine de l'incertitude varie dans une certaine mesure. La seconde partie aborde donc un ensemble de problèmes représentatifs d'un large spectre de caractéristiques.

Notons que les ordonnancements proposés dans les chapitres 3 et 4 sont statiques. Nous avançons trois principales raisons :

- Les méthodes d'ordonnancement dynamiques peuvent se baser sur un ordonnancement statique en l'ajustant au besoin. Les deux approches sont donc complémentaires.
- L'approche statique offre plus de liberté quant aux méthodes d'optimisation utilisables : elle ne se limite pas à des méthodes de construction gloutonne ou incrémentale.
- Les problèmes levés par l'ordonnancement dynamique se recouvrent avec ceux posés par l'ordonnancement statique. Comme l'analyse de stratégies dynamiques est souvent plus complexe, l'étude de techniques d'ordonnancement statiques constitue une première étape.

Par ailleurs, les complexités des problèmes que nous traitons sont défavorables. La majeure partie des résultats proposés sont donc heuristiques, c'est-à-dire que la qualité des résultats générés n'est pas garantie. C'est pourquoi, nous validons empiriquement nos approches par l'intermédiaire de simulations.

Collaborations

Quelques unes des contributions présentées dans le chapitre 1 sont décrites dans [9, 12]. Les résultats détaillés dans le chapitre 2 et le chapitre 3 sont présentés dans [2, 4, 5, 11]. Le chapitre 4 résulte d'une collaboration avec Anne BENOIT et Yves ROBERT dont les principaux éléments se

retrouvent dans [10]. Enfin, le chapitre 5 est le fruit d'un travail commun avec Jon Wiessman qui est publié dans [13].

La liste des articles publiés dans le cadre de cette thèse est donnée page 159.

Notations

Nous nous somme efforcés de rendre les notations cohérentes à travers les chapitres afin d'éviter les collisions. Cependant, le nombre d'objets mathématiques manipulés est large et certains symboles sont parfois réutilisés si leur usage est naturel. Nous avons donc choisi un compromis entre la facilité de lecture et le risque d'ambiguïté. La plupart du temps toutefois, les symboles ne sont redéfinis que dans le cadre d'une explication technique qui s'étend sur une section (voire une sous-section).

Par convention, nous utilisons une capitale pour dénoter un ensemble ou une variable aléatoire. Si les éléments d'un ensemble sont déjà représentés par une capitale, alors nous utilisons une notation calligraphique. En toute généralité, les tâches sont indicées par la lettre i et les processeurs par la lettre j. Les notations les plus communément rencontrées sont données dans le tableau 2.

Symbole	Définition
$T = \{t_i : i \in [1..n]\}$	Ensemble des tâches
n	Nombre de tâches ($n = \|T\|$)
$G = (T, E)$	Graphe orienté acyclique contenant les tâches et les contraintes de dépendance
$\mathrm{Pred}(t_i)$	Ensemble des prédécesseurs de la tâche t_i ($\mathrm{Pred}(t_i) \subset T$)
$P = \{p_j : j \in [1..m]\}$	Ensemble des processeurs
m	Nombre de processeurs ($m = \|P\|$)
w_i^j	Durée d'exécution de la tâche t_i sur le processeur p_j
$d_{ii'}^{jj'}$	Durée de la communication $(t_i, t_{i'}) \in E$ entre les tâches t_i et $t_{i'}$ exécutées respectivement sur les processeurs p_j et $p_{j'}$
S_i	Date du début d'exécution de la tâche t_i
C_i	Date de fin d'exécution de la tâche t_i
$C_{\max}$	Durée totale de l'ordonnancement (maximum des dates de fin C_i)

Table 2 – Notations générales de la thèse

Première partie

Outils

Chapitre 1

Évaluation de graphes orientés acycliques et stochastiques

Sommaire

1.1 Introduction

La gestion de l'incertitude dans les systèmes distribués apporte une dimension probabiliste aux problèmes étudiés. Par exemple, les problématiques traitées dans le chapitre 3 et le chapitre 5 de cette thèse induisent la manipulation arithmétique de variables aléatoires.

Dans ces deux chapitres, des opérations de maximum et de somme sont réalisées sur des variables aléatoires. Dans le chapitre 3, le problème est plus spécifique car ces opérations résultent de la structure d'une application modélisée par un graphe de tâches. En particulier, chaque durée est une variable aléatoire. Ainsi, dans un ordonnancement, la date de début d'une tâche est donnée par le maximum des dates de fin de tous ses prédécesseurs, et la date de fin d'une tâche est la somme de sa durée et de sa date de début. Comme toutes ces dates sont des variables aléatoires, le calcul d'une mesure de performance pour un ordonnancement donné requiert d'évaluer le résultat d'une expression comportant uniquement des opérations de maximum et de somme sur des variables aléatoires.

L'évaluation de mesures relativement à un graphe orienté acyclique pondéré par des variables aléatoires lève un problème fondamental qui se retrouve dans plusieurs domaines tels que la gestion de projet [113] et la conception de circuit digitaux [28].

En pratique, ce problème est difficile car la plupart des approches pour le résoudre nécessitent de prendre en compte des dépendances causées par la structure du graphe lors des calculs des opérations de maximum et de somme sur des variables aléatoires.

De nombreuses méthodes ont été proposées pour résoudre ce problème (de manière exacte ou approchée). Cependant, il existe peu de comparaison de leurs performances respectives. D'autre part, dans un contexte d'ordonnancement, nous avons besoin d'une méthode précise et suffisamment rapide pour être utilisable dans le cadre d'une heuristique de construction qui évalue des solutions partielles itérativement. Devant le manque de solutions pour le compromis rapidité/précision que nous recherchons, nous proposons de nouvelles méthodes que nous comparons par ailleurs avec plusieurs approches existantes.

Nous formalisons le problème dans la section 1.2 en restant général, c'est-à-dire indépendant d'une problématique d'ordonnancement particulière. Ce problème a fait l'objet de nombreuses contributions dont nous en présentons les éléments essentiels dans la section 1.3. La section 1.4 statue sur la complexité du problème. Enfin, nous proposons de nouvelles méthodes dans la section 1.5 que nous validons empiriquement dans la section 1.6.

1.2 Modèle

Nous définissons formellement le problème dans cette section. Après avoir introduit la structure du problème, nous montrons qu'un certain nombre de variantes sont possibles et qu'elles sont pour la plupart équivalentes. Nous précisons le type de variables aléatoires que nous utilisons et montrons enfin le lien avec l'ordonnancement. Les notations utilisées dans ce chapitre sont résumées dans le tableau 1.1.

1.2.1 Graphe orienté acyclique et stochastique

Soit un graphe orienté acyclique $G = (V, E, \mathcal{X})$. Un *poids* est associé à chaque sommet et à chaque arc. Le poids du sommet $v_i \in V$ est noté $X_i \in \mathcal{X}$ et celui de l'arc $(v_i, v_j) \in E$ est noté $X_{ij} \in \mathcal{X}$. Chaque poids est une variable aléatoire et le graphe est dit *stochastique*. Le graphe G possède n sommets et m arcs (i.e., $|V| = n$, $|E| = m$ et $|\mathcal{X}| = m + n$).

La structure du graphe possède une sémantique qui spécifie une expression arithmétique sur les poids des sommets et des arrêtes. À chaque sommet $v_i \in V$, nous définissons le *résultat*

Symbole	Définition
$G = (V, E, \mathcal{X})$	Graphe orienté acyclique stochastique
$V = \{v_i : i \in [1..n]\}$	Ensemble des sommets
n	Nombre de sommets ($n = \|V\|$)
E	Ensemble des arcs
m	Nombre d'arcs ($m = \|E\|$)
$\mathrm{Pred}(v_i)$	Ensemble des prédécesseurs du sommet $v_i \in V$ ($\mathrm{Pred}(v_i) \subset V$)
$\mathrm{Succ}(v_i)$	Ensemble des successeurs du sommet $v_i \in V$ ($\mathrm{Succ}(v_i) \subset V$)
$\mathcal{X}$	Ensemble des variables aléatoires ($\|\mathcal{X}\| = n + m$)
X_i	Poids du sommet $v_i \in V$ ($X_i \in \mathcal{X}$)
X_{ij}	Poids de l'arc $(v_i, v_j) \in E$ ($X_{ij} \in \mathcal{X}$)
Y_i	Résultat intermédiaire du sommet $v_i \in V$ ($Y_i = X_i + \max_{v_j \in \mathrm{Pred}(v_i)} Y_{ji}$)
Y_{ij}	Résultat intermédiaire de l'arc $(v_i, v_j) \in E$ ($Y_{ij} = X_{ij} + Y_j$)
Y_n	Résultat final
f_η	Densité de probabilité d'une variable aléatoire η
F_η	Fonction de répartition d'une variable aléatoire η ($F_\eta(x) = \Pr[\eta \leq x]$)
μ_η	Espérance d'une variable aléatoire η
σ_η^2	Variance d'une variable aléatoire η
N_η	Taille du domaine de définition d'une variable aléatoire η
T	Quantité de tirages réalisés par l'approche de Monte Carlo
$\eta \sim \mathcal{N}(\mu_\eta, \sigma_\eta)$	Une variable aléatoire η suit une loi normale d'espérance μ_η et d'écart type σ_η
$\rho_{\eta,\varepsilon}$	Coefficient de corrélation linéaire entre les variables aléatoires η et ε
$G' = (A, B)$	Instance de CONNEXION
$\mathrm{prof}(v_i)$	Profondeur du sommet $v_i \in V$

TABLE 1.1 – Notations du chapitre

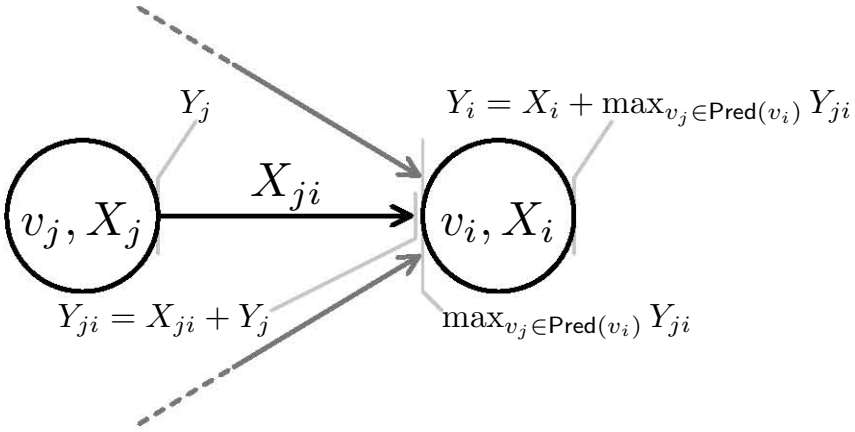


FIGURE 1.2.1 – Résultats intermédiaires dans un sous-graphe de deux sommets (v_j et v_i). Les opérations arithmétiques réalisées sont : la somme pour calculer Y_{ji} (à la fin de l'arc (v_j, v_i)) et le maximum pour Y_i (maximum de tous les arcs entrants avant le sommet, puis somme avec le poids du sommet X_i).

intermédiaire de cette expression par la variable $Y_i = X_i + \max_{v_j \in \mathrm{Pred}(v_i)} Y_{ji}$. De même, pour chaque arc $(v_i, v_j) \in E$, le résultat intermédiaire est $Y_{ij} = X_{ij} + Y_j$. Un maximum est réalisé lorsque plusieurs arcs pointent vers le même sommet. Parallèlement, les poids rencontrés sur un chemin sont additionnés. Ces deux opérations sont représentées sur la figure 1.2.1. Le résultat intermédiaire Y_n du sommet puits v_n est le résultat final de l'expression arithmétique décrite par le graphe stochastique.

Finalement, le graphe G encode une variable aléatoire qui suit une loi de probabilité que nous cherchons à caractériser. Nous dirons que notre problème consiste à évaluer *la distribution d'un graphe stochastique.*

La figure 1.2.2 permet d'illustrer les opérations arithmétiques représentées dans un graphe stochastique. Le résultat intermédiaire correspondant au sommet v_2 est $Y_2 = X_2 + Y_{12}$. Comme $Y_{12} = X_{12} + Y_1$ et que $Y_1 = X_1$, alors $Y_2 = X_2 + X_{12} + X_1$. Un maximum est rencontré lors du calcul du résultat intermédiaire du sommet v_3. Nous avons $Y_3 = X_3 + \max(Y_{23}, Y_{13})$. On peut exprimer Y_3 uniquement en fonction des poids de $\mathcal{X}$. Nous obtenons alors $Y_3 = X_3 + \max(X_{23} + X_2 + X_{12} + X_1, X_{13} + X_1)$. L'expression encodée par ce graphe est donnée par le résultat final Y_4 (le résultat intermédiaire du sommet puits s_4). Il s'agit de $Y_4 = X_4 + \max(X_{24} + X_2 + X_{12}, X_{34} + X_3 + \max(X_{23} + X_2 + X_{12}, X_{13})) + X_1$. Nous avons factorisé le poids X_1 dans les maximums et la formule ne se factorise pas davantage. Notre problème consiste à caractériser Y_4, les variables aléatoires de l'ensemble $\mathcal{X}$ étant connues.

Remarquons que toutes les expressions arithmétiques permises par une algèbre max-plus ne sont pas représentables par un graphe stochastique. Par exemple, il n'est pas possible d'encoder l'expression $X + X$ où X est une variable aléatoire quelconque. Nous sommes donc confrontés à un ensemble d'expressions spécifiques.

1.2.2 Équivalence des représentations

Suivant le cadre applicatif dans lequel survient cette problématique, les poids des sommets (ou des arcs) peuvent être nuls. En revanche, le problème ainsi obtenu est équivalent au nôtre. En effet, nous pouvons réduire une instance de notre problème en une instance où les sommets ont des poids nuls en temps polynomial. Pour ce faire, chaque sommet est remplacé par une paire de sommets liés par un arc dont le poids est celui du sommet initial. Le premier (resp., second)

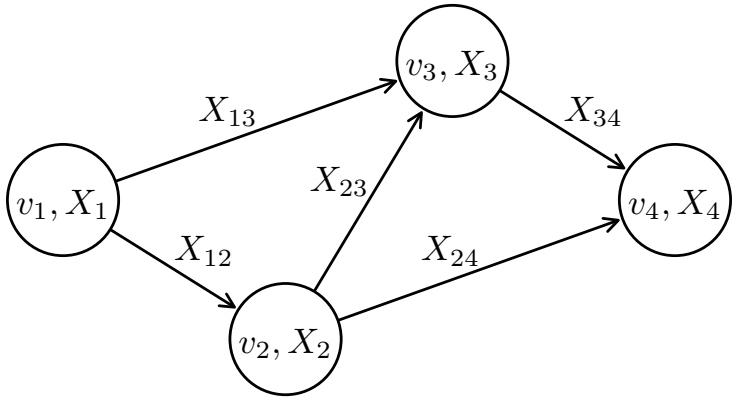


FIGURE 1.2.2 – Graphe stochastique avec quatre sommets (v_1,v_2, v_3 et v_4) ainsi que les poids associés à ces sommets et aux arcs qui les relient

sommet de cette paire devient un successeur (resp., prédécesseur) de tous les prédécesseurs (resp., successeurs) du sommet initial. On constate que le nombre de sommets augmente d'un facteur deux par cette transformation. D'autre part, son coût est linéaire en fonction du nombre d'arcs m. La réduction réciproque est immédiate. De façon analogue, si tous les arcs ont des poids nuls, alors un algorithme de réduction similaire existe.

Comme ces représentations sont équivalentes, nous préciserons au besoin si les poids des sommets ou ceux des arcs sont nuls.

1.2.3 Variable aléatoire

Soit η une variable aléatoire quelconque. Sa densité de probabilité f_η est définie sur $\mathbb{R}$. Sa fonction de répartition est $F_\eta(x) = \int_{-\infty}^{x} f_\eta(x)dx$. La fonction F_η donne la probabilité que η prenne une valeur inférieure à un seuil donné, i.e., $F_\eta(x) = \Pr[\eta \leq x]$. Enfin, nous dénotons l'espérance de η par μ_η et sa variance par σ_η^2.

Il n'existe pas de contrainte sur la nature des variables aléatoires associées aux sommets et aux arcs à priori. En revanche, les variables aléatoires sont manipulées numériquement dans certaines méthodes. Il existe principalement deux situations non-exclusives :

- Chaque variable aléatoire est définie par une fonction de masse dont le domaine de définition est fini et dénombrable. Pour qu'une telle variable aléatoire soit représentable par une machine de Turing, il faut que les probabilités et les valeurs des domaines de définition soient toutes rationnelles. Cela implique immédiatement que l'espérance et la variance sont des nombres rationnels. Nous parlerons alors de représentation *discrète*. Le domaine de définition d'une variable aléatoire discrète est *régulier* si les valeurs de ce domaine sont toutes espacées suivant la même constante.
- Chaque variable aléatoire est définie par une loi de probabilité connue. Il s'agit alors d'une représentation *symbolique*. En toute généralité, les valeurs du domaine de définition ne sont pas encodables numériquement (le domaine pouvant même être infini). Dans ce cas, la densité de probabilité est discrétisée aboutissant ainsi à une approximation. L'espérance et la variance qui en résultent sont donc des nombres rationnels.

Lorsqu'une loi de probabilité est discrétisée, cela signifie que N valeurs de la densité de probabilité sont stockées dans un vecteur avec les abscisses correspondantes. Si ces valeurs ou si les abscisses ne sont pas rationnelles, alors elles sont arrondies.

Nous précisons par la suite si le problème accepte des variables aléatoires discrètes ou des variables aléatoires symboliques. Notons que dans les méthodes ne nécessitant pas une manipu-

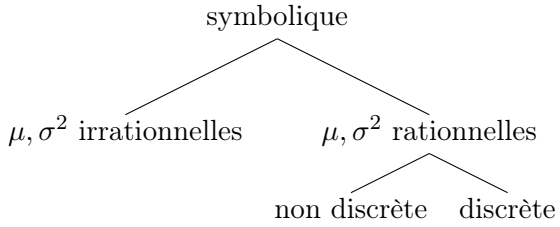


FIGURE 1.2.3 – Classification des variables aléatoires considérées

lation numérique des lois de probabilité, il n'existe pas de contraintes sur les variables aléatoires. Cependant, la majorité des méthodes requièrent que l'espérance et la variance soient rationnelles. La figure 1.2.3 montre une classification des variables aléatoires.

1.2.4 Ordonnancement à durées aléatoires

Dans le cas de l'ordonnancement, les durées des tâches ordonnancées peuvent être aléatoires, c'est-à-dire modélisées par des variables aléatoires. Nous considérons le cas général où les tâches sont soumises à des contraintes de dépendance. La durée totale d'un ordonnancement est alors aussi une variable aléatoire et son évaluation se réduit au problème que nous étudions dans ce chapitre.

Cette réduction s'obtient en remarquant qu'une tâche assignée à un processeur donné ne peut pas commencer son exécution tant qu'un de ses prédécesseurs n'a pas terminé la sienne et tant que le processeur en question n'a pas d'abord fini les tâches qui lui sont antérieures. On a donc deux types de dépendances, celles qui proviennent du graphe de tâches et celles qui sont liées à l'ordre dans lequel les tâches sont exécutées sur chaque processeur (ces dernières contraintes venant enrichir le graphe de tâches d'arcs supplémentaires sans le rendre cyclique). Ces derniers arcs qui correspondent à des contraintes d'antériorité imposées par l'ordonnancement sont appelés *arcs disjonctifs* dans [135].

D'une part, une date de début d'exécution d'une tâche s'obtient en réalisant une opération de maximum sur des dates de fin d'exécution. D'autre part, une date de fin d'exécution est le résultat d'une somme entre une durée aléatoire et une date de début d'exécution. Nous avons donc le même problème que celui décrit dans cette section.

Une application composée de quatre tâches est ordonnancée sur deux processeurs sur la figure 1.2.4. La tâche t_1 ne dépend d'aucune autre et est donc la première à commencer son exécution. Les tâches t_2 et t_3 dépendent toutes les deux de la tâche t_1 et ne peuvent commencer leur exécutions avant que t_1 n'ait fini la sienne. Enfin, la tâche t_4 dépend de ces deux précédentes tâches. Les tâches t_1 et t_4 sont exécutées sur le processeur p_1 et les tâches t_2 et t_3 le sont sur p_2. Comme l'exécution de t_2 est antérieure à celle de t_3, un arc disjonctif est rajouté entre ces deux tâches dans le graphe stochastique réduit. Dans cet exemple, les durées des tâches et les poids ne sont pas représentés. La durée de chaque tâche pondère le sommet du graphe stochastique correspondant à la tâche en question et les poids des arcs disjonctifs sont nuls.

1.2.5 Dépendance des résultats intermédiaires

Pour déterminer le résultat intermédiaire associé à un sommet $v_i \in V$, il faut évaluer l'expression $Y_i = X_i + \max_{v_j \in \mathrm{Pred}(v_i)} Y_{ji}$. Les opérandes du maximum sont toujours des résultats intermédiaires. Le problème majeur que révèle la plupart des travaux existants concerne la dépendance entre ces résultats intermédiaires. S'ils étaient indépendants, évaluer la distribution

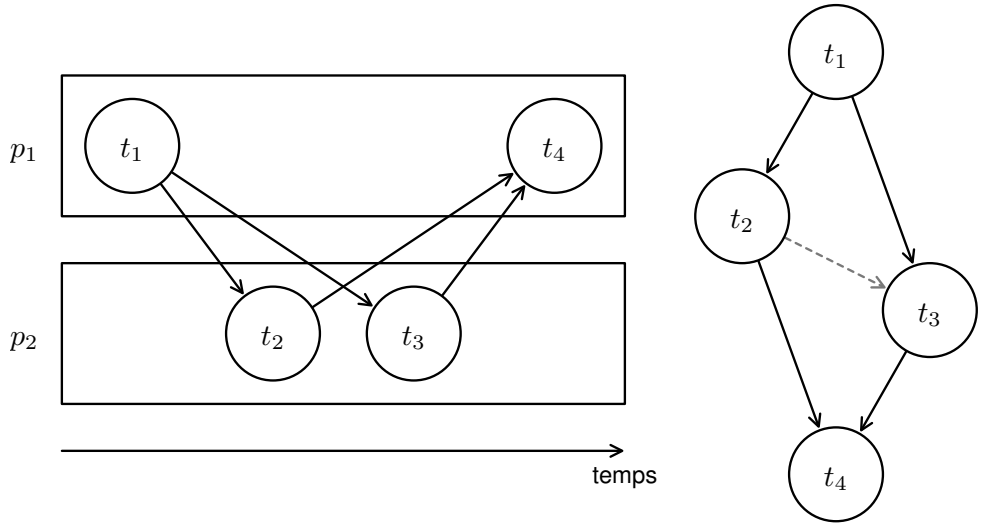


FIGURE 1.2.4 – Quatre tâches (t_1, t_2, t_3 et t_4) sont ordonnancées sur deux processeurs (p_1 et p_2). Le graphe stochastique obtenu après réduction contient un arc disjonctif (en gris) absent du graphe de tâches initial.

d'un graphe stochastique serait en effet facile.

La figure 1.2.2 montre ce phénomène. Le résultat intermédiaire du sommet v_3 se formule $X_3 + \max(Y_{23}, Y_{13})$. Les opérandes du maximum sont Y_{23} et Y_{13} qui sont dépendantes car chacune s'exprime en fonction de X_1.

Suivant les domaines, le problème posé par la dépendance entre les opérandes est aussi appelé *reconvergence des chemins* ou *dépendance topologique*.

1.3 État de l'art

La détermination de la distribution d'un graphe stochastique est un problème fondamental qui a été traité dans plusieurs domaines scientifiques :

- Le problème est défini pour la première fois par Malcolm et al. [113] dans le cadre de la gestion de projets. Il a depuis fait l'objet de développements réguliers.
- L'ordonnancement de graphe de tâches à durées stochastiques sur machines parallèles a été introduit peu après [38, 150, 118]. De nombreuses références sont disponibles dans [119].
- Enfin, la problématique se retrouve lors de la conception des circuits digitaux. Le problème y est en revanche enrichi de contraintes supplémentaires qui se traduisent notamment par l'existence de dépendances entre les variables aléatoires de l'ensemble $\mathcal{X}$ (voir l'état de l'art proposé par Blaauw et al. [28] pour plus de détails sur ce domaine).

Nous commençons par présenter différents mécanismes pour évaluer le résultat d'une opération arithmétique sur une paire de variables aléatoires. Sur la base de ces mécanismes, nous abordons ensuite des méthodes qui estiment la distribution d'un graphe stochastique. Parmi les méthodes existantes, nous distinguons d'abord les méthodes heuristiques qui offrent un résultat approché, les méthodes produisant des bornes, les méthodes exactes (qui ne sont par abordées car elles sont trop coûteuses) et enfin l'approche de Monte Carlo.

1.3.1 Évaluation des opérations arithmétiques

Dans l'évaluation de la distribution d'un graphe stochastique, deux opérations arithmétiques sont rencontrées : le maximum et la somme. Ces opérations sont réalisées sur des variables aléatoires que nous appelons *opérandes*. Nous présentons dans cette section des méthodes qui permettent de caractériser le résultat de ces opérations. Les premières méthodes sont numériques et travaillent sur les densités de probabilité des opérandes. Les secondes méthodes permettent de déterminer l'espérance et la variance du résultat si les opérandes suivent une loi normale.

1.3.1.1 Évaluation numérique

Caractériser la densité de probabilité du maximum ou de la somme de deux variables aléatoires indépendantes ne présente pas de difficulté majeure. Nous donnons les développements nécessaires pour obtenir les résultats correspondants provenant de la théorie des probabilités. Nous adaptons au besoin ces résultats aux variables aléatoires discrètes et à celles dont nous avons discrétisé la densité de probabilité.

Maximum de deux variables aléatoires indépendantes Soit η et ε deux variables aléatoires indépendantes. Nous appelons $\omega = \max(\eta, \varepsilon)$ le maximum de η et ε. La valeur de ω est inférieure à une constante z si et seulement si les deux opérandes lui sont aussi inférieures. Ainsi :

$$\begin{aligned} \Pr[\omega \leq z] &= \Pr[\eta \leq z \cap \varepsilon \leq z] && \text{définition de } \omega \\ &= \Pr[\eta \leq z] \times \Pr[\varepsilon \leq z] && \eta \text{ et } \varepsilon \text{ sont indépendants} \\ F_\omega(z) &= F_\eta(z) \times F_\varepsilon(z) && \text{définition de la fonction de répartition} \end{aligned}$$

La dernière ligne permet de déterminer la fonction de répartition de ω. Ainsi, la fonction de répartition du maximum de deux variables aléatoires est le produit de leurs fonctions de répartition. Ce résultat est également valide avec des opérandes discrètes. Dans ce cas, le nombre de valeurs du domaine de définition résultant est au pire la somme des tailles des domaines de définition des opérandes moins un. Si les opérandes ont été discrétisées en revanche, nous employons des méthodes d'analyse numérique pour estimer au mieux le résultat. Notons que pour déterminer une densité de probabilité à partir d'une fonction de répartition, il faut procéder à une dérivation. Or, cette opération est numériquement délicate et nous préférons obtenir directement la densité de probabilité de ω. Pour cela, nous dérivons analytiquement la dernière ligne : $f_\omega(z) = F_\eta(z) \times f_\varepsilon(z) + f_\eta(z) \times F_\varepsilon(z)$. Cette formule requiert les fonctions de répartition de η et de ε qui s'obtiennent par intégration numérique.

Somme de deux variables aléatoires indépendantes Considérons les mêmes opérandes dans la somme $\omega = \eta + \varepsilon$. La densité de probabilité jointe de η, ε et ω est notée $f_{\eta,\varepsilon,\omega} : \mathbb{R}^3 \mapsto \mathbb{R}^+$.

$$\begin{aligned} f_\omega(z) &= \iint_{xy} f_{\eta,\varepsilon,\omega}(x, y, z)dxdy && \text{loi des probabilités totales} \\ &= \iint_{xy} f_\eta(x) f_\varepsilon(y) f_\omega(z|\eta = x, \varepsilon = y)dxdy && \eta \text{ et } \varepsilon \text{ sont indépendants} \\ &= \iint_{xy} f_\eta(x) f_\varepsilon(y) \delta(z - (x + y))dxdy && f_\omega \text{ est une fonction } \delta \text{ de Dirac} \\ &= \int_x f_\eta(x) f_\varepsilon(z - x)dx && \text{substitution de } y \text{ par } z - x \\ f_\omega &= f_\eta * f_\varepsilon && \text{définition de la convolution} \end{aligned}$$

À la troisième ligne, nous utilisons le fait que $f_\omega(z|\eta = x, \varepsilon = y) = \delta(z - (x + y))$ car la fonction f_ω prend une valeur nulle pour tous les antécédents différents de $x + y$. La densité de probabilité de deux variables aléatoires est donc la convolution de leurs densités de probabilité. Si les opérandes sont discrètes, la fonction de masse de la somme s'obtient grâce à la formule $\Pr[\omega = z] = \sum_x \Pr[\eta = x] \times \Pr[\varepsilon = z - x]$. Soit N_ω la taille du domaine de définition de la variable aléatoire ω. Dans le pire des cas, $N_\omega = N_\eta \times N_\varepsilon$. Si les valeurs des domaines de définition de η et de ε sont espacées également (ce qui implique leurs régularités), alors $N_\omega = N_\eta + N_\varepsilon - 1$. Si les opérandes sont des estimations numériques, alors nous avons recours à la convolution discrète.

Numériquement, la complexité du calcul d'une convolution est $O(N^2)$ où N est le nombre de valeurs représentant une densité de probabilité. L'usage de la transformée de Fourier accélère ce calcul. En effet dans le domaine fréquentiel, la convolution est un produit dont le coût est linéaire. La transformation peut être réalisée par l'algorithme de transformée de Fourier rapide qui est en $O(N \log N)$.

1.3.1.2 Espérance et variance en cas de normalité

Lorsque qu'une opération est réalisée sur des variables aléatoires qui suivent chacune une loi normale, alors il est possible de déterminer l'espérance et la variance de la variable aléatoire résultante. Remarquons que les opérandes peuvent être dépendantes. Quand les opérandes sont normales, cette dépendance se réduit entièrement à leur coefficient de corrélation linéaire.

Maximum de lois normales corrélées Clark [42] propose un ensemble de formules pour traiter de cas du maximum. Les quatre premiers moments du maximum de deux loi normales sont ainsi caractérisés. Soit η et ε, deux variables aléatoires suivant chacune une loi normale. Leurs espérances et variances sont notés μ_η, μ_ε σ_η^2 et σ_ε^2. Le coefficient de corrélation linéaire entre η et ε est $\rho_{\eta,\varepsilon}$. Nous définissons deux fonctions : $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ et $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$. Clark caractérise l'espérance et la variance de $\omega = \max(\eta, \varepsilon)$, c'est-à-dire μ_ω et σ_ω^2 :

$$\mu_\omega = \mu_\eta \Phi(b) + \mu_\varepsilon \Phi(-b) + a\varphi(b) \tag{1.3.1}$$

$$\sigma_\omega^2 = (\mu_\eta^2 + \sigma_\eta^2)\Phi(b) + (\mu_\varepsilon^2 + \sigma_\varepsilon^2)\Phi(-b) + (\mu_\eta + \mu_\varepsilon)a\varphi(b) - \mu_\omega^2 \tag{1.3.2}$$

où $a = \sqrt{\sigma_\eta^2 + \sigma_\varepsilon^2 - 2\sigma_\eta\sigma_\varepsilon\rho_{\eta,\varepsilon}}$ et $b = \frac{\mu_\eta - \mu_\varepsilon}{a}$.

De plus, Clark fournit une formule pour déterminer le coefficient de corrélation linéaire entre le résultat d'un maximum et une variable aléatoire quelconque τ :

$$\rho_{\tau,\omega} = \frac{\sigma_\eta \rho_{\tau,\eta} \Phi(b) + \sigma_\varepsilon \rho_{\tau,\varepsilon} \Phi(-b)}{\sigma_\omega} \tag{1.3.3}$$

Somme de lois normales corrélées Dans le cas d'une somme, ces formules possèdent chacune un équivalent. Soit la somme $\omega = \eta + \varepsilon$. Les formules qui suivent sont des résultats probabilistes généraux. Elles sont d'ailleurs valides quelque soit les lois de probabilité de η et ε.

$$\mu_\omega = \mu_\eta + \mu_\varepsilon \tag{1.3.4}$$

$$\sigma_\omega^2 = \sigma_\eta^2 + 2\sigma_\eta\sigma_\varepsilon\rho_{\eta,\varepsilon} + \sigma_\varepsilon^2 \tag{1.3.5}$$

$$\rho_{\tau,\omega} = \frac{\sigma_\eta \rho_{\tau,\eta} + \sigma_\varepsilon \rho_{\tau,\varepsilon}}{\sigma_\omega} \tag{1.3.6}$$

1.3.2 Approches heuristiques

Nous classifions les heuristiques en plusieurs catégories : les approches basées sur les réductions séries-parallèles ; celles basées sur la supposition de la normalité ; et, l'approche canonique.

1.3.2.1 Réductions séries-parallèles

Une méthode basée sur une succession de réductions est d'abord présentée par Martin [114], puis par Dodin [53]. Si toutes les variables aléatoires sont discrètes sur des domaines de définition réguliers (suivant la même constante pour tous les poids), alors le coût de cette méthode est polynomial.

Son fonctionnement repose sur deux types de réductions que nous décrivons en considérant que les poids des sommets sont nuls :

Réduction séries Si un sommet possède exactement un arc entrant et un arc sortant, alors une réduction séries est réalisée. Celle-ci consiste à éliminer le sommet et à remplacer les deux arcs par un seul ayant pour poids la somme des poids des deux arcs d'origine. Comme les poids additionnés sont des variables aléatoires indépendantes, toutes les techniques décrites dans section 1.3.1 sont applicables.

Réduction parallèle Une réduction parallèle est réalisée s'il existe deux arcs ayant la même source et la même destination. Ils sont alors remplacés par un arc ayant pour poids le maximum des poids des deux arcs d'origine. Dans ce cas là aussi, les opérandes sont indépendantes et nous pouvons à nouveau utiliser les techniques décrites précédemment.

Pour un graphe stochastique donné, toutes les réductions possibles sont effectuées. Notons qu'une réduction peut rendre possible une nouvelle réduction. Les réductions sont donc réalisées de façon itérative jusqu'à obtenir un graphe réduit où toute réduction supplémentaire est impossible. L'opération aboutit à un unique arc entre un sommet source et un sommet puits si et seulement si le graphe d'origine est un graphe séries-parallèle. La distribution exacte du graphe stochastique est alors donnée par le poids de l'arc final.

Si le graphe n'est pas séries-parallèle, alors les réductions aboutissent à un graphe irréductible possédant plusieurs arcs. Dans ce graphe, un sommet est sélectionné aléatoirement parmi ceux qui ne possèdent qu'un seul arc entrant. Ce sommet et son arc entrant sont alors dupliqués autant de fois que le sommet possède d'arc sortants. Chacun de ces arcs sortants est connecté à un sommet distinct parmi ceux générés. S'il n'existe pas de sommet ne possédant qu'un seul arc entrant, un procédé analogue pour les sommets ne possédant qu'un seul arc sortant est utilisé. Après ce mécanisme de duplication, toutes les réductions possibles sont de nouveau réalisées jusqu'à l'obtention d'un graphe irréductible. Si celui-ci possède encore plusieurs arcs, alors l'étape entière est répétée jusqu'à l'obtention d'un unique arc.

Comme des arcs sont dupliqués lorsque le graphe n'est pas séries-parallèle, les poids correspondants sont aussi dupliqués. Cela signifie qu'à une étape donnée, le graphe contient des poids qui ne sont pas indépendants. Les opérations de maximum et de somme sur des variables aléatoires dépendantes posent problème (sauf si les opérandes suivent une loi normale). En toute généralité, le résultat est donc une estimation.

Kamburowski [24] améliore cette méthode en minimisant le nombre de sommets à dupliquer. D'autre part, Ludwig et al. [112] perfectionne l'approche en diminuant le coût algorithmique nécessaire pour trouver les réductions séries-parallèles possibles.

1.3.2.2 Supposition de la normalité

Considérer que toutes les variables aléatoires d'un graphe stochastique suivent des lois normales est une approximation largement répandue dans la littérature. La supposition de la normalité concerne aussi les résultats intermédiaires et la distribution finale. Cela offre un cadre applicatif idéal pour l'application des formules de Clark [42] qui estiment les deux premiers moments du maximum de deux normales (voir la section 1.3.1.2).

Cette supposition est supportée par le théorème central de la limite qui établit que la somme d'un grand nombre de variables aléatoires tend à être distribuée normalement. Comme le graphe stochastique encode une expression arithmétique contenant de nombreuses additions, le résultat tend à se rapprocher d'une loi normale si les maximums n'ont pas d'impact significatif sur la distribution du graphe stochastique.

La méthode proposée par Sculli [131] est une application directe de l'approche de Clark. Chaque variable aléatoire est réduite à son espérance et à sa variance. L'opération de maximum est appliqué directement en considérant que les opérandes suivent des lois normales indépendantes. Le résultat obtenu est de nouveau réduit en une loi normale dont les moments sont ceux générés grâce aux formules de Clark.

L'approche de Sculli possède toutefois quelques limites. La première est liée au fait que le coefficient de corrélation entre les opérandes est systématiquement considéré comme nul. Ceci est faux lorsque les opérandes sont associées à des arcs qui possèdent un ancêtre commun. Ignorer l'effet induit par la reconvergence des chemins conduit à une accumulation des erreurs qui est significative dès lors que le graphe est de grande taille. Nous proposons dans ce chapitre deux méthodes basées sur le même principe avec des techniques qui permettent d'estimer les coefficients de corrélation.

La seconde limite concerne la supposition de la normalité. En toute généralité, il est naturellement erroné de considérer que les variables aléatoires initiales suivent des lois normales. Quand bien même ce serait le cas pour une instance spécifique, il reste que le résultat de chaque opération de maximum est approximé par une loi normale, ce qui est encore invalide. Cela lève une réflexion fondamentale sur la pertinence des modèles en terme d'efficacité et de précision. Citons à ce propos George Box [32] : « tous les modèles sont faux ; certains sont utiles ».

La supposition de la normalité offre cependant plusieurs avantages : nous disposons de résultats probabilistes formels ; l'erreur ainsi commise est faible, ce que nous validons empiriquement dans la section 1.6 ; et, le coût algorithmique des méthodes basées sur cette supposition est favorable.

La qualité de cette supposition dépend de plusieurs critères : la normalité des variables aléatoires initiales ; la profondeur du graphe qui détermine le nombre de sommes réalisées ; et, la dépendance et la similarité entre les opérandes de chaque maximum qui déterminent la normalité des résultats intermédiaires.

1.3.2.3 Approche canonique

L'évaluation de la distribution d'un graphe stochastique est un problème qui apparaît également lors de la conception des circuits digitaux. Toutefois, les méthodes proposées sont conçues pour gérer les corrélations dites spatiales, c'est-à-dire les dépendances existantes entre les poids du graphe stochastique. Par exemple, Chang et al. [128] décrivent comment appliquer l'analyse en composantes principales pour prendre en compte l'existence de corrélation spatiale. Nous considérons au contraire que toutes les variables aléatoires de $\mathcal{X}$ sont indépendantes.

Cependant, c'est dans ce contexte qu'est apparue la représentation canonique [147] qui permet de gérer efficacement à la fois les corrélations spatiales et les dépendances entre les opérandes

des maximums. Une extension est proposée par Zhang et al. [157] notamment pour réduire le coût algorithmique de la méthode.

Dans l'approche canonique chaque variable aléatoire (poids et résultat intermédiaire) se formule grâce à une espérance et une somme pondérée et finie de lois normales centrées réduites :

$$\eta = \mu + \sum_i \alpha_i \Upsilon_i$$

où μ est l'espérance de η. Chaque $\Upsilon_i \sim \mathcal{N}(0,1)$ est une variable aléatoire suivant une loi normale d'espérance nulle et de variance unitaire. Les paramètres α_i déterminent ainsi l'écart type de η. Dans ce formalisme, toutes des normales Υ_i sont indépendantes. Nous adaptons cette représentation à notre problématique, c'est-à-dire à l'indépendance des poids. Cela signifie que chaque poids est associé à une variable aléatoire Υ qui lui est spécifique. Nous utilisons la notation $\hat{X}_i$ car la représentation canonique est une estimation des véritables variables aléatoires X_i. En considérant que tous les arcs sont de poids nul, alors $\hat{X}_i = \mu_i + \alpha_i \Upsilon_i$ et $\hat{Y}_j = \mu_j + \sum_i \alpha_{ji} \Upsilon_i$. Cette seconde équation signifie que tout résultat intermédiaire s'exprime comme une combinaison linéaire des normales centrées réduites associées aux poids du graphe stochastique. Comme un résultat intermédiaire Y_j peut résulter d'un maximum (qui ne produit pas une forme linéaire), il est clair que la seconde équation n'en fournit qu'une approximation.

L'évaluation des opérations arithmétiques sur des variables aléatoires sous formes canoniques fait appel aux formules proposées par Clark. Soit $\eta = \mu_\eta + \sum_i \alpha_{\eta,i} \Upsilon_i$ et $\varepsilon = \mu_\varepsilon + \sum_i \alpha_{\varepsilon,i} \Upsilon_i$. La somme $\omega = \eta + \varepsilon$ s'évalue ainsi :

$$\omega = (\mu_\eta + \mu_\varepsilon) + \sum_i (\alpha_{\eta,i} + \alpha_{\varepsilon,i}) \Upsilon_i$$

Le maximum est défini par $\omega = \max(\eta, \varepsilon)$. Dans la section 1.3.1.2, nous définissons la fonction $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$ et les deux symboles a et b :

$$a = \sqrt{\sigma_\eta^2 + \sigma_\varepsilon^2 - 2\sigma_\eta \sigma_\varepsilon \rho_{\eta,\varepsilon}}$$
$$b = \frac{\mu_\eta - \mu_\varepsilon}{a}$$

La valeur $\Phi(b)$ est en fait équivalente à la probabilité que η prenne une valeur supérieure à ε, i.e., $\Pr[\eta > \varepsilon]$. Le maximum est approximé par

$$\begin{aligned}\hat{\omega} &= \Phi(b)\eta + \Phi(-b)\varepsilon \\ &= (\Phi(b)\mu_\eta + \Phi(-b)\mu_\varepsilon) + \sum_i (\Phi(b)\alpha_{\eta,i} + \Phi(-b)\alpha_{\varepsilon,i}) \Upsilon_i\end{aligned}$$

Évaluer le maximum ainsi nécessite de calculer la corrélation entre les opérandes :

$$\rho_{\eta,\varepsilon} = \frac{\sum_i \alpha_{\eta,i}\alpha_{\varepsilon,i}}{\sqrt{\sum_i \alpha_{\eta,i}^2}\sqrt{\sum_i \alpha_{\varepsilon,i}^2}}$$

L'approche canonique repose sur la supposition de la normalité avancée plus haut. Représenter chaque variable aléatoire grâce à un ensemble de lois normales centrées réduites offre une méthode élégante et rapide pour spécifier la dépendance entre les résultats intermédiaires du graphe stochastique. Cela se fait en revanche au détriment de l'opération de maximum qui perd en précision par rapport à la méthode proposée par Clark.

1.3.3 Bornes

De nombreuses méthodes permettent de borner la distribution d'un graphe stochastique. Avant d'en présenter quelques unes, nous définissons la relation de dominance stochastique d'ordre 1 [129, Section 2.1.4.4]. Nous l'utilisons pour déterminer si deux variables aléatoires sont comparables et, le cas échéant, pour savoir laquelle est supérieure. Soit η et ε deux variables aléatoires. La variable aléatoire η domine stochastiquement ε si $\Pr[\eta > x] \geq \Pr[\varepsilon > x]$ pour tout x.

Kleindorfer [97] prouve une borne inférieure et une borne supérieure. Cette dernière borne est obtenue en parcourant le graphe stochastique dans un ordre topologique et en appliquant les mécanismes décrits dans la section 1.3.1.1. Les opérandes de chaque maximum sont ainsi supposées indépendantes. Pour la borne inférieure, les opérations de maximum ne sont pas effectuées. À la place, la distribution d'une des opérandes est sélectionnée comme étant le résultat du maximum.

L'approche par réductions séries-parallèles décrite dans la section 1.3.2.1 fournit une borne supérieure en transformant un graphe quelconque en graphe séries-parallèle. Ce résultat améliore la borne supérieure de Kleindorfer.

Yazici-Pekergin et Vincent [155] propose de remplacer chaque distribution NBUE (New Better Than Used in Expectation) d'un graphe stochastique par une loi exponentielle de même espérance. Le résultat du graphe stochastique obtenu est une borne supérieure. Cette technique présente un intérêt lorsque l'on connaît uniquement l'espérance de chacune des variables aléatoires et le fait qu'elles vérifient toutes la propriété NBUE.

Certaines méthodes bornent uniquement l'espérance du résultat. Fulkerson [68] proposent l'une des premières bornes inférieures. Cette dernière est améliorée par Robillard et Trahan [123] en utilisant l'inégalité de Jensen. Enfin, Kamburowski [93] propose de borner l'espérance et la variance en supposant la normalité des variables aléatoires et en utilisant les formules de Clark.

Bien que de nombreuses bornes aient été proposées, nous y préférons des méthodes générant rapidement une estimation qui se révèle précise en pratique même si aucune garantie n'est fournie.

1.3.4 Méthode de Monte Carlo

La méthode de Monte Carlo proposée pour ce problème [146, 36] consiste à réduire itérativement le graphe stochastique en un graphe déterministe.

Pour chaque variable aléatoire du graphe stochastique, une valeur est tirée aléatoirement suivant sa loi. Après avoir tiré une valeur pour chaque poids, le résultat d'un graphe stochastique peut être évalué en considérant une seule valeur par variable aléatoire. Nous obtenons alors une valeur unique. L'opération peut être répétée plusieurs fois, générant ainsi une nouvelle valeur à chaque itération. L'ensemble des valeurs obtenues définissent une fonction de répartition empirique qui approche la fonction de répartition de la distribution du graphe stochastique lorsque le nombre de tirages augmente.

Nous devons donc définir la quantité de tirages à réaliser, que nous notons T, pour obtenir une précision acceptable. Si nous supposons que la distribution du graphe stochastique suit une loi normale, le théorème de Cochran stipule que l'estimation de la variance suit une loi de χ^2 avec $T-1$ degrés de liberté. Ainsi, le nombre de degrés nécessaire pour obtenir un intervalle de confiance resserré donne directement le nombre de tirage à réaliser pour la méthode de Monte Carlo. Avec 20 000 tirages, l'erreur relative de l'écart type de la distribution du graphe stochastique est inférieure à 5% avec un degré de confiance de 99%. Avec un million de tirages, on a moins de 1% d'erreur.

La mesure de Kolmogorov caractérise l'erreur en mesurant la distance entre la fonction de répartition empirique et la fonction de répartition réelle. D'après la distribution de Kolmogorov,

Type de variables aléatoires	Chaîne	Jointure	Séries-parallèle
Discrète	domaines réguliers	toutes	domaines réguliers
Non-discrète	normale, gamma, Erlang	exponentielle, Weibull	aucune

TABLE 1.2 – Résumé des types de graphes stochastiques pour lesquelles une méthode exacte à coût polynomial est connue

cette différence est inférieure à $1,629/\sqrt{T}$ avec un degré de confiance de 99% si le nombre de tirages dépasse 100. Pour 20 000 tirages, la différence entre les deux fonctions de répartition est inférieure à $1,2\%$. Pour un million de tirages, elle est inférieure à $1,629‰$.

La méthode de Monte Carlo présente deux avantages. D'abord, la fonction de répartition empirique converge vers la distribution du graphe aléatoire lorsque $T \to \infty$ d'après le théorème de Glivenko-Cantelli. Ensuite, cette méthode est insensible aux dépendances entre les opérandes des opérations de maximum.

1.3.5 Classes d'instances particulières

Le tableau 1.2 résume les sous-problèmes pour lesquels un algorithme à coût polynomial existe. Trois types de graphes sont retenus : les chaînes qui sont constituées d'une succession de sommets ; les jointures où chaque arc part d'un sommet distinct pour aboutir au sommet puits (les poids des sommets étant nuls) ; et, les graphes séries-parallèles. Nous distinguons les variables aléatoires discrètes pour lesquelles les méthodes de la section 1.3.1.1 produisent des résultats exacts. Pour les chaînes et les graphes séries-parallèles, tous les domaines de définition doivent être réguliers suivant la même constante (comme cela est défini dans la section 1.2.3, un domaine est régulier si toutes ses valeurs sont également espacées). Pour les variables aléatoires non-discrètes, certaines classes de distributions sont closes sous les opérations de maximum ou de somme. La distribution du graphe stochastique est alors caractérisée par une formule analytique.

1.4 Complexité du problème général

Nous détaillons d'abord les classes de complexité qui présentent un intérêt dans notre contexte, ainsi que quelques problèmes et résultats de références qui y sont relatifs. Nous prouvons ensuite la classe de complexité du problème étudié.

1.4.1 #P'-Complétude

Les classes de problèmes #P et #P-Complet ont été introduites par Valiant [145]. Informellement, il s'agit des problèmes qui « comptent le nombre de solutions ». Compter le nombre de solutions à un problème est au moins aussi dur que de déterminer s'il en existe au moins une. Les problèmes #P sont donc au moins aussi difficiles que leurs problèmes homologues NP. De la même manière que pour prouver qu'un problème est NP-Complet, une preuve de #P-Complétude nécessite deux parties : la première prouve l'appartenance à la classe #P et la seconde se base sur une réduction partant d'un problème connu #P-Complet. Il y a une complication technique pour les problèmes de fiabilité en ordonnancement ainsi que pour les problèmes de graphe [120, 30] : nous manipulons des probabilités qui sont des rationnels dans l'intervalle $[0; 1]$, par opposition aux entiers manipulés dans [145]. Aussi, nous utilisons la classe proposée par Bodlaender et

al. [30] et établissons la #P'-Complétude de notre problème. La classe #P' est une extension naturelle de la classe #P qui prend en compte les nombres rationnels : elle permet d'appliquer une fonction de coût polynomial sur la sortie entière d'un problème #P, produisant dans notre cas un nombre rationnel. Formellement, la classe #P' se définit par

$$\begin{aligned} \#P' \quad = \quad \{ \quad & h | h : \Sigma^* \Rightarrow \Sigma^*, \\ & \exists f \in \#P, f : \Sigma^* \Rightarrow \mathbb{N}, \\ & \exists g \in FP, g : \mathbb{N} \times \Sigma^* \Rightarrow \Sigma^*, \\ & \forall x \in \Sigma^* : h(x) = g(f(x), x) \quad \} \end{aligned}$$

Nous rappelons que la classe FP est la classe des fonctions qui s'évaluent en temps polynomial par une machine de Turing. Par ailleurs, Σ^* représente l'ensemble des langages possibles (c'est-à-dire la fermeture de Kleene de l'alphabet Σ). D'après cette définition, il apparaît qu'un problème h dans #P' consiste en un problème f dans #P dont le résultat est transformé en temps polynomial par $g(f(x), x)$ où x est une instance d'un problème donné.

1.4.2 Problèmes #P'-Complet

Dans [145, Problème 11], Valiant étudie le problème de graphe deux-terminaux. Il prouve que calculer le nombre de scénarios pour lesquels il existe un chemin entre deux sommets donnés est #P-Complet.

Provan et Ball [120, Problème 10 et section 3] étendent les résultats de Valiant dans le cas des graphes orientés acycliques. En particulier, ils prouvent qu'évaluer la probabilité R qu'il existe au moins un chemin d'arcs actifs entre deux sommets donnés est #P-Complet (il s'agit de #P'-Complétude d'après la terminologie que nous utilisons). D'après leurs résultats, l'approximation de R à une quantité arbitraire ε près, à savoir trouver une valeur r telle que $r - \varepsilon \leq R \leq r + \varepsilon$, est également #P'-Complet. D'autre part, les résultats de Rosenthal [125] permettent d'affirmer que l'approximation de R à un facteur arbitraire α près, c'est-à-dire déterminer une valeur r telle que $\frac{r}{\alpha} \leq R \leq r\alpha$, est aussi un problème #P'-Complet.

Nous étudions un problème similaire où le chemin d'arcs actifs relie le sommet source au sommet puits. Nous prouvons un résultat préliminaire sur ce problème, à savoir que le *problème de connexion*, dénoté CONNEXION est #P'-Complet. Il s'agit d'une conséquence directe des travaux antérieurs dans ce domaine.

Définition 1.1 (CONNEXION)**.** *Soit un graphe orienté acyclique $G' = (A, B)$ dont les arcs deviennent inactifs avec des probabilités rationnelles arbitraires. Les sommets source et puits du graphe G' sont uniques. Le* problème de connexion *consiste à calculer la* probabilité de connexion*, c'est-à-dire la probabilité que le sommet source et le sommet puits soient liés par un chemin d'arcs actifs.*

Dans le cas où un tel chemin existe, on dira que les sommets sources et puits sont connectés, ou plus concisément, que le sommet puits est *accessible* (implicitement, à partir du sommet source).

Théorème 1.2. CONNEXION *est #P'-Complet.*

Démonstration. La réduction se base sur une variante du problème qui est connu pour être #P'-Complet [120, Problème 10 et section 3] : calculer la probabilité que deux sommets arbitraires du graphe orienté acyclique $G' = (A, B)$, s et t, soient connectés. Nous réduisons une instance quelconque de ce problème à une instance de CONNEXION en insérant deux sommets dans le graphe G'. Le premier sommet inséré est relié à tous les sommets de A devenant ainsi le sommet

source. Chaque arc inséré est toujours inactif sauf celui à destination de s qui est quant à lui toujours actif. Par symétrie, le second sommet inséré devient le sommet puits et seul le nouvel arc partant de t est actif. D'autre part, plutôt que de considérer que toutes les probabilités d'activité des arcs sont égales, notre problème relâche cette contrainte et accepte des probabilités arbitraires. Ainsi, la probabilité que s et t soit connectés est la même que la probabilité que le sommet puits soit accessible dans l'instance obtenue par cette réduction. L'appartenance de CONNEXION à la classe #P' ne présente pas de difficulté et repose sur le même principe que la preuve plus élaborée du théorème 1.5. □

Corollaire 1.3. *Approximer le résultat de* CONNEXION *à une quantité ε près ou à un facteur α près est un problème #P'-Complet.*

Démonstration. D'après [120, Problème 10 et section 3], approximer la probabilité que deux sommets arbitraires du graphe orienté acyclique soient connectés à une quantité ε près ou à un facteur α près est un problème #P'-Complet. La preuve du théorème 1.2 montre que la réduction est parcimonieuse car le résultat de ce problème de départ est précisément le même que celui de l'instance de CONNEXION générée par l'algorithme de réduction. Le problème CONNEXION est donc lui aussi inapproximable à une quantité ou à un facteur près. □

1.4.3 Preuve de complexité

L'évaluation de la distribution d'un graphe stochastique est un problème très général. Nous en tirons un problème plus spécifique que nous appelons RÉPARTITION-GRAPHE et pour lequel nous précisons la nature des variables aléatoires. À savoir, nous considérons que les variables aléatoires sont discrètes. Par conséquent, les variables aléatoires manipulées sont définies par des fonctions de masse dont les domaines de définition sont finis et dénombrables. De plus, les probabilités et les valeurs des domaines de définition sont toutes rationnelles.

Définition 1.4. *Le problème* RÉPARTITION-GRAPHE *consiste à calculer la probabilité qu'un graphe stochastique $G = (V, E, \mathcal{X})$ prenne une valeur supérieure ou égale à un rationnel arbitraire R sachant que l'ensemble $\mathcal{X}$ ne contient que des variables aléatoires discrètes.*

La preuve de #P'-Complétude de RÉPARTITION-GRAPHE s'appuie sur une réduction partant du problème CONNEXION. Une preuve a déjà été proposée par Hagstrom [77] en se basant sur une réduction du même type. Cependant, cette réduction ne fonctionne pas pour toutes les instances de CONNEXION. Comme contre-exemple, nous proposons l'instance suivante : deux sommets reliés par un arc actif avec une probabilité quelconque. Notre preuve confirme et généralise les conclusions présentées dans [77].

Pour prouver que ce problème appartient à #P', nous devons caractériser le problème de décision NP sous-jacent et la transformation nécessaire pour générer la probabilité de sortie qui est dans notre cas un nombre rationnel.

Proposition 1.5. RÉPARTITION-GRAPHE *est dans #P'.*

Démonstration. Sans perte de généralité, nous supposons que chaque variable aléatoire $X_i \in \mathcal{X}$ du graphe stochastique est encodée sous la forme $((x_i^1, \frac{o_i^1}{d}), \ldots, (x_i^{s_i}, \frac{o_i^{s_i}}{d}))$ où X_i peut prendre s_i valeurs rationnelles $(x_i^1, \ldots, x_i^{s_i})$ avec les probabilités rationnelles correspondantes $(\frac{o_i^1}{d}, \ldots, \frac{o_i^{s_i}}{d})$ telles que $\sum_{k=1}^{s_i} o_i^k = d$ (où les o_i^k et d sont des entiers). Pour chaque variable aléatoire X_i, nous définissons une valeur entière $1 \leq \lambda_i \leq d$ qui spécifie la valeur que prend X_i lors d'un tirage spécifique. Si $\sum_{k=1}^{j-1} o_i^k < \lambda_i \leq \sum_{k=1}^{j} o_i^k$, alors X_i prend pour valeur x_j. Le vecteur $\lambda = (\lambda_i)_{1 \leq i \leq n}$

définie l'ensemble des valeurs que prennent les variables aléatoires du graphe stochastique lors d'un tirage donné.

Le problème NP inhérent à notre problème consiste à décider s'il existe ou non un vecteur λ tel que l'évaluation du graphe stochastique aboutisse à une valeur supérieure ou égale à R. Ce problème appartient à NP car le certificat est le vecteur λ de taille $n \times \log(d)$. Le problème #P reposant sur ce problème de décision compte le nombre de vecteurs distincts λ acceptés. Comme chaque scénario encodé par ces vecteurs sont équiprobables, nous obtenons la probabilité que G prenne une valeur supérieure ou égale à R en divisant le nombre de vecteurs λ favorables par le nombre total de scénarios possibles. □

Théorème 1.6. RÉPARTITION-GRAPHE *est #P'-Complet.*

Démonstration. Soit une instance de CONNEXION composée d'un graphe orienté acyclique $G' = (A, B)$ dont les arcs deviennent inactifs avec des probabilités rationnelles arbitraires. Nous proposons un algorithme de réduction qui transforme cette instance en un graphe $G = (V, E, \mathcal{X})$, instance de RÉPARTITION-GRAPHE. Chaque sommet $a_i \in A$ équivaut à un sommet $v_i \in V$ dont le poids est nul. Nous ignorons donc dans le reste de cette preuve les poids associés aux sommets. Pour chaque arc de B, un arc dans E est créé. Soit p la probabilité d'activité de ce premier arc. La variable aléatoire associée à l'arc correspondant dans E prend la valeur 0 avec une probabilité $1 - p$ et la valeur 1 avec une probabilité p. Enfin, nous transformons le graphe obtenu G en rajoutant des arcs et des sommets de telle sorte que pour chaque sommet, la profondeur de chacun de ses prédécesseurs est identique. Le poids de chacun des arcs ajoutés est une variable aléatoire prenant le valeur 1 avec une probabilité 1. L'insertion de ces arcs peut être réalisée en parcourant G dans un ordre topologique : pour chaque sommet, autant de sommets et d'arcs que nécessaires sont rajoutés entre ses prédécesseurs et lui.

La figure 1.4.1 montre un exemple d'une telle réduction sur une instance de CONNEXION possédant trois sommets. L'instance de RÉPARTITION-GRAPHE obtenue possède un sommet v_4 qui ne correspond à aucun sommet de l'instance de CONNEXION. Grâce à l'ajout de v_4, les prédécesseurs de v_3 sont tous de profondeur 2.

Nous prouvons par induction sur les tâches (dans un ordre topologique) qu'un sommet a_i de l'instance de CONNEXION est accessible si et seulement si la valeur du graphe stochastique en son sommet équivalent est supérieure ou égale à sa profondeur moins un, i.e., $Y_i \geq \text{prof}(v_i) - 1$. L'initialisation est réalisée pour le sommet source qui est par définition toujours accessible. Le poids du sommet correspondant est nul et sa profondeur moins un est également nulle. Les événements sont donc équivalent pour le sommet source. Supposons que pour tous les sommets a_k avec $1 \leq k < i$, a_k est accessible si et seulement si $Y_k \geq \text{prof}(v_k) - 1$. Nous considérons deux cas : le sommet a_i est accessible ou bien il ne l'est pas.

Si a_i est accessible, alors il possède un prédécesseur a_k qui est accessible et l'arc (a_k, a_i) est actif. Dans le graphe stochastique, cela signifie que v_i possède un ancêtre v_k pour lequel $Y_k \geq \text{prof}(v_k) - 1$ (d'après l'hypothèse d'induction) et il existe un chemin de sommets entre v_k et v_i tel que le premier arc prend pour valeur 1 (car l'arc (a_k, a_i) est actif) et les autres prennent toujours la valeur 1 (il s'agit de ceux rajoutés par construction). Nous avons alors $Y_i \geq Y_k + t$ où t est le nombre d'arcs entre v_k et v_i. Par construction, $\text{prof}(v_i) = \text{prof}(v_k) + t$. Ainsi, $Y_i \geq \text{prof}(v_i) - 1$.

Si a_i n'est pas accessible, alors pour chaque prédécesseur $a_k \in \text{Pred}(a_i)$, soit a_k est inaccessible, soit l'arc (a_k, a_i) est inactif. Nous montrons par l'absurde que $Y_i < \text{prof}(v_i) - 1$ dans ce cas. Remarquons d'abord que le cas $Y_i > \text{prof}(v_i) - 1$ est impossible par construction (le poids des arcs est au plus unitaire). Si $Y_i = \text{prof}(v_i) - 1$, alors il existe un prédécesseur $a_k \in \text{Pred}(a_i)$ dans CONNEXION dont le sommet correspondant dans le graphe stochastique vérifie soit $Y_k = \text{prof}(v_k) - 1$, soit $Y_k = \text{prof}(v_k)$ suivant que le poids du premier arc entre v_k et v_i prend la valeur 1 ou 0,

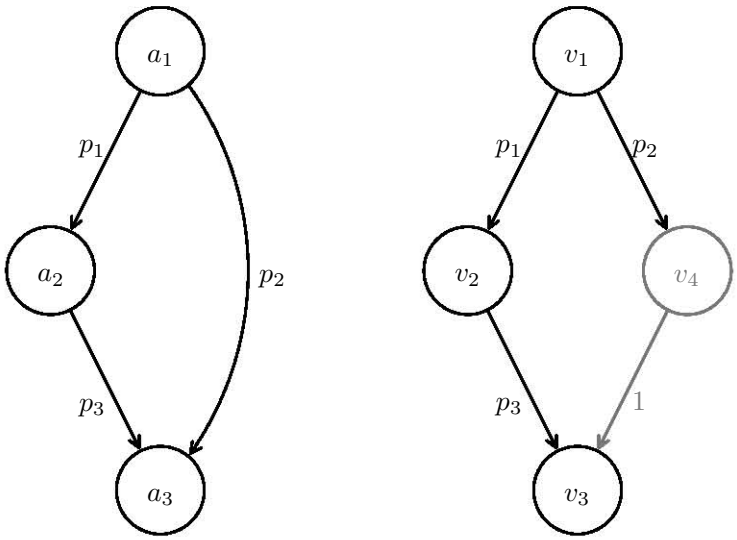


FIGURE 1.4.1 – Instance de CONNEXION (à gauche) avec trois sommets (a_1, a_2 et a_3) et trois arcs avec les probabilités d'activité p_1, p_2 et p_3. Le graphe stochastique obtenu après réduction (à droite) contient un sommet et un arc de poids unitaire (v_4 en gris) absent dans le graphe initial. Pour chaque sommet du graphe stochastique, tous les prédécesseurs ont la même profondeur.

respectivement. Le premier cas contredit le fait que soit a_k est inaccessible (ce qui implique que $Y_k < \text{prof}(v_k) - 1$ d'après l'hypothèse d'induction), soit (a_k, a_i) est inactif. Le second cas est impossible par construction.

Nous avons montré que si a_n est accessible, alors $Y_n \geq \text{prof}(v_n) - 1$. D'autre part, si a_n est inaccessible, alors $Y_n < \text{prof}(v_i) - 1$. L'équivalence entre ces deux événements est donc établie et le passage aux probabilités montre que le résultat de l'instance de CONNEXION est le même que celui de RÉPARTITION-GRAPHE pour $R = \text{prof}(v_n) - 1$. □

Corollaire 1.7. *Approximer le résultat de* RÉPARTITION-GRAPHE *à une quantité ε près ou à un facteur α près est un problème #P'-Complet.*

Démonstration. D'après le corollaire 1.3, approximer le résultat de CONNEXION à une quantité ε près ou à un facteur α près est un problème #P'-Complet. La preuve du théorème 1.6 montre que le résultat d'une instance de CONNEXION est le même que celui de l'instance de RÉPARTITION-GRAPHE générée par l'algorithme de réduction. La réduction est donc parcimonieuse. □

1.5 Méthodes

Nous proposons deux heuristiques qui se basent sur la supposition de la normalité présentée dans la section 1.3.2.2 et sur les formules de Clark [42] décrites dans la section 1.3.1.2. À savoir, nous supposons que chaque variable aléatoire suit une loi normale et le résultat de chaque maximum est converti en une loi normale. À ce titre, nos deux méthodes améliorent l'approche de Sculli [131] car elles comprennent chacune un mécanisme pour estimer les corrélations entre les opérandes lors des maximums.

1.5.1 CorLCA

La première heuristique que nous dérivons se nomme *CorLCA* (Correlation based on Lowest Commun Ancestor). Cette méthode parcourt l'ensemble des sommets du graphe une seule fois.

Algorithme 1.1 L'heuristique CorLCA basée sur la recherche d'un plus proche ancêtre commun pour estimer les corrélation entre deux opérandes

Entrée: $G = (V, E, \mathcal{X})$ {Graphe stochastique}
Sortie: (μ, σ^2) {Espérance et variance de la distribution de G}

1: **pour chaque** $v_i \in V$ **faire** {Parcourt les sommets dans un ordre topologique}
2: $\quad \dot{v}_i = 0$ {Initialisation du parent de v_i dans l'arbre de corrélation}
3: $\quad$ **pour chaque** $v_j \in \mathrm{Pred}(v_i)$ **faire**
4: $\quad\quad Y_{ji} = X_{ji} + Y_j$ {Équations 1.3.4 et 1.3.5}
5: $\quad\quad$ **si** $\dot{v}_i = 0$ **alors** {Première itération de la boucle}
6: $\quad\quad\quad \dot{v}_i = v_j$
7: $\quad\quad\quad \eta = Y_{ji}$
8: $\quad\quad$ **sinon**
9: $\quad\quad\quad v_k = \mathrm{LCA}(\dot{v}_i, v_j)$ {Détermine le plus proche ancêtre commun de $\dot{v}_i$ et v_j}
10: $\quad\quad\quad \rho_{\eta, Y_{ji}} = \frac{\sigma^2_{Y_k}}{\sigma_\eta \sigma_{Y_{ji}}}$ {Estime la corrélation entre η et Y_{ji}}
11: $\quad\quad\quad$ **si** $\Pr[\eta < Y_{ji}] > 0,5$ **alors** {Si le sommet v_j est prépondérant dans la maximum}
12: $\quad\quad\quad\quad \dot{v}_i = v_j$ {Change le prédécesseur de v_i dans l'arbre de corrélation}
13: $\quad\quad\quad$ **fin si**
14: $\quad\quad\quad \eta = \max(\eta, Y_{ji})$ {Équations 1.3.1 et 1.3.2, et ligne 10}
15: $\quad\quad$ **fin si**
16: $\quad$ **fin pour**
17: $\quad$ **si** $\dot{v}_i = 0$ **alors** {Le sommet v_i n'a pas de prédécesseur}
18: $\quad\quad Y_i = X_i$
19: $\quad$ **sinon**
20: $\quad\quad Y_i = X_i + \eta$ {Équations 1.3.4 et 1.3.5}
21: $\quad$ **fin si**
22: **fin pour**
23: **retourner** $(\mu_{Y_n}, \sigma^2_{Y_n})$

Pour chaque sommet, les corrélations entre les opérandes du maximum sont estimées en utilisant une méthode peu coûteuse. L'objectif de cette méthode est de fournir des résultats les plus précis possibles sans augmenter significativement le coût algorithmique par rapport à l'approche de Sculli (voir la section 1.3.2.2). Pour cela, les corrélations sont estimées en déterminant le plus proche ancêtre commun d'un couple d'opérandes. L'algorithme 1.1 décrit les étapes que suit CorLCA.

Nous décrivons d'abord le fonctionnement général de l'algorithme puis nous détaillons la construction de l'arbre de corrélation qui permet la recherche efficace d'un plus proche ancêtre commun. Nous montrons ensuite comment calculer un coefficient de corrélation grâce à cet ancêtre. Enfin, nous analysons la complexité de CorLCA.

L'heuristique CorLCA repose sur une boucle principale qui parcourt les sommets d'un graphe stochastique $G(V, E, \mathcal{X})$ dans un ordre topologique, c'est-à-dire que les indices des sommets sont ordonnés tels que $\forall (v_i, v_j) \in V^2, i < j \Rightarrow (v_j, v_i) \notin E$.

À chaque itération, deux types d'opérations sont réalisées :
– l'évaluation des résultats intermédiaires (lignes 4, 9, 10, 14 et 20)
– la construction incrémentale de l'*arbre de corrélation* (lignes 11 à 13)

Cet arbre est enraciné et sert uniquement au calcul des corrélations entre les opérandes de chaque maximum. Il possède les mêmes sommets que le graphe G ainsi qu'un sous-ensemble des arcs E. À savoir, chaque sommet de l'arbre de corrélation ne possède qu'un seul arc entrant (définition

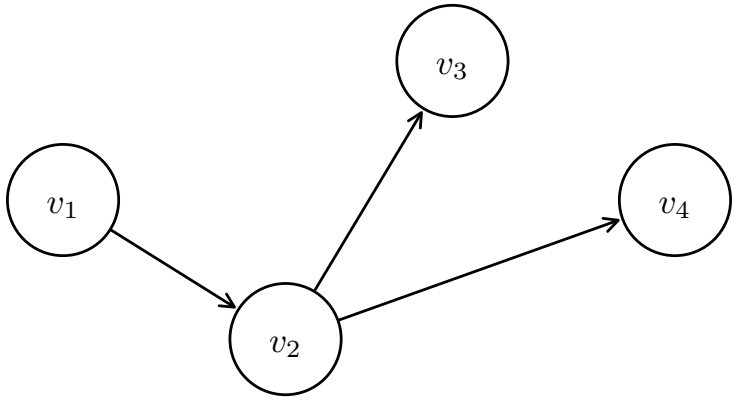


FIGURE 1.5.1 – Arbre de corrélation possible associé au graphe stochastique de la figure 1.2.2

d'un arbre). À chaque itération de CorLCA, le prédécesseur $\dot{v}_i$ du sommet parcouru v_i est retenu comme l'unique sommet parent de v_i dans l'arbre de corrélation.

1.5.1.1 Construction de l'arbre de corrélation

La sélection d'un unique parent pour un sommet donné dans l'arbre de corrélation se fait en déterminant quel prédécesseur a la plus grande influence lors du maximum. Cela signifie que lorsqu'un maximum est réalisé entre au moins deux opérandes, nous souhaitons conserver uniquement l'arc qui impacte le plus fortement le résultat intermédiaire du sommet donné. Ainsi, l'arbre de corrélation représente une structure approximative des corrélations entre chaque résultat intermédiaire.

Lorsqu'un sommet possède plusieurs arcs entrants, l'arc sélectionné est celui dont le résultat intermédiaire est supérieur aux résultats intermédiaires des autres arcs entrants avec la plus grande probabilité. Soit $Y_{ji} \sim \mathcal{N}(\mu_{Y_{ji}}, \sigma_{Y_{ji}})$ et $Y_{j'i} \sim \mathcal{N}(\mu_{Y_{j'i}}, \sigma_{Y_{j'i}})$ deux variables aléatoires suivant chacune une loi normale. Nous supposons que Y_{ji} et $Y_{j'i}$ représentent deux résultats intermédiaires associés à deux arcs dirigés vers le même sommet v_i. Dans la section 1.3.1.2, nous définissons la fonction $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$ et les deux symboles a et b :

$$a = \sqrt{\sigma^2_{Y_{ji}} + \sigma^2_{Y_{j'i}} - 2\sigma_{Y_{ji}}\sigma_{Y_{ji}}\rho_{Y_{ji},Y_{j'i}}}$$
$$b = \frac{\mu_{Y_{ji}} - \mu_{Y_{j'i}}}{a}$$

Nous mentionnons dans la section 1.3.2.3 que la probabilité que Y_{ji} prenne une valeur supérieure à $Y_{j'i}$ est $\Pr[Y_{ji} > Y_{j'i}] = \Phi(b)$. C'est grâce à ce mécanisme que nous sélectionnons le prédécesseur (lignes 11 à 13). En pratique, il suffit de comparer uniquement les espérances de Y_{ji} et $Y_{j'i}$ pour prendre la décision.

Un arbre de corrélation complet pouvant correspondre au graphe stochastique de la figure 1.2.2 est représenté sur la figure 1.5.1. Pour chaque sommet ayant plusieurs arcs entrants, un seul arc est sélectionné.

1.5.1.2 Estimation des coefficients de corrélation

Les équations 1.3.1 et 1.3.2 sont utilisées à la ligne 14 pour calculer le maximum de deux variables aléatoires. Ces équations nécessitent de déterminer au préalable leur corrélation (ce qui est fait à la ligne 10).

L'arbre de corrélation rend possible l'estimation rapide des corrélations entre chaque paire de résultats intermédiaires. En trouvant le plus proche ancêtre commun de deux sommets, il est possible de calculer directement la corrélation entre les résultats intermédiaires associés à ces deux sommets. Soit Y_{ji} et $Y_{j'i}$ deux résultats intermédiaires associés à deux arcs dirigés vers le même sommet v_i. Soit v_k le plus proche ancêtre commun des sommets v_j et $v_{j'}$ dans l'arbre de corrélation. Son résultat intermédiaire est dénoté Y_k. Notre approximation consiste à considérer que $Y_{ji} = \eta + Y_k$ et $Y_{j'i} = \varepsilon + Y_k$ où η et ε sont des variables aléatoires indépendantes de Y_k. Les variables aléatoires η et ε sont indépendantes car elles représentent les sommes des poids entre le sommet v_k et les sommets v_j et $v_{j'}$, respectivement. Ainsi, la corrélation entre Y_{ji} et $Y_{j'i}$ s'exprime

$$\rho_{Y_{ji},Y_{j'i}} = \frac{\sigma^2_{Y_k}}{\sigma_{Y_{ji}}\sigma_{Y_{j'i}}}$$

Il s'agit néanmoins d'une estimation car les sommets v_j et $v_{j'}$ peuvent avoir plusieurs plus proches ancêtres communs dans un graphe orienté acyclique.

1.5.1.3 Complexité

La complexité de CorLCA dépend du coût de la méthode utilisée pour rechercher le plus proche ancêtre commun de deux sommets dans un arbre dans lequel les sommets sont ajoutés incrémentalement. Soit λ (resp., υ) le coût en temps (resp., mémoire) requis pour insérer les sommets et pour réaliser les recherches dans l'arbre. Alors, la complexité en temps de l'approche est $O(m\lambda)$ et la complexité en mémoire est $O(n + \nu)$.

Cole et Hariharan [44] présentent une méthode qui réalise l'insertion des sommets et les recherches des plus proches ancêtres communs en temps constant si les insertions doublent au plus la taille de l'arbre. Cette supposition ne tient pas dans notre cas et les structures de données auraient besoin d'être reconstruites périodiquement avec leur méthode. D'autre part, ils considèrent l'insertion de sommet à l'intérieur de l'arbre et la suppression des sommets, ce que CorLCA ne nécessite pas. Gabow [69] décrivent un algorithme qui réalise l'insertion de feuilles en temps amorti constant.

Le problème consiste à alterner des recherches de plus proche ancêtre commun dans un arbre avec des insertions de feuille dans ce même arbre. Il s'agit d'un problème spécifique et la littérature ne permet pas de statuer sur le coût optimal nécessaire pour le résoudre. D'après les travaux présentés dans le paragraphe précédent cependant, nous pouvons conjecturer que $\lambda = O(1)$ et $\nu = O(n)$, ce qui conduirait à un coût temporel en $O(m)$ et à un coût spatial en $O(n)$ pour CorLCA.

1.5.2 Cordyn

Cette seconde heuristique, nommée *Cordyn* (Correlation based on a dynamic programming approach), prend également en compte les dépendances induites par la reconvergence des chemins. Pour déterminer les coefficients de corrélation requis pour appliquer les formules de Clark, une méthode de programmation dynamique est utilisée. Bien que le coût de Cordyn soit plus élevé que celui de CorLCA, la détermination des coefficients de corrélation est plus précise. En effet, aucune autre approximation que la supposition de la normalité n'est faite. De ce point de vue, les coefficients de corrélation entre les opérandes des maximums sont précisément déterminés.

Algorithme 1.2 L'heuristique Cordyn basée sur une méthode de programmation dynamique pour déterminer les corrélations entre deux opérandes

Entrée: $G = (V, E, \mathcal{X})$ {Graphe stochastique}
Sortie: (μ, σ^2) {Espérance et variance de la distribution de G}
1: **pour chaque** $v_i \in V$ **faire** {Parcourt les sommets dans un ordre topologique}
2: **pour chaque** $v_j \in \mathrm{Pred}(v_i)$ **faire** {Parcourt les prédécesseurs de v_i par indice croissant}
3: $Y_{ji} = X_{ji} + Y_j$ {Équations 1.3.4 et 1.3.5}
4: calculer $(\rho_{Y_{ji},Y_k})_{1 \leq k < i}$ {Équation 1.3.6 et matrice P}
5: calculer $(\rho_{Y_{ji},Y_{j'i}})_{v_{j'} \in \mathrm{Pred}(v_i), j' < j}$ {Équation 1.3.6, matrice P et ligne 4}
6: **fin pour**
7: $\eta = \max_{v_j \in \mathrm{Pred}(v_i)}(Y_{ji})$ {Équations 1.3.1 et 1.3.2, et ligne 5}
8: $Y_i = X_i + \eta$ {Équations 1.3.4 et 1.3.5}
9: calculer $(\rho_{\eta,Y_k})_{1 \leq k < i}$ {Équation 1.3.3 et ligne 4}
10: calculer $(\rho_{Y_i,Y_k})_{1 \leq k < i}$ et compléter P {Équation 1.3.6 et ligne 9}
11: **fin pour**
12: **retourner** $(\mu_{Y_n}, \sigma^2_{Y_n})$

1.5.2.1 Algorithme

Le principe de l'algorithme réside dans la caractérisation continue de l'ensemble des coefficients de corrélation qui pourraient être requis lors de l'évaluation d'une opération de maximum avec les équations 1.3.1 et 1.3.2. Bien qu'il soit toujours possible de déterminer récursivement un coefficient de corrélation quelconque (par une approche descendante avec l'équation 1.3.3), certains des coefficients calculés sont réutilisés et il est sous-optimal de devoir les recalculer. Comme le problème levé par la détermination de ces coefficients exhibe des sous-problèmes reliés les uns aux autres, nous proposons une approche de programmation dynamique. Ainsi, à chaque nouveau sommet v_i considéré, tous les coefficients de corrélations ρ_{Y_i,Y_j} sont calculés et conservés dans une matrice carré symétrique $P = (\rho_{Y_i,Y_j})_{1 \leq i \leq n, 1 \leq j \leq n}$ de dimension n.

L'algorithme 1.2 décrit la boucle principale de Cordyn qui parcourt les sommets dans un ordre topologique par indice de sommet croissant. L'évaluation des résultats intermédiaires a lieu aux lignes 3, 7 et 8 en utilisant les coefficients de corrélation stockés dans P. À chaque itération, la matrice P est complétée avec les coefficients de corrélation nouvellement calculés (ligne 10). Les lignes 4, 9 et 10 servent à calculer les coefficients de corrélation entre une variable aléatoire donnée et chacun des résultats intermédiaires associés aux sommets déjà visités (i.e., toutes les variables aléatoires de l'ensemble $\{Y_k : 1 \leq k < i\}$ si v_i est le sommet parcouru par la boucle). Les coefficients calculés à la ligne 4 sont nécessaires uniquement pour les calculs aux lignes 5 et 9 qui servent eux-mêmes aux lignes 7 et 10, respectivement. En revanche, les corrélations déterminées à la ligne 10 servent potentiellement lors des itérations suivantes à la ligne 4. C'est la raison pour laquelle la matrice P est mise à jour avec les valeurs obtenues.

Nous terminons la description de Cordyn sur quelques remarques. Lorsque le nombre d'arcs entrants de v_i est supérieur à deux, les opérandes à la ligne 7 sont regroupées par paires et les formules de Clark sont successivement utilisées jusqu'à l'obtention du résultat recherché (il faut alors adapter le calcul des coefficients de corrélation à la ligne 9). La seconde remarque porte sur le fait qu'un résultat intermédiaire peut être ignoré lors du calcul des corrélations dès lors que tous les successeurs du sommet correspondant ont été parcourus. En effet, aux lignes 4, 9 et 10, nous pouvons réduire l'ensemble des résultats intermédiaires considérés à ceux correspondant aux sommets $\varphi(v_i)$, où $\varphi(v_i)$ est l'ensemble des sommets v_j tel que $j < i$ et tel que $\exists (v_j, v_k) \in E, k > i$. Ainsi, l'ordre topologique dans lequel les sommets sont parcourus a un impact sur l'efficacité en

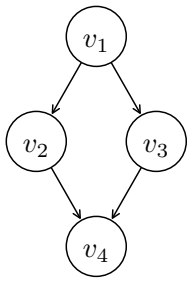
temps et en mémoire de la méthode sans pour autant que la qualité du résultat produit ne change. Enfin, cette approche se généralise si les poids du graphe stochastique sont dépendants. Dans ce cas, il faut étendre le calcul des corrélations aux coefficients de corrélation entre Y et chaque poids $X \in \mathcal{X}$. La complexité de l'approche reste alors asymptotiquement la même.

1.5.2.2 Complexité

Pour déterminer la complexité de l'heuristique Cordyn, nous introduisons et rappelons quelques notations : soit $\deg^-(v)$ le nombre d'arcs entrants du sommet v, $n = |V|$ le nombre de sommets et $m = |E|$ le nombre d'arcs. L'étape la plus coûteuse consiste à caractériser les corrélations entre les résultats des sommes et tous les résultats intermédiaires obtenues (ligne 4). Rappelons que les sommets sont parcourus par indice croissant. À l'étape i, déterminer les coefficients de corrélation entre chaque résultat intermédiaire précédent $(Y_k)_{1 \leq k < i}$ et Y_{ji} coûte $O(i)$ opérations et est répété $\deg^-(v_i)$ fois pour chaque prédécesseur de v_i. La complexité temporelle de l'approche est donc en $O(\sum_{i=1}^{n}(i \times \deg^-(v_i))) = O(nm)$. De plus, il faut stocker $O(m^2)$ éléments dans la matrice P.

1.5.3 Exemple défavorable

La figure 1.5.2 illustre toutes les étapes de l'évaluation de la distribution d'un graphe stochastique composé de quatre sommets. Les poids des arcs sont nuls et ceux des sommets suivent des lois exponentielles d'espérance unitaire. Comme les lois exponentielles différent significativement des lois normales, il s'agit d'une situation défavorable pour les approches basées sur la supposition de la normalité. L'heuristique CorLCA améliore toutefois l'approche de Sculli qui ignore la dépendance entre les résultats intermédiaires Y_2 et Y_3. Ces deux variables aléatoires sont les opérandes du maximum réalisé pour calculer le résultat final Y_4. Pour ce graphe stochastique, les méthodes CorLCA et Cordyn produisent le même résultat car la structure du graphe est simple et chaque paire de sommets ne possède qu'un seul plus proche ancêtre commun.



Résultats intermédiaires	Sculli [131]		CorLCA		Monte Carlo	
	μ	σ^2	μ	σ^2	μ	σ^2
Y_1	1	1	1	1	1	1
Y_2	2	2	2	2	2	2
Y_3	2	2	2	2	2	2
Y_4	3.80	2.36	3.56	2.68	3.50	3.25

FIGURE 1.5.2 – Résultats intermédiaires pour un graphe de quatre sommets dont les poids suivent des lois exponentielles d'espérance 1. Les heuristiques CorLCA et Cordyn donnent le même résultat pour ce graphe stochastique.

1.6 Validation empirique

1.6.1 Implémentation

Afin d'évaluer la qualité des heuristiques CorLCA et Cordyn, nous avons développé une plateforme logicielle, *Emapse* (Evaluation of MAx-Plus Stochastic Expression), pour l'évaluation de graphes stochastiques. Cette plateforme implémente de façon générique quelques méthodes

de la littérature, ce qui permet de confronter empiriquement nos méthodes à celles existantes sur des instances données. Elle représente environ 7 400 lignes de code en langage Java et a été développée en collaboration avec Ihab El Alami.

1.6.1.1 Entrées et sorties

Emapse gère plusieurs types de graphes orientés acycliques, à savoir ceux provenant de projets (dans le cadre de la gestion de projets), ceux provenant de graphes de tâches, ou encore ceux issues de circuits digitaux. La structure d'un graphe stochastique peut donc provenir de multiples sources.

Concernant l'aspect aléatoire, plusieurs lois sont disponibles pour définir les poids du graphe stochastique : uniforme, normale, exponentielle, bêta, gamma, Dirac, Weibull, triangulaire. À la fois leur densité de probabilité et leur fonction de répartition sont implémentées (à l'exception de la fonction de répartition de la loi bêta).

Une fois qu'une méthode produit une variable aléatoire pour un graphe stochastique donné, il est possible de caractériser plusieurs propriétés sur le résultat (espérance, écart type et asymétrie) et de la comparer à une autre variable aléatoire par l'intermédiaire de l'une des quatre mesures implémentées (il s'agit des mesures d'erreur de nature probabiliste ou statistique décrites dans la section 1.6.3.1).

Il est aussi possible de représenter des expressions arithmétiques générales sur des variables aléatoires. Certaines méthodes ne sont pas spécifiques aux graphes stochastiques (comme la méthode de Monte Carlo et l'heuristique Cordyn qui se généralise pour traiter les expressions) et il est alors possible de les utiliser.

1.6.1.2 Méthodes

La figure 1.6.1 dresse un bilan des principales méthodes implémentées. Nous séparons la façon dont les dépendances sont gérées des méthodes utilisées pour évaluer les opérations arithmétiques. En effet, il est possible de supposer la normalité (section 1.3.2.2) tout en procédant à des réductions séries-parallèles (section 1.3.2.1). Inversement, l'approche de Sculli (section 1.3.2.2) n'est pas antagoniste avec les méthodes numériques (section 1.3.1.1). Les combinaisons entre ces deux types de méthodes ne sont cependant pas possibles pour l'approche canonique (section 1.3.2.3) et la méthode de Monte Carlo (section 1.3.4).

1.6.1.3 Aspect numériques

Les calculs numériques induits par les méthodes décrites dans la section 1.3.1.1 présentent quelques écueils. Rappelons qu'une somme implique une convolution dont nous évitons le coût quadratique en manipulant les densités de probabilité dans le domaine fréquentiel. Lorsque deux fonctions sont définies sur des intervalles de tailles différentes, une interpolation doit précéder la transformée de Fourier, ce qui peut conduire à une perte de précision conséquente ou une augmentation du nombre de valeurs considérable (en fonction de la stratégie adoptée). Nous utilisons donc la méthode *OverLap-Add* décrite dans [140] qui réalise efficacement la convolution de façon fragmentée lorsque les densités de probabilité sont définies sur des intervalles de tailles différentes.

Enfin, la génération reproductible de valeurs aléatoires est antinomique. Aussi, nous utilisons le Mersenne Twister [115] comme générateur pseudo-aléatoire pour la méthode de Monte Carlo.

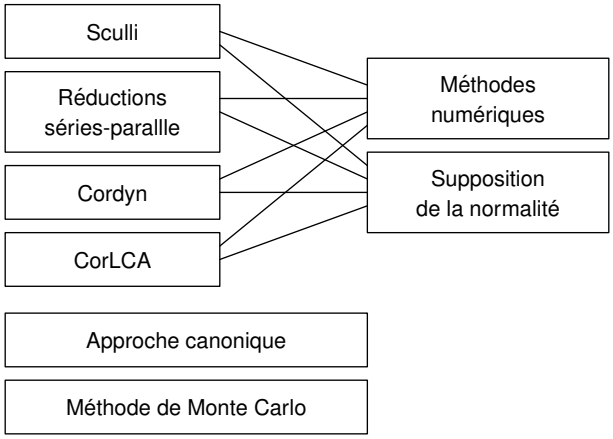


FIGURE 1.6.1 – Méthodes implémentées dans Emapse. Les différentes approches pour traiter la dépendance entre les résultats intermédiaires sont positionnées à gauche. Les méthodes pour évaluer les opérations arithmétiques sont à droite.

1.6.2 Instances

Pour valider empiriquement nos méthodes, nous utilisons un jeu d'instances basé sur des instances existantes. Ces instances qui servent de base aux nôtres définissent la structure des graphes et des poids déterministes. Chaque poids est donc remplacé par une variable aléatoire dont l'espérance est la valeur du poids déterministe. Nous décrivons par la suite la méthode utilisée pour déterminer l'écart type et la loi de chaque variable aléatoire.

La structure des graphes provient des trois jeux d'instances suivant :

- Les instances RCPSP de la bibliothèque PSPLIB [99] dans le cadre de la gestion de projet. Les graphes sont classés suivant quatre tailles : 30, 40, 60, 90. Il y a 48 instances pour chacune des trois premières classes et 60 pour la dernière.
- Le jeu STG [142] pour les graphes de tâches. Nous en retenons 24 instances (celles numérotées par un multiple de dix) par taille distincte : 50, 100, 300, 500, 750, 1000, 1250 et 1500.
- ISCAS-85 [81] et ISCAS'89 [34] pour les circuits digitaux (45 instances).

L'espérance de chaque poids est déterminée par ces instances (sauf pour les circuits digitaux où les poids ne sont pas définis, chaque espérance étant alors fixée à 1).

La structure du graphe et l'espérance de chaque poids étant spécifiées par ces instances déterministes, nous les probabilisons en déterminant une loi et un écart type pour chaque poids. Dans un graphe stochastique donné, tous les poids suivent le même type de lois (s'il s'agit de lois bêtas, alors nous précisons que les paramètres sont $\alpha = 2$ et $\beta = 5$).

Il reste donc à spécifier l'écart type de chaque poids pour aboutir à une instance complète d'un graphe stochastique (sauf si la loi exponentielle est choisie, car elle ne possède qu'un seul paramètre déjà déterminé par l'espérance). Plutôt que de manipuler des écarts types, nous utilisons une mesure de dispersion relative, à savoir le coefficient de variation. Il s'agit du rapport entre l'écart type et la moyenne d'un ensemble de valeurs (voir [129, Section 5.3.2.3]). Il faut donc caractériser le coefficient de variation associé à chaque poids. Pour cela, nous utilisons une loi gamma avec une espérance et un écart type donnés, et pour chaque poids d'un graphe, nous tirons une valeur suivant cette loi pour fixer son coefficient de variation. L'écart type des coefficients de variation d'un graphe vaut 10% de l'espérance de ces mêmes coefficients de variation (comme ce rapport reste constant, nous pouvons dire que nous fixons à $0,1$ le coefficient de variation des coefficients de variation d'un graphe).

Les $48 \times 3 + 60 = 204$ instances PSPLIB, les $24 \times 8 = 192$ instances STG et les 45 ins-

tances ISCAS sont générées avec les valeurs par défaut données par le tableau 1.3. De plus, un sous-ensemble de ces instances (48 instances PSPLIB, 24 instances STG et toutes les instances ISCAS) sont utilisées avec les valeurs testées. Pour chaque valeur testée, l'autre paramètre prend sa valeur par défaut. Nous aboutissons ainsi à un total de $(204+192+45)+(48+24+45)\times 7 = 1260$ graphes stochastiques.

Paramètre	Valeur par défaut	Valeurs testées
Loi	uniforme	normale, exponentielle, bêta, Weibull
Espérance du coefficient de variation	0,1	0,01, 0,03, 0,3

TABLE 1.3 – Paramètres et valeurs pour la génération d'un graphe stochastique

1.6.3 Mesures d'erreur

Pour toutes les instances décrites, douze méthodes sont exécutées parmi lesquelles la méthode de Monte Carlo avec un million de tirages. Le résultat de cette méthode nous sert de résultat de référence pour chaque instance (l'écart entre ce résultat et le vrai résultat du graphe stochastique est analysé dans la section 1.3.4).

1.6.3.1 Description des mesures

Il existe plusieurs manières de quantifier l'erreur commise par chaque méthode en comparant le résultat qu'elles produisent au résultat de référence. Nous étudions ainsi plusieurs mesures d'erreur.

Il est possible de dériver trois types d'erreurs relatives : sur l'espérance, sur l'écart type et sur l'asymétrie. Par exemple, l'erreur relative commise sur l'espérance se note $\left|\frac{\hat{\mu}-\mu}{\mu}\right|$ où μ est la moyenne des valeurs obtenues par la méthode de Monte Carlo et $\hat{\mu}$ l'estimation de l'espérance produite par une méthode donnée.

Outre ces trois mesures directes, la théorie des probabilités et des statistiques fournit quatre autres mesures. Soit F la fonction de répartition empirique obtenu via la méthode de Monte Carlo et $\hat{F}$ la fonction de répartition estimée par une méthode donnée.

Statistique d'Anderson-Darling $A^2 = -T - S$ où $S = \frac{1}{T}\sum_{k=1}^{T}(2k-1)\times(\ln(\hat{F}(X_k)) + \ln(1-\hat{F}(X_{T+1+k})))$, T est le nombre de tirages dans la méthode de Monte Carlo et X_k est le $k^{\text{ième}}$ élément de la liste des valeurs ordonnées produites par cette même méthode [139]. Son ensemble image est $[0;\infty[$. Si les domaines de définition des deux variables aléatoires à comparer sont différents, alors cette méthode peut induire de calculer le logarithme de zéro, ce qui aboutit à un échec.

Statistique de Kolmogorov-Smirnov $D = \sup_x |F(x) - \hat{F}(x)|$ [129, Section 14.6.2.2]. Son ensemble image est $[0;1]$.

Statistique de Cramér-von-Mises $\omega^2 = \frac{1}{n}\int (F(x)-\hat{F}(x))^2 d\hat{F}(x)$ [129, Section 14.6.2.3]. Son ensemble image est $[0;\frac{1}{3}]$.

Distance de Hellinger $H = \sqrt{(1-BC)}$ où $BC = \int_x \sqrt{f(x)\times\hat{f}(x)}dx$ est le coefficient de Bhattacharyya [27]. Cette mesure nécessite les densités de probabilité des variables aléatoires, ce qui requiert une estimation numérique pour la méthode de Monte Carlo. Son ensemble image est $[0;1]$.

Pour chacune de ces quatre mesures, une valeur élevée signifie que les deux variables aléatoires sont significativement différentes, tandis qu'une valeur proche de zéro indique qu'elles sont similaires.

1.6.3.2 Comparaison empirique

La section précédente présente trois mesures d'erreur de type relative (erreur relative de l'espérance, de l'écart type et de l'asymétrie) ainsi que quatre mesures de nature probabiliste ou statistique (la distance de Hellinger et trois statistiques). Nous les comparons dans cette section.

La figure 1.6.2 montre les corrélations entre les mesures d'erreur lorsqu'on les utilise pour quantifier la précision de onze méthodes sur toutes les instances générées. Sur cette figure, les sept mesures décrites dans la section 1.6.3.1 sont comparées les unes aux autres. Nous obtenons donc une matrice symétrique de 7×7 graphiques et seulement 21 d'entre eux sont représentés dans la partie inférieure-gauche. Le nom de chaque mesure est donné sur la diagonale ainsi que l'histogramme des valeurs pour chacune.

La partie supérieure-droite de la matrice contient les coefficients de corrélation de Spearman des mesures spécifiées par la ligne et la colonne. Plus une corrélation est forte, plus la valeur absolue du coefficient Spearman est proche de 1. Ce test de corrélation est plus robuste que le coefficient de Pearson dans le cas où les relations entre les mesures ne sont pas linéaires.

De façon générale, toutes les mesures sont liées à l'exception de l'erreur relative de l'asymétrie. Nous postulons donc que les six mesures concernées sont toutes pertinentes pour quantifier l'erreur, mais chacune sur un aspect différent. Comme les erreurs relatives de l'espérance sont fortement concentrées autour de zéro, il est difficile de distinguer la dépendance entre cette mesure et les autres. Cependant, les fortes valeurs des coefficients de Spearman qui y sont relatifs (entre 0,88 et 0,90, l'erreur relative de l'asymétrie mise à part) indiquent que lorsque les autres mesures d'erreur sont élevées, alors celle-ci tend à l'être aussi, et réciproquement.

Nous remarquons aussi que les quatre mesures de nature probabiliste ou statistique sont toutes quasiment équivalentes. En effet, les coefficients de corrélation entre la statistique d'Anderson-Darling, de Kolmogorov-Smirnov et de Cramér-von-Mises sont tous égaux à $1,00$. Seule la distance de Hellinger se distingue légèrement.

Comme ces quatre mesures prennent en compte l'entièreté de la distribution dans leurs calculs, nous les préférons aux deux premières mesures d'erreur relative. Parmi ces mesures, la statistique de Kolmogorov-Smirnov et la distance de Hellinger présentent deux avantages : leurs ensembles images sont mieux définis (i.e., il s'agit de $[0;1]$ dans les deux cas) et les valeurs mesurées sont mieux réparties dans cet intervalle. D'autre part, la statistique de Kolmogorov-Smirnov s'applique directement sur les fonctions de répartition et la distribution de Kolmogorov nous permet d'affirmer que le calcul de cette statistique est précis à $1,16‰$ avec un million de tirages (voir la section 1.3.4).

La mesure d'erreur que nous retenons par la suite est donc la statistique de Kolmogorov-Smirnov.

1.6.4 Qualité des méthodes

Pour évaluer la qualité des heuristiques CorLCA et Cordyn, nous les comparons à trois autres méthodes : l'approche de Sculli (section 1.3.2.2), l'approche par réductions séries-parallèles avec les méthodes numériques (section 1.3.2.1) et l'approche canonique (section 1.3.2.3).

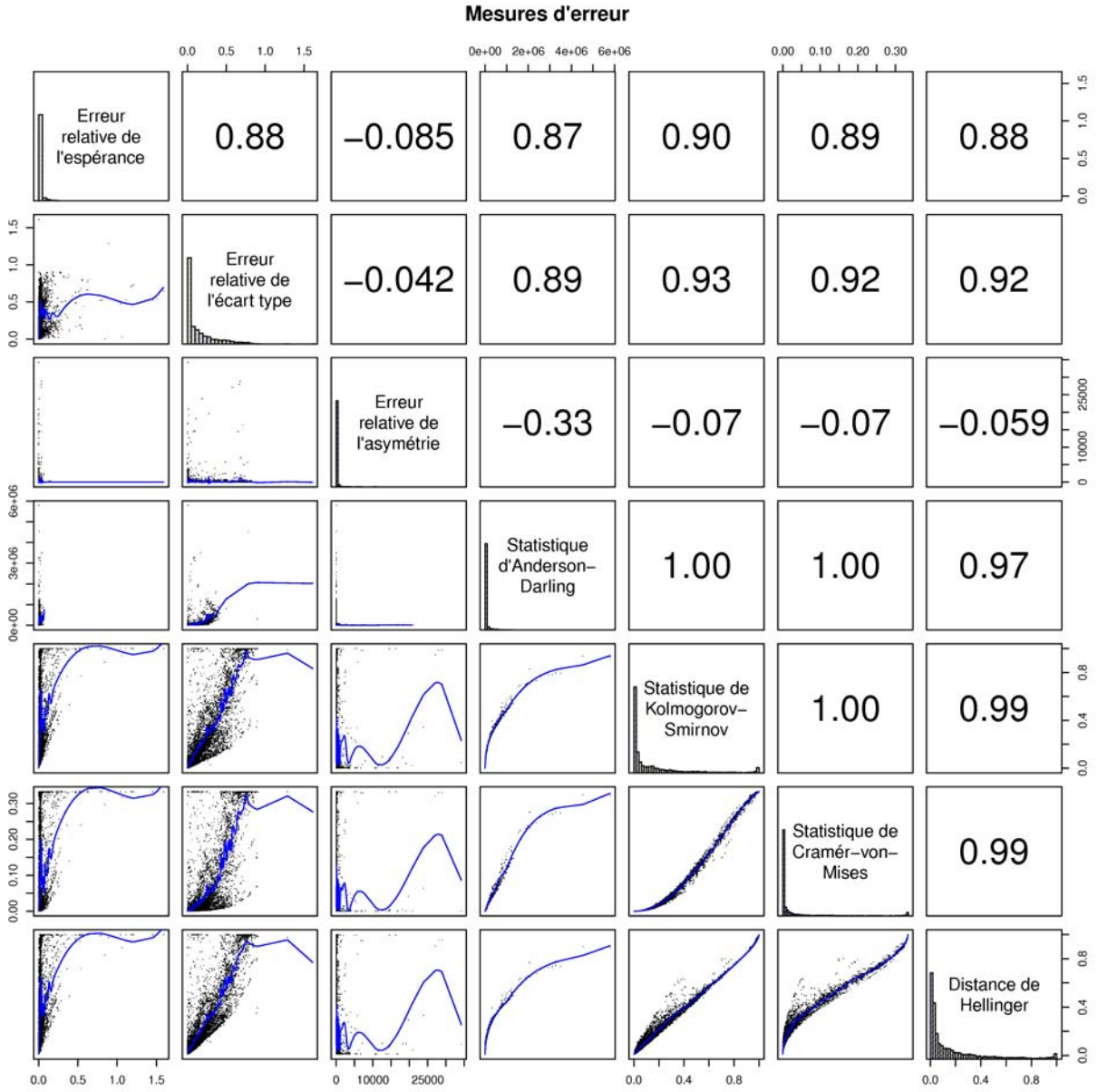


FIGURE 1.6.2 – Corrélations entre les mesures d'erreur appliquées sur onze méthodes pour 1260 graphes stochastiques. Partie inférieure-gauche : graphiques avec les 13 860 mesures (avec une interpolation par spline cubique). Partie supérieure-droite : valeurs des coefficients de Spearman.

1.6.4.1 Comparaison générale

La figure 1.6.3 présente un résumé des qualités respectives obtenues avec les cinq méthodes comparées. Pour chacune des instances générées et pour chaque paire de méthodes (caractérisée par une ligne et une colonne), nous calculons le rapport entre les statistiques de Kolmogorov-Smirnov obtenues pour les deux méthodes (rapport de la ligne par la colonne). Si ce rapport est égal à 1, alors les deux méthodes produisent chacune un résultat identique en terme de précision. Si le rapport est inférieur à 1, alors le résultat de la méthode sur la ligne est meilleur que celui de la méthode sur la colonne. Chaque graphique de la partie inférieure-gauche montre l'histogramme et la fonction de répartition empirique de ces rapports. La partie supérieure-droite résume ces données. Ainsi, nous constatons que les résultats produits par l'approche canonique sont moins précis que ceux produits par CorLCA dans 75,08% des cas et qu'il y a égalité dans 7,78% des cas. Le rapport médian est inférieur à 1 (0,761 en l'occurrence).

Nous pouvons en conclure que les méthodes que nous proposons se comportent favorablement relativement aux trois autres méthodes. Nous constatons aussi que l'approche de Sculli est la plus mauvaise, suivie de près par l'approche par réductions séries-parallèles. L'heuristique Cordyn est la meilleure talonnée de près par CorLCA. L'approche canonique se situe au milieu : elle est à la fois significativement meilleure que les deux premières approches et significativement moins précise que nos deux heuristiques.

1.6.4.2 Effets des paramètres

Pour analyser l'effet des paramètres sur la qualité des méthodes, nous limitons les instances aux graphes stochastiques construits à partir des jeux PSPLIB et STG. Cela permet de simplifier la présentation des résultats.

Nous étudions trois paramètres sur les figures 1.6.4 à 1.6.7 : le nombre de poids dans un graphe stochastique, la loi que suivent ces poids et l'espérance des coefficients de variation de ces poids. La précision des cinq méthodes considérées s'obtient par l'intermédiaire de la statistique de Kolmogorov-Smirnov. Toutes les mesures d'erreur d'une méthode pour des graphes stochastiques vérifiant une modalité d'un paramètre donné sont représentées par une boîte à moustaches. Une boîte à moustaches consiste en un résumé en cinq nombres d'un ensemble de valeurs : la médiane est le trait horizontal gras, la boîte s'étend du premier quartile jusqu'au troisième quartile, et la longueur des moustaches vaut 1,5 fois l'intervalle interquartile.

Sur les figures 1.6.4 et 1.6.5, nous voyons comment évolue la précision de chaque méthode lorsque le nombre de variables aléatoires varie dans les instances. La précision de l'approche de Sculli et celle par réductions séries-parallèles tend à diminuer lorsque la taille des graphes stochastiques augmente avec un changement localisé lorsque la taille prend pour valeur 300. En effet, la médiane des statistiques de Kolmogorov-Smirnov dépasse systématiquement la valeur 0,2, alors qu'elle en était inférieure pour les graphes de taille plus réduite. L'approche canonique affiche un comportement similaire, même si la précision de cette méthode est globalement meilleure. La précision des heuristiques CorLCA et Cordyn est au contraire relativement stable quelque soit la taille de l'instance.

Toutes les approches auxquelles nous confrontons nos heuristiques sont considérablement imprécises pour les graphes stochastiques possédant mille variables aléatoires (les médianes des statistiques de Kolmogorov-Smirnov sont proches du maximum pour les trois approches). Nous faisons l'hypothèse que la précision est impactée par la profondeur critique du graphe, c'est-à-dire le nombre de sommets formant le chemin du sommet source au sommet puits et tel que l'augmentation du poids de l'un des sommets sur ce chemin induit une augmentation du résultat final. Dans le cas stochastique, il s'agit du nombre moyen de ces sommets et nous l'obtenons

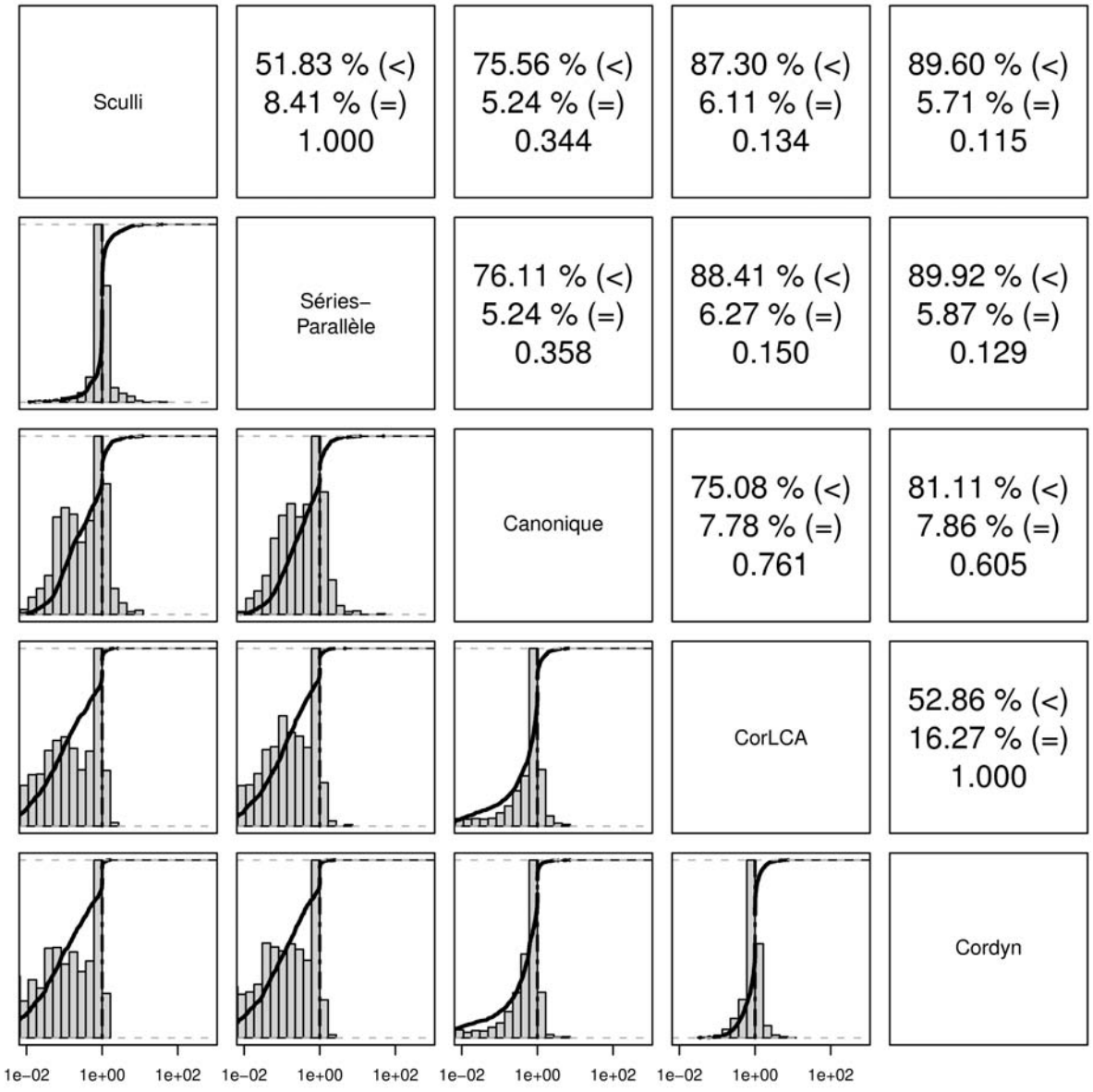


FIGURE 1.6.3 – Comparaison des méthodes par paire sur 1 260 graphes stochastiques en utilisant la statistique de Kolmogorov-Smirnov comme mesure d'erreur. Partie inférieure-gauche : histogrammes et fonctions de répartition empiriques des rapports entre les mesures d'erreur de chaque méthode. Partie supérieure-droite : proportions des rapports supérieurs et égales à 1 et médianes des rapports.

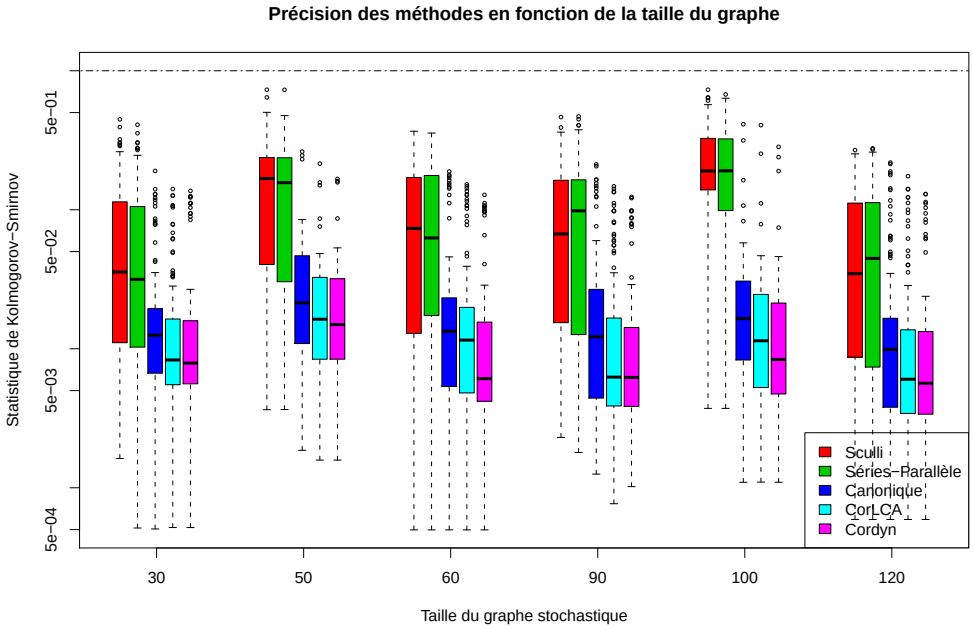


FIGURE 1.6.4 – Précision de cinq méthodes en fonction du nombre de variables aléatoires dans le graphe stochastique (de 30 à 120)

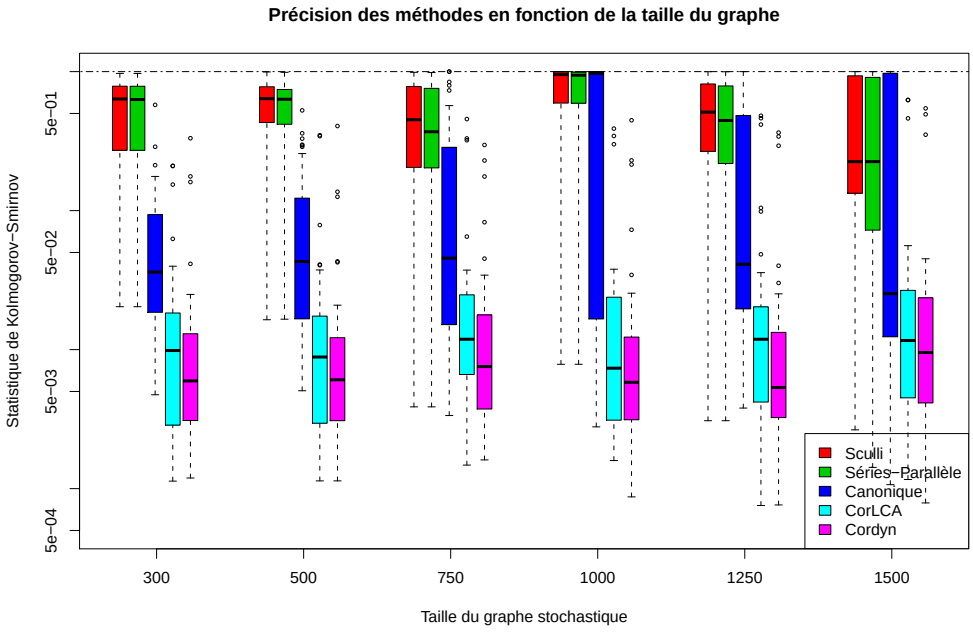


FIGURE 1.6.5 – Précision de cinq méthodes en fonction du nombre de variables aléatoires dans le graphe stochastique (de 300 à 1500)

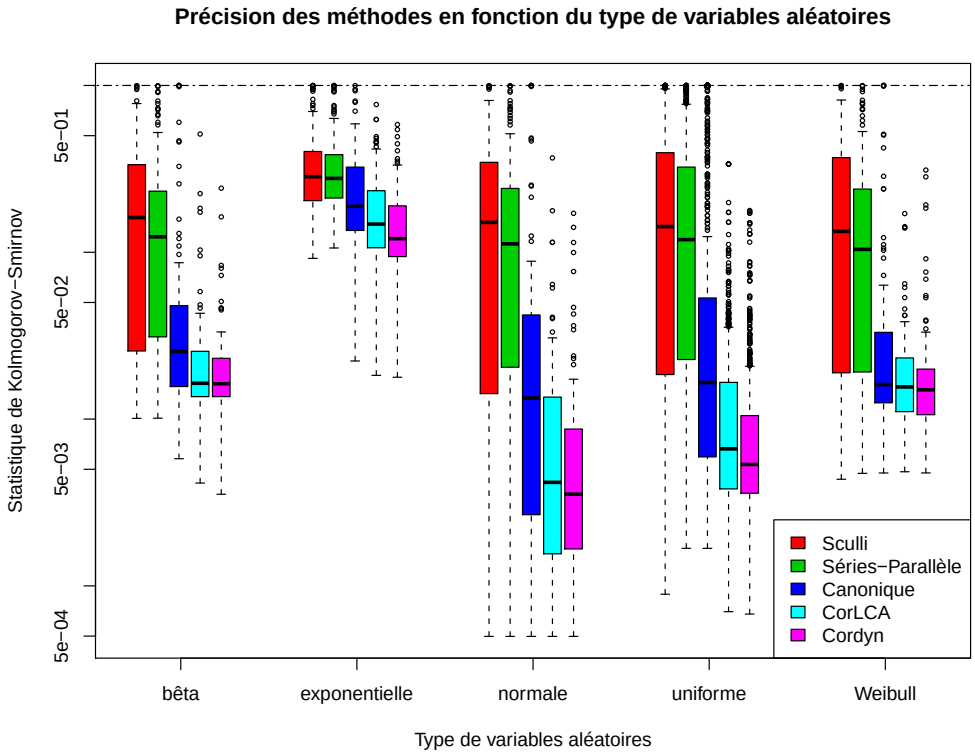


FIGURE 1.6.6 – Précision de cinq méthodes en fonction de la loi que suivent les poids d'un graphe stochastique

grâce à la méthode de Monte Carlo. Le coefficient de corrélation de Pearson entre la profondeur critique des graphes stochastiques et la précision de l'approche canonique vaut $0,77$, ce qui est élevé mais peu concluant. Ce phénomène mérite de faire l'objet de travaux futur.

L'effet de la loi de probabilité choisie pour l'ensemble de chaque graphe stochastique est montré sur la figure 1.6.6. La loi exponentielle est celle qui pose le plus de difficultés pour les méthodes (même pour l'approche par réductions séries-parallèles qui utilise les méthodes numériques). C'est en effet la loi qui diffère le plus de la loi normale parmi celles testées. De plus, sa discrétisation régulière est difficile à cause de sa forte décroissante à l'origine et de sa longue queue, ce qui impose de faire un choix antagoniste entre un faible intervalle de discrétisation et un domaine de définition large. Les lois normales et uniformes sont au contraire les plus favorables pour nos deux heuristiques et l'approche canonique (les deux autres méthodes se comportant similairement pour toutes les lois autres que la loi exponentielle).

Enfin, nous représentons sur la figure 1.6.7 l'effet du coefficient de variation sur la précision. Nous remarquons une tendance générale, partagée par toutes les méthodes, à perdre en précision lorsque le coefficient de variation augmente. Nous l'expliquons par la simplicité avec laquelle s'évalue une opération de maximum lorsque ses opérandes sont définies sur des intervalles qui ne se recouvrent pas. Dans ce cas, l'opérande qui domine l'autre constitue directement le résultat de l'opération. L'augmentation des coefficients de variation au sein d'un graphe diminue le nombre de situations ou la méthode directe précédemment décrite peut être appliquée. Ainsi, augmenter l'incertitude dans un graphe amplifie l'impact des erreurs dues aux maxima.

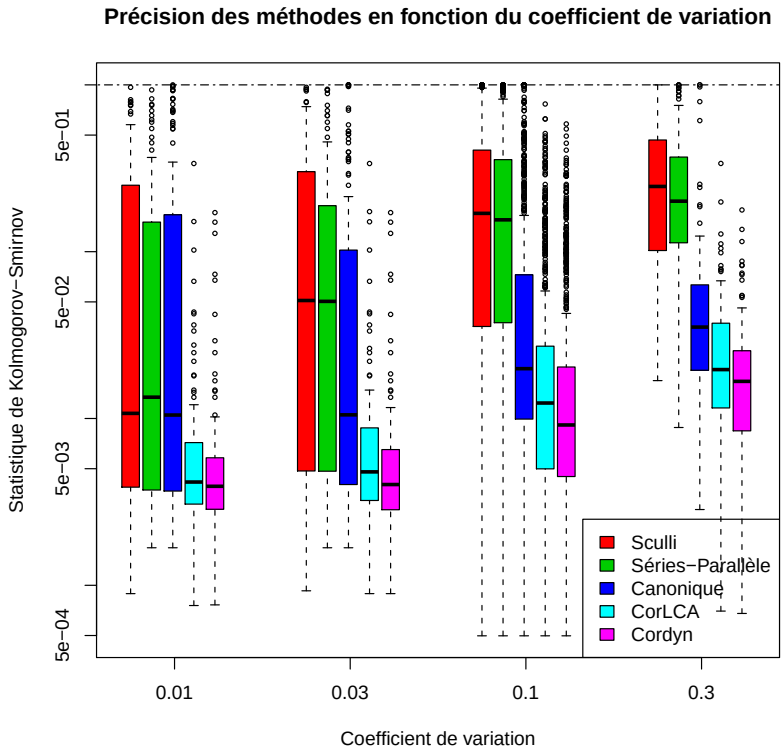


FIGURE 1.6.7 – Précision de cinq méthodes en fonction de l'espérance des coefficients de variation des poids d'un graphe stochastique

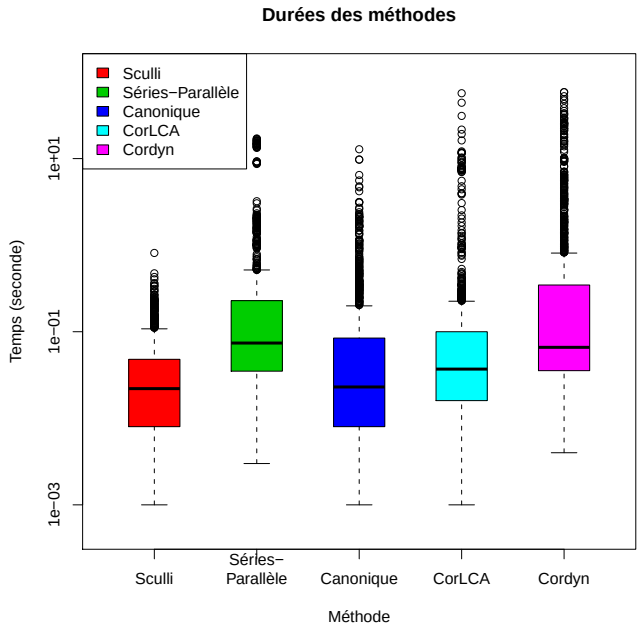


FIGURE 1.6.8 – Temps de calcul de cinq méthodes

1.6.5 Temps de calcul

La figure 1.6.8 présente l'ensemble des temps de calcul groupés par méthode. L'approche de Sculli et l'approche canonique sont les deux méthodes les plus rapides, la seconde étant légèrement moins rapide. L'approche canonique constitue ainsi un compromis rapidité/précision intéressant. L'heuristique CorLCA est la troisième méthode en terme de vitesse. Cette figure ne permet pas de comparer entre elles les deux dernières méthodes (l'approche par réductions séries-parallèles et Cordyn). Celles-ci sont toutefois les plus lentes.

Cette figure résume tous les temps de calcul indépendamment du nombre de poids dans les graphes stochastiques. Bien qu'il existe une relation entre la vitesse et la taille du graphe, l'étude de cette relation n'apporte pas de conclusions supplémentaires par rapport à celles que nous tirons grâce à cette figure.

Comme la complexité de CorLCA est supposée être la même que celle de l'approche de Sculli, leurs vitesses devraient être similaires. Cependant, ce n'est pas le cas car la méthode implémentée pour rechercher un plus proche ancêtre commun n'a pas un coût constant. Cela défavorise l'heuristique CorLCA en terme de temps de calcul (la plateforme Emapse n'a d'ailleurs pas pour objectif principal d'être optimale en terme de vitesse mais d'implémenter correctement des méthodes représentatives de la littérature).

Dans tous les cas, l'approche par réductions séries-parallèles est dominée par l'approche canonique à la fois en terme de vitesse et en terme de qualité. Cette dominance est suffisament importante pour en conclure que l'approche par réductions séries-parallèles n'est pas une solution efficace pour notre problème. Les quatre autres approches représentent chacune un compromis rapidité/précision différent.

1.7 Conclusion

Nous proposons un formalisme général qui couvre le problème fondamental tel qu'il apparaît dans plusieurs domaines. Plus précisément, l'évaluation de la distribution d'un graphe stochastique est le problème qui consiste à caractériser la loi que suit une variable aléatoire définie par une succession de maximums et de sommes de variables aléatoires.

La complexité du problème dans le cas particulier où les variables aléatoires sont discrètes est analysée. Ce problème est #P'-Complet, c'est-à-dire au moins aussi dur que les problèmes NP-Complet. Il est en plus inapproximable à une quantité ou à un facteur près.

En pratique, la difficulté provient des dépendances entre les résultats intermédiaires induites par la reconvergence des chemins. En effet, lorsque les opérations de somme et de maximum ne sont appliquées que sur des variables aléatoires indépendantes, l'évaluation du résultat est numériquement directe.

Deux heuristiques, CorLCA et Cordyn, sont proposées. Elles apportent chacune un nouveau compromis en terme de vitesse et de précision par rapport aux méthodes auxquelles nous les avons confrontées. D'autre part, notre validation empirique compare des méthodes de la littérature entre elles. Ainsi, l'approche par réductions séries-parallèles ne présente pas d'intérêt significatif pour les instances considérées car elle est souvent moins précise et moins rapide que l'approche canonique.

La méthode CorLCA est l'heuristique la plus rapide parmi celles dont la précision reste stable lorsque la taille du graphe stochastique augmente. Elle constitue donc une solution de choix dans le cadre d'heuristiques d'ordonnancement faisant appel fréquemment à un mécanisme d'évaluation.

Il est possible de poursuive ces travaux sur de nombreux aspects : l'étude analytique du problème ; la validation empirique des méthodes ; et, le coût algorithmique des heuristiques proposées.

Concernant ce premier aspect, des méthodes exactes existent lorsque le graphe suit une structure spécifique et que toutes les variables aléatoires suivent une loi particulière. Utiliser une loi close à la fois sous les opérations de maximum ou sous celle de somme permettrait d'obtenir une méthode analytique exacte pour les graphes séries-parallèles.

L'étude empirique s'étendrait facilement pour comparer d'autres méthodes de la littérature. Par ailleurs, les facteurs qui compliquent l'estimation par les différentes approches (notamment l'approche canonique) pourrait être étudiés plus en détail.

Enfin, le coût algorithmique des deux heuristiques proposées mérite d'être approfondi. D'une part, préciser le coût de CorLCA nécessite d'étudier la problématique liée à la recherche d'un plus proche ancêtre commun dans un arbre enraciné. D'autre part, la minimisation du coût de Cordyn en utilisant un ordre topologique optimal est liée aux problèmes de la séparation des sommets et de la somme des coupures [52]. Les problèmes ainsi levés ne dépendent toutefois pas de l'évaluation d'un graphe stochastique.

Chapitre 2

Techniques d'optimisation multicritère pour l'ordonnancement

Sommaire

2.1 Introduction

Comme nous le verrons dans la seconde partie de cette thèse (plus spécifiquement dans le chapitre 3 et dans une moindre mesure dans le chapitre 4), la gestion de l'incertitude en ordonnancement multiplie le nombre d'objectifs et il est alors nécessaire de concevoir des techniques d'optimisation multicritères. Nous proposons des méthodes d'ordonnancement génériques qui s'adaptent aux nouveaux critères que l'on cherche à optimiser en lien avec l'incertitude.

Ce chapitre présente un état de l'art partial sur l'optimisation multicritère combinatoire en se focalisant sur les spécificités liées à l'ordonnancement. Quelques contributions mineures sont également proposées. Les sous-sections comportant des apports incrémentaux aux techniques existantes pour notre problématique sont suivies d'une étoile $\star$ dans le titre.

La section 2.2 traîte du problème général et précise le modèle. La section 2.3 aborde les opérateurs d'agrégation qui réduisent un ensemble de critères en un seul. Ces opérateurs sont utiles pour adapter directement des heuristiques de construction gloutonne au cas multicritère. Enfin, un type de métaheuristiques, les algorithmes évolutionnaires en l'occurrence, est présenté dans la section 2.4.

Nous terminons cette introduction sur quelques remarques terminologiques. Nous utilisons sans distinction le terme « solution » et « ordonnancement » qui sont équivalents dans notre formulation du problème. En conséquence, les méthodes décrites sont spécifiques aux problèmes d'ordonnancement. Certains concepts sont toutefois également applicables dans le cadre de l'optimisation combinatoire. Par ailleurs, nous restreignons la dimension multicritère au cas bicritère qui peuvent parfois se généraliser directement pour un nombre de critères arbitraire. Enfin, il est impératif de distinguer les *décisions* des *solutions*. Il est possible que ces deux concepts soient interchangeables lorsqu'une décision aboutit à une solution. En revanche, avec des algorithmes gloutons, une solution se définit par la succession des décisions prises pour la construire. Dans tous les cas, les décisions et les solutions sont représentables dans l'espace des valeurs objectives que nous définissons par la suite.

2.2 Ordonnancement multicritère

Nous présentons dans cette section les définitions fondamentales de l'optimisation multicritère. Le modèle d'ordonnancement considéré est ensuite décrit. Enfin, nous détaillons des méthodes pour mesurer la qualité d'un ensemble de solutions non-dominées. Les notations utilisées dans ce chapitre sont résumées dans le tableau 2.1.

2.2.1 Optimisation multicritère

Matthias Ehrgott [60] aborde un ensemble de concepts liés à l'optimisation multicritère continue ou combinatoire. Il détaille notamment plusieurs définitions de dominance qui enrichissent dans le cas de l'optimisation continue celles que nous présentons.

L'ordonnancement multicritère consiste à générer un ou plusieurs ordonnancements possédant des caractéristiques optimales relativement à plusieurs critères. Formellement, nous notons $\mathcal{O}$ l'ensemble des ordonnancements admissibles. La fonction objective est notée $f : \mathcal{O} \mapsto \mathbb{R}^d$. Soit $f_i(o)$ la valeur du $i^{\text{ième}}$ objectif pour l'ordonnancement o. Alors, $f(o) = (f_1(o), \ldots, f_d(o))$ est le vecteur des valeurs objectives de l'ordonnancement o. Sans perte de généralité, nous cherchons à minimiser chacun des d objectifs. L'*espace des valeurs objectives* est l'ensemble des vecteurs de valeurs objectives que peuvent prendre les solutions admissibles, i.e., $\{f(o) : \forall o \in \mathcal{O}\}$.

D'après ces définitions, deux solutions peuvent être incomparables. La *dominance de Pareto* formalise le concept d'optimalité dans le cas multicritère. Nous dirons que l'ordonnancement o

Symbole	Définition
$T = \{t_i : i \in [1..n]\}$	Ensemble des tâches
n	Nombre de tâches ($n = \|T\|$)
$G = (T, E)$	Graphe orienté acyclique contenant les tâches et les contraintes de dépendance
$P = \{p_j : j \in [1..m]\}$	Ensemble des processeurs
m	Nombre de processeurs ($m = \|P\|$)
w_i^j	Durée d'exécution de la tâche t_i sur le processeur p_j
$d_{ii'}^{jj'}$	Durée de la communication $(t_i, t_{i'}) \in E$ entre les tâches t_i et $t_{i'}$ exécutées respectivement sur les processeurs p_j et $p_{j'}$
$\text{rang}(t_i)$	Rang de la tâche t_i
$\mathcal{O}$	Ensemble des ordonnancements valides
η	Nombre d'ordonnancements valides ($\eta = \|\mathcal{O}\|$)
O	Ensemble d'ordonnancements non-dominés ($O \subseteq \mathcal{O}$)
$f : \mathcal{O} \mapsto \mathbb{R}^d$	Fonction objective avec d valeurs objectives
$\preceq$	Dominance de Pareto faible
$\prec$	Dominance de Pareto stricte
$I_\epsilon : \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O}) \mapsto \mathbb{R}$	ϵ-indicateur qui compare deux ensembles d'ordonnancements non-dominés
$I_C : \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O}) \mapsto \mathbb{R}$	Indicateur de couverture qui compare deux ensembles d'ordonnancements non-dominés
θ	Paramètre de compromis entre deux critères

TABLE 2.1 – Notations du chapitre

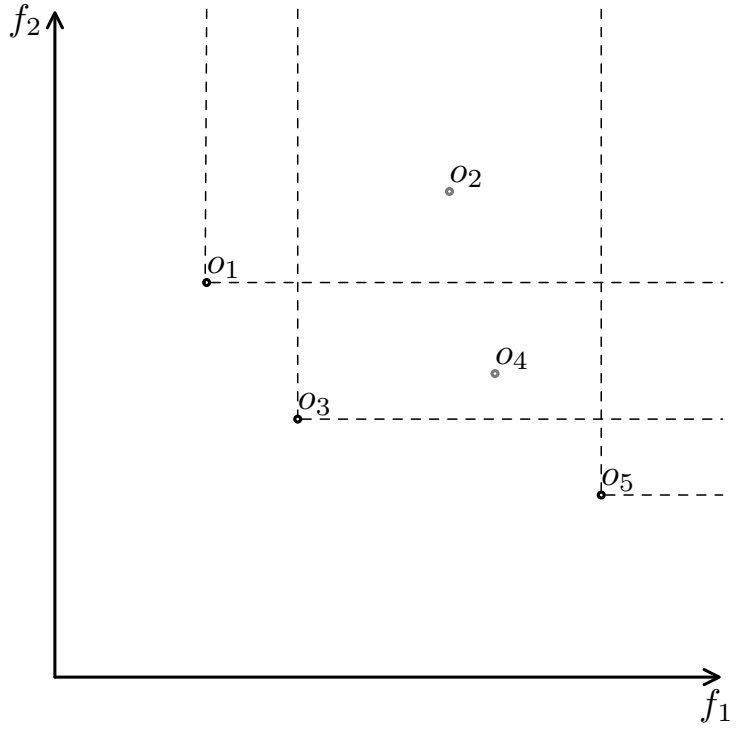


FIGURE 2.2.1 – Exemple d'espace de valeurs objectives en deux dimensions et avec quatre ordonnancements. Comme les valeurs objectives sont à minimiser, les ordonnancements o_2 et o_4 sont dominés.

domine faiblement o', i.e., $o \preceq o'$, si et seulement si $\forall i \in [1..d], f_i(o) \leq f_i(o')$. Par analogie, l'ordonnancement o *domine strictement* o', i.e., $o \prec o'$, si et seulement si $o \preceq o'$ et $o' \npreceq o$. La dominance stricte implique donc la dominance faible. Un ordonnancement o est *Pareto-optimal* s'il n'existe pas d'ordonnancement qui le domine strictement. Sauf mention contraire, la relation de dominance faible est utilisée. Par exemple, sur la figure 2.2.1, nous avons les relations $o_1 \prec o_2$, $o_3 \prec o_2$ et $o_3 \prec o_4$ relativement aux deux fonctions objectives f_1 et f_2. Ces relations sont aussi valables avec la dominance faible car il n'y aucun cas d'égalité. Les ordonnancements Pareto-optimaux sont donc o_1, o_3 et o_5 même si ce dernier ne domine aucun autre ordonnancement.

Comme la relation de dominance de Pareto définit un ordre partiel sur les ordonnancements, plusieurs ordonnancements peuvent être Pareto-optimaux. Le *front de Pareto* contient toutes les solutions Pareto-optimales. La dominance de Pareto s'applique aussi à l'espace des valeurs objectives. Ainsi, nous pouvons dire que le vecteur de valeurs objectives d'une solution Pareto-optimale est Pareto-optimale. Notons enfin qu'un ordonnancement o qui n'est dominé par aucun ordonnancement d'un ensemble $O \subseteq \mathcal{O}$ est dit *non-dominé* relativement à ce même ensemble.

Dans un problème multicritère, nous cherchons donc à générer un front de Pareto ou une *approximation* de cet ensemble.

Parallèlement à l'optimisation multicritère, il est également possible de réaliser une *décision multicritère*. Dans ce cas, une décision ne débouche pas systématiquement sur la génération d'une solution. C'est le cas pour une méthode de construction gloutonne qui consiste en une succession de décisions multicritères et qui aboutit à la génération d'une solution. Des valeurs objectives sont associées à chacune de ces décisions. Nous utilisons alors la même terminologie pour caractériser les dominances entre les décisions que nous le faisons entre les solutions.

2.2.2 Modèle d'ordonnancement

L'ordonnancement multicritère a été considérablement étudié. T'kindt et al. [141] décrivent les avancées générales de ce domaine de recherche. L'approche principale retenue dans leur ouvrage consiste à reformuler les problèmes de façon à ce qu'il ne reste qu'une seule valeur à optimiser. De cette manière, une fois le problème et le compromis entre les critères définis, une seule solution est optimale. Cette solution est par ailleurs Pareto-optimale relativement à l'ensemble des critères du problème initial. Si une autre solution est recherchée, alors le compromis est modifié et la même procédure algorithmique est de nouveau exécutée. À l'inverse, nous cherchons à générer une approximation du front de Pareto, c'est-à-dire un ensemble de solutions dont les valeurs objectives sont proches des solutions Pareto-optimales.

Le problème d'ordonnancement multicritère considéré se note $R|prec|\#(f)$ suivant la notation $\alpha|\beta|\gamma$ [76] et celle décrite dans [141]. Cela signifie qu'une application modélisée par un graphe de tâches orienté et acyclique $G = (T, E)$ doit être ordonnancée sur un ensemble de processeurs hétérogènes P. Précisément, chacune des n tâches contenues dans T doit être assignée à l'un des m processeurs de P. L'ordre dans lequel chaque processeur exécute les tâches qui lui sont assignées doit respecter les contraintes de dépendance définies dans E. D'autre part, l'exécution de la tâche t_i sur le processeur p_j prend w_i^j unités de temps. Enfin, les tâches sont toutes exécutées au plus tôt. Nous postulons que la fonction f permet de mesurer les critères à partir d'un tel ordonnancement et nous présentons des techniques d'optimisation multicritère indépendamment de ces critères.

2.2.3 Quantification de la qualité d'un ensemble de solutions

Que nous considérions un problème multicritère dont le nombre de solutions Pareto-optimales est exponentiel ou une méthode qui génère une approximation du front de Pareto, il est nécessaire de quantifier l'intérêt d'un ensemble de solutions non-dominées pour évaluer la qualité du résultat produit.

De nombreux indicateurs sont proposés et analysés dans la littérature [82, 98, 165]. Les indicateurs unaires s'appliquent à un ensemble de solutions et lui attribuent une valeur numérique correspondant à sa qualité. Les indicateurs binaires donnent une valeur pour une paire d'ensembles de solutions, ce qui permet de comparer deux à deux des ensembles de solutions.

Nous allons détailler trois de ces indicateurs : l'ϵ-indicateur binaire introduit par Zitzler et al. [165] ; l'hypervolume qui est utilisé dans le cadre des algorithmes évolutionnaires dans [164] ; et l'indicateur de couverture également défini dans [164].

L'ϵ-indicateur binaire, noté I_ϵ, s'exprime

$$I_\epsilon(O, O') = \inf_{\epsilon \in \mathbb{R}} \{\forall o' \in O', \exists o \in O, \forall i \in [1..d] : f_i(o) \leq \epsilon \times f_i(o')\}$$

L'indicateur I_ϵ donne donc le facteur ϵ minimal tel que chaque solution de l'ensemble O' dont les valeurs objectives sont multipliées par ϵ est dominée par au moins une solution de O. Cet indicateur binaire n'est pas symétrique, la valeur $I_\epsilon(O', O)$ doit aussi être considérée. Il est établi que si $I_\epsilon(O, O') \leq 1$ et $I_\epsilon(O', O) > 1$, alors l'ensemble O est meilleur que O'. Si les deux valeurs sont soit inférieures à 1, soit supérieures à 1, alors les ensembles sont incomparables.

L'hypervolume est un indicateur unaire qui mesure le volume de l'espace des valeurs objectives où tout point est dominé par un front donné et qui domine un point de référence. Plus ce volume est large, plus le front s'éloigne du point de référence. Si ce dernier est choisi convenablement, alors l'objectif est de maximiser l'hypervolume. La figure 2.2.2 illustre l'hypervolume en grisé pour un problème bicritère. Chaque point correspond ici à une solution et chaque endroit situé dans l'aire en cyan est dominé par une paire de valeurs objectives et domine le point de référence r.

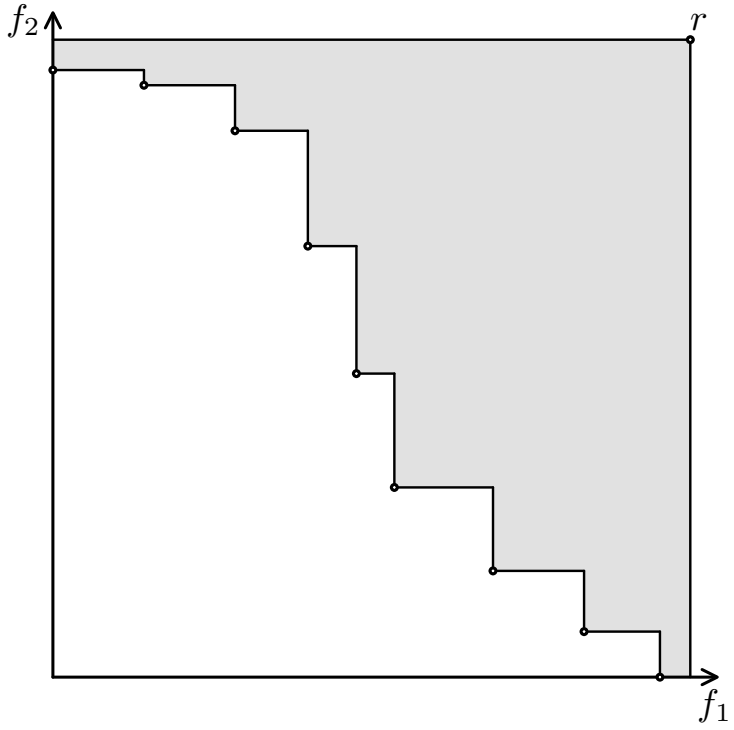


FIGURE 2.2.2 – Hypervolume (en gris) entre un point de référence r et un ensemble de solutions

Nous décrivons enfin l'indicateur de couverture que nous notons I_C et qui se définit par

$$I_C(O, O') = \frac{|\{o' \in O' : \exists o \in O, o \preceq o'\}|}{|O'|}$$

Cet indicateur évalue la proportion des solutions de l'ensemble O' qui sont dominées par au moins une solution de l'ensemble O. Si $I_C(O, O') = 1$ et $I_C(O', O) = 0$, alors l'ensemble O est meilleur que l'ensemble O'.

2.3 Agrégation des critères

Un problème multicritère peut se résoudre en fusionnant un ensemble de valeurs pour se réduire au cas monocritère. Dans certains cas, les solutions algorithmiques monocritères peuvent alors résoudre le problème. Cette méthode agrège les critères et est aussi appelée *scalarisation* dans la littérature [60].

Dans le cas bicritère, l'agrégation repose sur un paramètre de compromis θ prenant une valeur dans l'intervalle $[0; 1]$. À chacune des extrémités de cet intervalle, l'un des deux critères est ignoré. Lorsque l'on considère d critères, il faut alors $d - 1$ paramètres de compromis. Nous nous restreignons dans cette section au cas bicritère.

Un première méthode simple et largement répandue est d'abord présentée, suivie de développements provenant de la théorie de la prise de décision multicritère.

Pour discuter de l'intérêt de ces méthodes, il est nécessaire de définir ce qu'est un *problème convexe*. Informellement, un problème d'optimisation combinatoire est convexe si l'ensemble des solutions ou des décisions qui sont Pareto-optimales sont sur le bord « sud-ouest » de l'enveloppe convexe des valeurs objectives admissibles (nous considérons que le bord de l'enveloppe convexe

des valeurs objectives des solutions admissibles se découpent en quatre régions, la région « sud-ouest » contenant une partie du front de Pareto).

Nous proposons ensuite une méthode pour les problèmes combinatoires. Le principe d'agrégation proposé s'inspire d'approches existantes basées sur une direction au sens géométrique. Ce principe est utilisé conjointement avec un ensemble de techniques qui permettent d'éviter certains cas pathologiques dans le but de réaliser un choix pertinent relativement au paramètre de compromis θ.

Cette méthode d'agrégation peut être utilisée au sein d'une heuristique consistant en une succession de décisions multicritères qui sont ainsi remplacées par des décisions monocritères. Dans ce cas, chaque valeur du paramètre θ conduit à une solution potentiellement nouvelle. Nous proposons une méthode de recherche dichotomique pour générer efficacement un ensemble représentatif de ces solutions.

Remarquons que dans le cas d'une heuristique où les décisions multicritères se succèdent pour aboutir finalement à une solution donnée (une heuristique gloutonne par exemple), il est possible de générer ainsi toutes les solutions Pareto-optimales d'un problème non-convexe quand bien même toutes les décisions prises sont sur l'enveloppe convexe des valeurs objectives admissibles. C'est d'ailleurs le cas des problèmes non-convexes qui se divisent en une succession de sous-problèmes convexes. Nous pouvons en revanche postuler que ces problèmes sont rares.

Enfin, ces méthodes sont greffées à une heuristique d'ordonnancement existante pour la rendre multicritère.

2.3.1 Techniques d'agrégation

Les méthodes présentées ci-dessous ne concernent que la prise de décision multicritère et non pas la construction d'ordonnancements. Nous considérons donc un espace de valeurs objectives dont les points correspondent à des décisions. Comme cela est décrit par la suite, c'est en prenant une succession de décisions qu'un ordonnancement entier peut être généré.

2.3.1.1 Moyenne arithmétique pondérée

Réaliser une moyenne arithmétique pondérée (ou plus généralement, une somme pondérée [60]) d'un ensemble de valeurs objectives est une technique simple, rapide et efficace pour des problèmes convexes.

Le principe consiste à trouver la décision qui minimise la quantité $\theta f_1(d)+(1-\theta)f_2(d)$ où $f_1(d)$ et $f_2(d)$ sont les valeurs objectives d'une décision d pour le premier et second critère, respectivement. En faisant varier le paramètre de compromis θ, la recherche aboutit à des décisions distinctes mais toujours Pareto-optimales.

La figure 2.3.1 offre une interprétation géométrique de ce mécanisme d'agrégation. Pour une valeur donnée θ, on cherche la décision qui, dans l'espace des valeurs objectives, est présente sur la droite de pente $\frac{-\theta}{1-\theta}$ dont l'ordonnée à l'origine λ est minimale. Il s'agit donc de tracer une droite de pente $\frac{-\theta}{1-\theta}$ et de la translater en direction de l'origine. Le dernier point rencontré correspond à la décision recherchée. En l'occurrence, la décision d_2 est retenue.

Toutefois, cette méthode n'est efficace que pour les problèmes convexes. L'exemple représenté sur la figure 2.3.1 ne correspond pas à un problème convexe. Bien que la décision d_1 soit Pareto-optimale, la moyenne pondérée ne permet pas de la retenir quelque soit la valeur du paramètre θ.

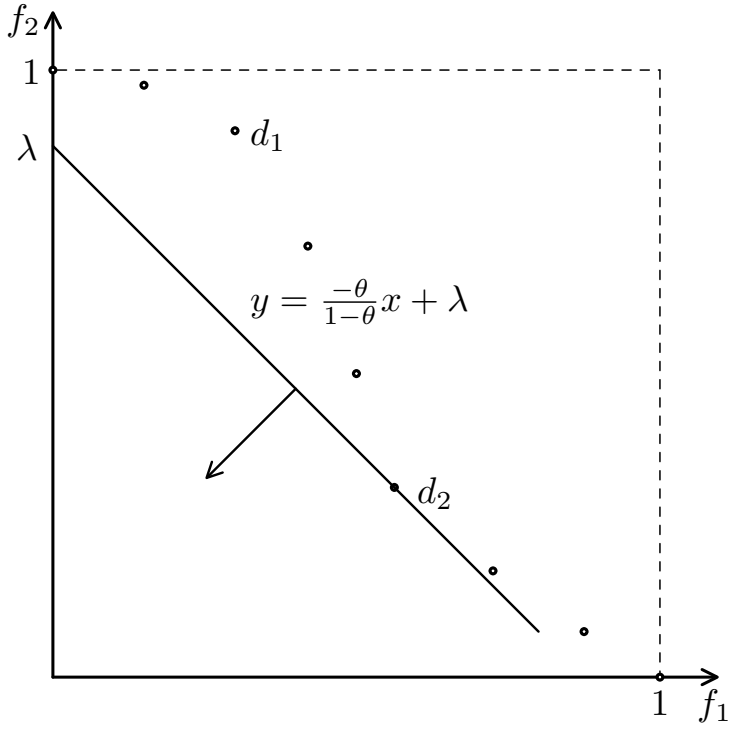


FIGURE 2.3.1 – Interprétation géométrique de la moyenne arithmétique pondérée. La droite de pente $\frac{-\theta}{1-\theta}$ est translatée vers l'origine. La dernière décision rencontré est d_2.

2.3.1.2 Opérateurs d'agrégation pour la décision multicritère

La théorie de la prise de décision multicritère propose des outils pour gérer un ensemble de critères. Nous en abordons brièvement quelques uns dans cette section.

Un ordre lexicographique peut être spécifié sur les critères. Le problème résultant consiste à optimiser en priorité le premier critère, puis à considérer successivement chaque critère dans l'ordre donné en conservant les meilleures valeurs possibles pour les critères déjà optimisés. Les décisions prises dépendent de l'ordre lexicographique choisi. Cependant, aucune décision représentant un compromis entre plusieurs critères ne sera retenue par ce mécanisme.

L'approche ϵ-contrainte est présentée pour la première fois par Haimes et al. [79]. Elle reformule le problème multicritère en un problème monocritère qui se focalise en la minimisation d'une seule des valeurs objectives initiales. Chacune des autres valeurs objectives est contrainte par une valeur maximale qu'elle ne doit pas dépasser. En alternant le critère à optimiser et en variant les bornes supérieures pour les autres critères, il est possible d'atteindre l'ensemble des décisions Pareto-optimales.

Une autre manière d'aborder le problème est de considérer la norme de Tchebycheff pondérée de chaque décision relativement à un point utopique dans l'espace des valeurs objectives qui domine toutes les décisions admissibles. La norme de Tchebycheff non-pondérée correspond exactement à la norme infini. Celle-ci mesure la distance suivant l'une des dimensions entre deux points. La dimension sélectionnée pour calculer la différence entre les valeurs objectives des deux points est celle qui maximise cette distance. La pondération de cette norme permet de modifier l'importance accordée à chaque critère. En variant les poids, chaque décision Pareto-optimale peut être atteinte.

Citons enfin la méthode OWA (Ordered Weighted Averaging) [154] qui généralise le principe de la moyenne pondérée et qui est elle-même généralisée par l'intégrale de Choquet [41]. Le

domaine de l'agrégation des données et les problématiques associées sont présentés par Torra et al. [144].

2.3.1.3 Approche basée sur une direction ⋆

Des approches spécifiant un point et une direction de recherche pour obtenir une décision Pareto-optimale ont été proposées dans le cadre de l'optimisation continue. Citons l'approche de Pascoletti et Serafini [117] qui reformule un problème multicritère en problème monocritère où la solution recherchée est la décision Pareto-optimale qui se situe sur une droite définie par un point et une direction de recherche. Une présentation synthétique de ce type de méthodes est disponible dans [61]. Notons que ces méthodes ne souffrent pas de l'inconvénient de la moyenne pondérée pour les problèmes non-convexes.

Cependant, l'espace des valeurs objectives est discret pour l'optimisation combinatoire. Nous cherchons donc la décision Pareto-optimale la plus proche d'une droite donnée. Dans notre méthode, le compromis à réaliser est représenté par une direction à prendre dans l'espace des valeurs objectives à partir de l'origine.

Algorithmiquement, nous réduisons l'ensemble des décisions possibles à celles qui sont Pareto-optimales. Ensuite, étant donnée une valeur du paramètre de compromis θ, chaque décision est soumise au calcul suivant :

$$\sqrt{x^2+y^2}\sin\left|\theta\frac{\pi}{2}-\arccos\frac{x}{\sqrt{x^2+y^2}}\right|$$

où, par concision, $x = f_1(d)$ et $f_2(d)$. La décision qui minimise cette expression est celle qui est la plus proche de la droite affine faisant un angle de $\theta\frac{\pi}{2}$ avec l'axe des abscisses.

La distance entre une décision et une droite s'obtient par de simples développements trigonométriques dont nous détaillons les étapes principales dans l'exemple illustré sur la figure 2.3.2. Le terme $\arccos\frac{x}{\sqrt{x^2+y^2}}$ correspond à l'angle formé par l'axe des abscisses et la droite qui passe par la décision d_1 et l'origine. Considérons le triangle rectangle délimité par la décision d_1, l'origine et la projection de la décision d_1 sur la droite définie par le paramètre de compromis (i.e., la direction choisie). La longueur de son hypoténuse est $\sqrt{x^2+y^2}$. Dans cet exemple, la décision d_1 est retenue bien qu'elle ne soit pas sur le bord de l'enveloppe convexe.

2.3.2 Normalisation et changement de repère

Supposons que nous souhaitons obtenir une décision avec un compromis équitable entre les deux critères. Dans ce cas, la valeur du paramètre de compromis à choisir est $\theta = 0,5$. Dans l'exemple pathologique de la figure 2.3.3, la décision d_1 est retenue car l'amplitude des valeurs produites par la fonction f_1 est importante relativement à f_2. La décision d_1 est d'autre part la meilleure pour la fonction objective f_2. Ce n'est manifestement pas le résultat attendu. Intuitivement, nous souhaitons obtenir une décision qui soit située au milieu de l'espace des valeurs objectives pour $\theta = 0,5$.

Nous procédons donc par une normalisation des valeurs objectives. Pour chaque critère, les valeurs objectives sont divisées par la valeur maximale atteinte. Ainsi, l'espace des valeurs objectives se cantonne au maximum à $[0;1]^2$ dans le cas bicritère. Cette normalisation permet par ailleurs de manipuler des valeurs comparables, c'est-à-dire des valeurs sans dimension (au sens physique du terme).

Un autre cas pathologique, bien que normalisé, est représenté sur la figure 2.3.4. Dans cette exemple, la moitié des valeurs de θ (celles dans l'intervalle$[0;0,25] \cup [0,75;1]$) conduit à seule-

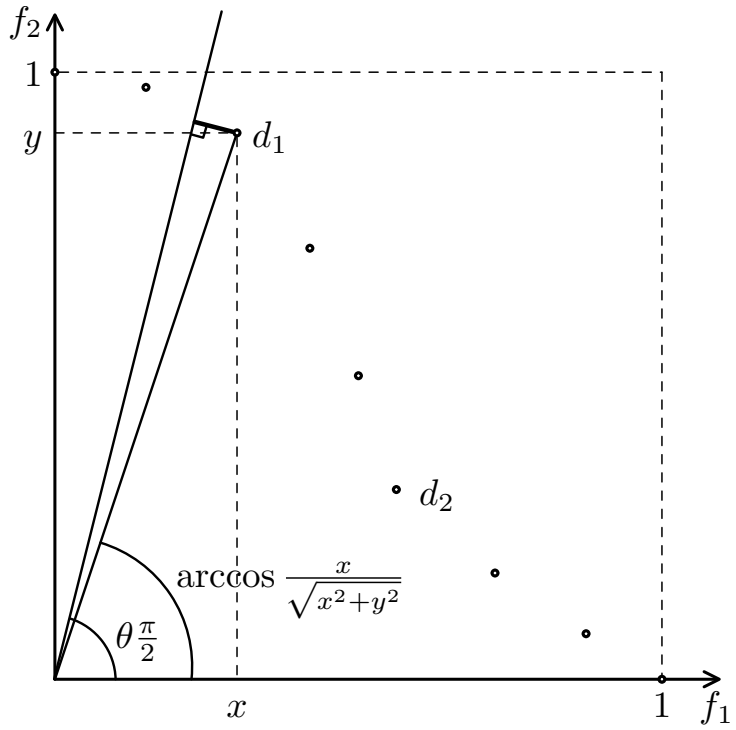


FIGURE 2.3.2 – Approche basée sur une direction à partir de l'origine. La décision la plus proche de la droite définie par la direction est sélectionnée, à savoir d_1.

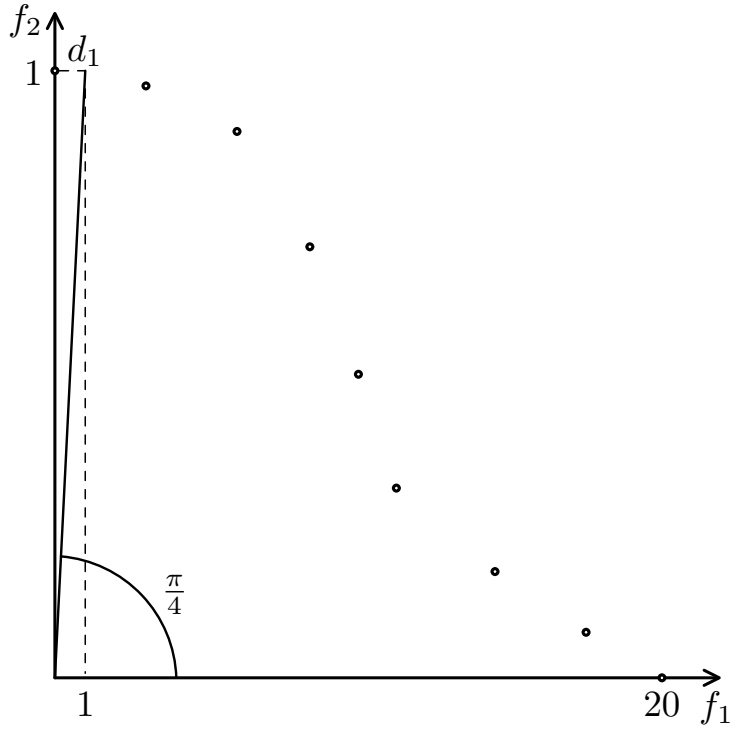


FIGURE 2.3.3 – Espace de valeurs objectives non-normalisé. La fonction f_1 fournit en générale des valeurs plus grandes que f_2.

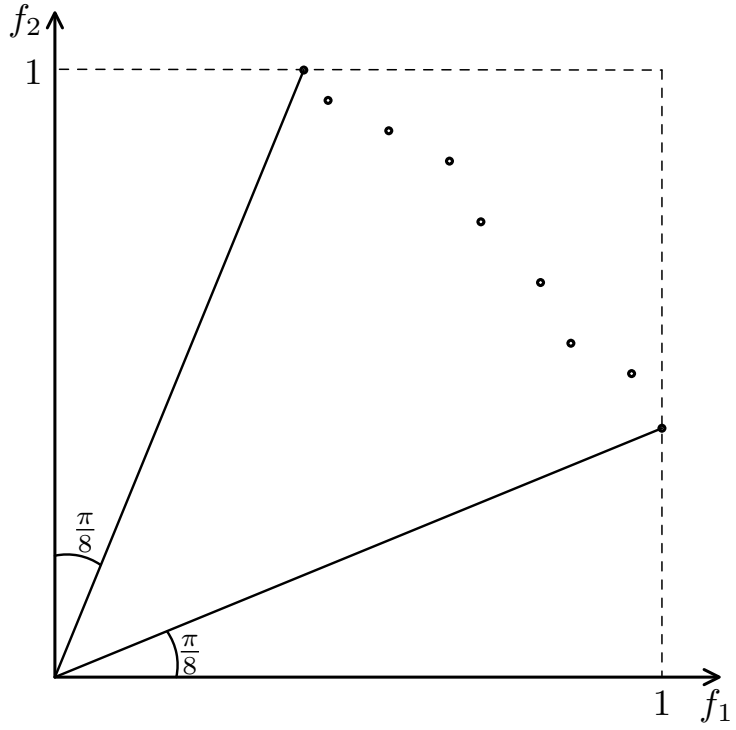


FIGURE 2.3.4 – Exemple de concentration des décisions dans l'espace des valeurs objectives

ment deux décisions. Or, si nous varions le paramètre θ, c'est dans l'idée d'obtenir des décisions présentant des compromis différents.

Notre solution consiste à changer l'origine de l'espace des valeurs objectives de telle sorte que les points de coordonnées $(0;1)$ et $(1;0)$ correspondent aux décisions les plus favorables pour le second critère et le premier critère, respectivement. Finalement, l'espace des valeurs objectives est $[0;1]^2$.

Bien que ces méthodes ne préviennent pas toutes les situations pathologiques, elles rendent deux fonctions objectives comparables grâce à des transformations simples et génériques.

2.3.3 Recherche dichotomique $\star$

Notre contexte d'étude est celui où une succession de décisions multicritères nécessitent d'être prises pour aboutir à un ordonnancement. Nous postulons que le compromis entre les valeurs objectives d'une solution donnée dépend du paramètre de compromis. Nous pouvons donc générer un ensemble de solutions en incrémentant itérativement le paramètre θ d'une faible valeur.

Une faible variation de θ ne conduit pas systématiquement à un nouvel ordonnancement. Pour éviter la redondance des solutions, nous proposons une méthode de recherche dichotomique. Nous faisons l'hypothèse que si la même solution est obtenue en utilisant deux valeurs θ et θ' distinctes, alors toutes les valeurs de l'intervalle $[\theta, \theta']$ conduiront à la même solution. Cette hypothèse justifie l'algorithme 2.1 qui génère efficacement un ensemble de solutions avec des compromis différents. La génération des solutions commence avec les valeurs extrêmes 0 et 1. Chaque étape consiste à choisir une valeur intermédiaire entre une paire de valeurs (en l'occurrence, leur moyenne) et à comparer la solution générée avec celles obtenues avec la paire de valeurs. Si une comparaison révèle une égalité, alors l'intervalle concerné pour le paramètre de compromis n'est pas exploré. Sinon, deux nouvelles paires de valeurs définissent deux nouveaux intervalles à explorer. Pour mettre fin à la recherche, une taille minimale ϵ pour les intervalles est spécifiée.

Algorithme 2.1 Recherche dichotomique

1: Générer les ordonnancements o_0 et o_1 avec les valeurs de compromis 0 et 1
2: Empiler l'intervalle $[0; 1]$
3: **tant que** il reste au moins un intervalle dans la pile **faire**
4: Dépiler un intervalle $[\theta, \theta']$
5: Générer un ordonnancement $o_{\frac{\theta+\theta'}{2}}$ avec la valeur de compromis $\frac{\theta+\theta'}{2}$
6: **si** $\frac{\theta'-\theta}{2} > \epsilon$ **alors** {Assure l'arrêt de la recherche}
7: **si** $o_\theta \neq o_{\frac{\theta+\theta'}{2}}$ **alors**
8: Empiler l'intervalle $[\theta; \frac{\theta+\theta'}{2}]$
9: **fin si**
10: **si** $o_{\theta'} \neq o_{\frac{\theta+\theta'}{2}}$ **alors**
11: Empiler l'intervalle $[\frac{\theta+\theta'}{2}; \theta']$
12: **fin si**
13: **fin si**
14: **fin tant que**

2.3.4 Application à l'ordonnancement

Nous étendons une heuristique d'ordonnancement, HEFT [143], au cas multicritère en utilisant les méthodes précédemment décrites dans les sections 2.3.1.3, 2.3.2 et 2.3.3.

La section 2.2.2 décrit le modèle d'ordonnancement de façon générale : il s'agit d'ordonnancer une application modélisée par un graphe de tâches sur une plateforme de calcul hétérogène. Le résultat de l'opération donne pour chaque processeur l'ensemble de tâches que ce dernier doit exécuter et dans quel ordre (les tâches étant ordonnancées au plus tôt).

La description de l'heuristique HEFT nécessite l'introduction des notations suivantes. Soit le graphe de tâches $G = (T, E)$. L'ensemble $\text{Succ}(t_i)$ désigne les successeurs de la tâche t_i, c'est-à-dire les tâches dont l'exécution ne peut démarrer tant que la tâche t_i n'a pas commencé. La tâche $t_i \in T$ sur le processeur $p_j \in P$ prend w_i^j unités de temps et la communication $(t_i, t_{i'}) \in E$ entre les tâches t_i et $t_{i'}$ exécutées respectivement sur les processeurs p_j et $p_{j'}$ nécessite $d_{ii'}^{jj'}$ unités de temps.

L'algorithme 2.2 détaille le fonctionnement de l'heuristique. On distingue deux étapes. D'abord, toutes les tâches sont triées en fonction de leurs rangs. Intuitivement, le rang d'une tâche est lié à la durée que prendrait l'ordonnancement pour finir son exécution s'il ne restait que cette tâche et ses descendants à exécuter sur un nombre infini de ressources. Le calcul des rangs utilise des coûts moyens. Cette quantité est donc heuristique et sert à ordonner les tâches en commençant par celles qui impactent potentiellement la durée totale de l'ordonnancement de façon critique. Chaque tâche, parcourue par rang décroissant, est ensuite ordonnancée au plus tôt, c'est-à-dire sur le processeur sur lequel son exécution finit le plus tôt.

Dans le cas bicritère, le calcul des rangs inclut le second critère s'il peut se mesurer pour chaque tâche. Chaque tâche possède alors deux rangs, un pour chaque critère. Les tâches sont ensuite triées en utilisant un mécanisme d'agrégation. Suivant la valeur du paramètre de compromis θ, l'un des critères est favorisé. L'étape d'ordonnancement proprement dite consiste à minimiser la date de fin d'exécution de chaque tâche. Le second critère est mesuré à chaque étape et nous cherchons aussi à minimiser la valeur obtenue. Le compromis entre chaque critère dépend une fois encore du paramètre de compromis. Un exemple d'utilisation où le second critère est considéré est décrit dans la section 3.6.2.2. Il s'agit en l'occurrence de la robustesse des ordonnancements.

Algorithme 2.2 Algorithme HEFT

pour chaque $t_i \in T$ **faire** {Parcourt les tâches dans un ordre topologique inverse}
  $w_i = \frac{1}{m} \sum_{p_j \in P} w_i^j$ {Considère le coût moyen de chaque tâche sur l'ensemble des machines}
  $d_{ii'} = \frac{1}{m^2} \sum_{(p_j, p_{j'}) \in P^2} d_{ii'}^{jj'}$ {Considère le coût moyen de chaque communication entre chaque paire de machines}
  $\text{rang}(t_i) = w_i + \max_{t_{i'} \in \text{Succ}(t_i)}(d_{ii'} + \text{rang}(t_{i'}))$ {Calcule le rang de chaque tâche}
fin pour
Trier les tâches suivant leurs rangs dans un ordre décroissant
tant que il reste au moins une tâche non-ordonnancée **faire**
  Soit t la première tâche non-ordonnancée
  pour chaque $p \in P$ **faire**
    Calculer la date de fin d'exécution de t si elle est ordonnancée au plus tôt sur p
  fin pour
  Assigner t sur la machine qui minimise sa date de fin d'exécution
fin tant que

2.4 Algorithmes évolutionnaires multicritères

Les algorithmes génétiques sont une classe d'heuristiques introduite par Golderg [75] et qui se sont révélés efficaces en pratique. Plusieurs variantes ont été proposées et nous retiendrons les algorithmes évolutionnaires qui ont pour principe de faire évoluer une population de solutions. Cette évolution se décompose en une succession de génération. À chaque génération, de nouvelles solutions sont générées à partir des solutions de la génération précédente. Il existe deux mécanismes permettant de produire une solution : les mutations et les croisements. L'application d'un opérateur de mutation sur une solution engendre une nouvelle solution qui diffère légèrement de la solution initiale. D'autre part, deux solutions peuvent être croisées de manière à aboutir à deux nouvelles solutions. L'hypothèse des algorithmes évolutionnaires est que de meilleures solutions peuvent ainsi être produites. Enfin, les meilleures solutions parmi toutes celles générées à une génération donnée sont sélectionnées. Pour obtenir un algorithme évolutionnaire pour un problème spécifique, il faut définir la façon dont sont représentées les solutions, la fonction objective, l'opérateur de croisement et l'opérateur mutation. Comme les algorithmes évolutionnaires définissent des règles d'optimisation génériques, ils font parties des métaheuristiques. Les algorithmes évolutionnaires constituent une méthode efficace pour les problèmes dont la combinatoire est exponentielle.

Grâce aux indicateurs présentés dans la section 2.2.3, nous pouvons détailler quelques techniques évolutionnaires récentes qui spécifient les méthodes de sélection utilisées à chaque génération. Les quelques algorithmes évolutionnaires multicritères présentés sont implémentés dans deux plateformes que nous évoquons.

Nous illustrerons ensuite les mécanismes de mutation et de croisement pour le problème de l'ordonnancement de graphes de tâches sur une plateforme hétérogène. Enfin, un algorithme évolutionnaire utilisant ces mécanismes est proposé et nous prouvons sa convergence pour le problème d'ordonnancement multicritère décrit dans la section 2.2.2.

Les notations introduites dans cette section sont résumées dans le tableau 2.2.

2.4.1 Méthodes récentes

De nombreux algorithmes évolutionnaires ont été proposés pour l'optimisation multicritère. Ces algorithmes définissent les procédures de sélection intervenant sur la population lorsque sa

Symbole	Définition
$S(t)$	Population archive de l'algorithme évolutionnaire à l'étape t
O^*	Ensemble des solutions Pareto-optimales
$\delta_O(O')$	Nombre de solutions de l'ensemble O' qui ne sont pas présentes dans l'ensemble O
N	Nombre maximal de solutions dans chaque population d'un algorithme évolutionnaire ($N = \|S(t)\|$)
M	Matrice stochastique homogène décrivant les transitions de l'algorithme évolutionnaire
$\text{led}(E)$	Diamètre de l'extension linéaire de l'ordre partiel E obtenu à partir d'un graphe de tâche
p_m	Probabilité de mutation
$\mathcal{H}$	Ensemble des états possibles d'un processus markovien correspondant toutes populations possibles, c'est-à-dire tous les ensembles O tel que $\|O\| = N$
H	Ensemble d'états possibles d'un processus markovien ($H \subseteq \mathcal{H}$)
x	État d'un processus markovien correspondant à une population, c'est-à-dire un ensemble O tel que $\|O\| = N$
$K(x, H)$	Noyau markovien donnant la probabilité que l'état soit dans H à l'étape $t+1$ lorsque son état est x à l'étape t

TABLE 2.2 – Liste des notations utilisées dans la section 2.4

taille s'accroît. En revanche, les opérateurs de mutation et de croisement doivent être spécifiés pour chaque problème.

Deb et al. [50] proposent NSGA-II, un algorithme qui se base sur le tri non-dominé et sur la distance de promiscuité (que nous décrivons plus bas) pour sélectionner un sous-ensemble de solutions parmi un ensemble de solutions donné. La procédure de tri non-dominé consiste à trier les solutions en les séparant en plusieurs fronts de solutions non-dominées. Les solutions non-dominées sont dans le premier front. D'autre part, le $k^{\text{ième}}$ front contient les solutions qui ne sont dominées que par les solutions du $k-1^{\text{ième}}$ front. Lors d'une sélection, les solutions qui sont dans les premiers fronts sont d'abord conservées. Dans le cas où les solutions d'un même front ne peuvent pas toutes être sélectionnées, la distance de promiscuité est utilisée pour favoriser les solutions dont les valeurs objectives se trouvent dans une partie de l'espace relativement peu explorée. Cette distance se calcule pour chaque solution en considérant ses deux solutions voisines les plus proches dans l'espace des valeurs objectives. La valeur de la distance est le périmètre du pavé formé par ses deux solutions voisines. Plus cette valeur est faible, plus les voisins en question sont rapprochés, auquel cas la solution considérée n'apporte sans doute pas beaucoup de diversité à la population. Un exemple du calcul de la distance de promiscuité est donné sur la figure 2.4.1.

Zitzler et al. [162] décrivent SPEA2 qui utilise une méthode de sélection similaire à celle de NSGA-II. Cette fois, chaque solution est sélectionnée suivant le nombre de solutions qu'elle domine. En cas d'égalité entre deux solutions, la densité de chacune des deux solutions est estimée par un mécanisme basé sur le voisinage dans l'espace des valeurs objectives. La solution présente dans la zone la moins dense est sélectionnée.

L'algorithme PESA-II [45] spécifie une méthode de sélection des solutions. L'espace des va-

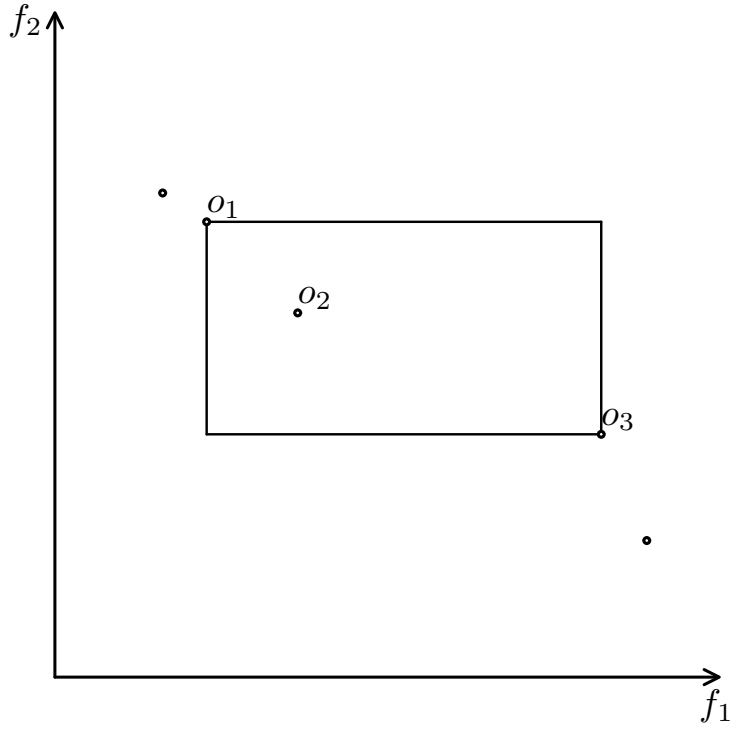


FIGURE 2.4.1 – La distance de promiscuité de l'ordonnancement o_2 est donnée par le périmètre du rectangle formé par les ordonnancements o_1 et o_3 qui sont les plus proches (dans chacune des directions) de o_2 dans l'espace des valeurs objectives

leurs objectives est divisé en plusieurs zones. Ensuite, plusieurs de ces zones, parmi celles non-vides, sont choisies. Dans chacune d'elles, une solution arbitraire est conservée. Le but de cette méthode est de favoriser une répartition uniforme des solutions dans l'espace des valeurs objectives.

L'algorithme IBEA est présenté dans [161]. Les auteurs constatent que la majeure partie des méthodes de recherche multicritère se focalisent sur deux aspects :

– la distance des solutions générées au front de Pareto, et
– l'uniformité de la répartition des solutions dans l'espace des valeurs objectives.

Cependant, il n'existe pas de justification théorique à cette démarche. L'algorithme IBEA permet d'incorporer des préférences définies par le preneur de décision sous la forme d'indicateurs, de fonctions d'utilité, de logique floue ou encore grâce à la programmation par contrat (voir [43] pour un inventaire des méthodes existantes pour spécifier des préférences). Les auteurs illustrent leur méthode en utilisant deux indicateurs : l'ϵ-indicateur binaire et l'hypervolume.

Deb et al. [51] introduisent OMNI comme approche s'appliquant à la fois pour les problèmes monocritères et ceux multicritères. Leur solution améliore NSGA-II sur différent points. Par exemple, le tri des solutions par front utilise la notion d'ϵ-domination (basée sur un principe similaire à l'ϵ-indicateur). D'autre part, des solutions distinctes partageant les mêmes valeurs objectives sont toutes conservées. Bien que les modifications proposées améliorent les performances pour quelques problèmes spécifiques, il n'est pas certain qu'elles aient un impact pour les problèmes d'ordonnancement de par leur nature combinatoire.

Une description des développements récents dans ce domaine est disponible dans [43]. D'autre part, Wagner et al. [148] comparent empiriquement les performances d'algorithmes évolutionnaires (dont NSGA-II, SPEA2 et IBEA) sur des problèmes comportant de trois à six critères.

2.4.2 Implémentations

La plateforme ParadisEO [37] permet d'utiliser de façon simplifiée à la fois des métaheuristiques basées sur l'optimisation d'une solution et des métaheuristiques travaillant sur une population de solutions. Le cas monocritère et le cas multicritère sont traités. Il est aussi possible de concevoir des méthodes parallèles et distribuées. Parmi les métaheuristiques que nous avons présentées, NSGA-II, SPEA2 et IBEA sont implémentées.

La plateforme PISA [29] se focalise sur les méthodes de recherche multicritère et les problèmes associés. La plateforme facilite la validation d'une nouvelle méthode sur l'ensemble des problèmes fournis. Inversement, le comportement des métaheuristiques implémentées pour un nouveau problème s'observe aisément. Notons l'implémentation des mêmes méthodes principales : NSGA-II, SPEA2 et IBEA. Du coté des problèmes d'optimisation, notons la disponibilité d'un problème de sac-à-dos et d'un problème d'ordonnancement.

2.4.3 Mutation et croisement pour l'ordonnancement

Pour tout problème spécifique, comme celui de l'ordonnancement, il est nécessaire de définir la représentation des solutions au sein de l'algorithme évolutionnaire, la fonction d'évaluation et les opérateurs de mutation et de croisement.

Nous rappelons que le problème considéré consiste à ordonnancer un graphe de tâche $G = (T, E)$ sur une plateforme hétérogène. Précisément, nous souhaitons déterminer sur quel processeur chaque tâche doit être exécutée et quel est l'ordre d'exécution des tâches sur chaque processeur. Dans ce problème, les tâches sont exécutés au plus tôt tout en respectant les contraintes de dépendance et l'ordre d'exécution des tâches. Nous notons $n = |T|$ le nombre de tâches.

De nombreuses méthodes ont été proposées pour l'ordonnancement sur un système parallèle dans le cas monocritère [86, 166, 151]. Nous retiendrons l'approche de Wang et al. [149] qui est adaptée à l'ordonnancement de tâches dépendantes sur une plateforme hétérogène. Nous détaillons désormais leur approche.

Un ordonnancement est représenté par deux tableaux de n valeurs qui sont appelés *chaîne de correspondance* et *chaîne d'ordonnancement*. La $i^{\text{ième}}$ tâche est assignée au processeur désigné par la $i^{\text{ième}}$ valeur de la chaîne de correspondance. Cette chaîne définit donc sur quel processeur doit être exécutée chaque tâche. La chaîne d'ordonnancement est un tri topologique des tâches, c'est-à-dire que si la tâche t est positionnée avant la tâche t' dans cette chaîne, alors la tâche t' n'est pas un ancêtre de la tâche t. Cette chaîne est utilisée pour déterminer l'ordre d'exécution des tâches sur chaque processeur. En considérant que les tâches t et t' mentionnées plus haut sont assignées au même processeur, alors la tâche t est exécutée avant t'. Ces deux chaînes déterminent entièrement une solution pour notre problème d'ordonnancement.

L'opérateur de mutation génère une solution aléatoire à partir d'une solution donnée. L'opération modifie à la fois la chaîne de correspondance et la chaîne d'ordonnancement. Une tâche est d'abord sélectionnée aléatoirement et son processeur correspondant dans la chaîne de correspondance est remplacé par un processeur choisi aléatoirement. La chaîne d'ordonnancement est altérée relativement à cette même tâche. Cependant, un nouveau tri topologique doit être obtenu lorsqu'une mutation est réalisée. Cela signifie que cette tâche ne peut être repositionnée dans le tri topologique que si son nouvel emplacement ne contredit pas la définition d'un tri topologique.

L'opérateur de croisement génère deux solutions à partir d'une paire de solutions. Pour cela les deux chaînes d'une des solutions de départ sont séparées en deux parties. Une nouvelle chaîne de correspondance est obtenue en prenant la partie gauche de la chaîne de correspondance d'une des solutions de départ et en la complétant avec la partie droite de l'autre solution de départ. Une nouvelle chaîne d'ordonnancement est obtenue grâce à la partie gauche d'une des chaînes

d'ordonnancement de départ et en complétant le tri topologique en respectant l'ordre défini par l'autre chaîne d'ordonnancement de départ. En réalisant ces opérations réciproquement, nous obtenons deux chaînes de correspondance et deux chaînes d'ordonnancement qui sont alors assemblées en deux solutions finales.

Les fonctions objectives dépendent du problème multicritère traité. La méthode décrite dans [149] s'applique à l'ordonnancement multicritère de graphes de tâches quelque soient les critères. Les critères spécifiques que nous avons utilisés sont décrits dans la section 3.6.2.3.

2.4.4 Extension de l'algorithme NSGA-II ⋆

L'algorithme NSGA-II (détaillé dans la section 2.4.1) est une métaheuristique multicritère faisant référence et générant des résultats comparables à IBEA pour les problèmes combinatoires [161]. Nous décrivons dans cette section un algorithme évolutionnaire multicritère qui diffère de NSGA-II par quelques ajustements afin de le rendre convergeant sous certaines conditions détaillées par la suite.

L'algorithme 2.3 décrit la boucle principale de l'algorithme de recherche que nous proposons. Cette boucle est répétée pour chaque étape t. La population parente $P(t)$ est combinée à la population générée $Q(t)$ à la ligne 1 pour obtenir la population $R(t)$. À la ligne 2, cette population est alors partitionnée en ensembles de solutions non-dominées de la même manière qu'avec NSGA-II. Ainsi, chaque solution du front O_{i+1} est dominée par au moins une solution de front O_i. L'ensemble S est mis à jour de la ligne 3 à la ligne 11. Cet ensemble contient les solutions non-dominées rencontrées au cours de la recherche et constitue un ajout vis-à-vis de l'algorithme initial. À la première itération, l'ensemble S est vide. Cette étape assure que toute solution insérée dans S y reste jusqu'à ce qu'elle soit dominée par une nouvelle solution. Il s'agit d'un mécanisme d'élitisme qui n'interagit pas avec la recherche de nouvelles solutions. Les lignes 12 à 16 probabilisent la sélection des solutions de $R(t)$ qui feront partie de la nouvelle population parente $P(t+1)$ et qui seront utilisées pour générer la population $Q(t+1)$. Cette sélection est nécessaire car la taille des populations à chaque étape est fixe. Le paramètre l dénote l'importance accordée aux premiers fronts. Lorsque la valeur de ce paramètre augmente, la probabilité qu'une solution parmi les premiers ensembles soit retenue dans $P(t+1)$ augmente. Inversement, lorsque $l = 0$, chaque front possède la même importance. La distance de promiscuité est utilisée pour comparer deux solutions appartenant au même front O_i. Elle permet de déterminer quelle solution est positionnée dans la zone de l'espace des valeurs objectives la moins dense et donc la solution qu'il faut conserver pour favoriser la diversité. Sur un front donné, la priorité est donnée aux solutions possédant la distance de promiscuité la plus large. Ces solutions sont d'ailleurs sélectionnées avec une probabilité plus importante (ligne 16). Au final, la solution s_j du front O_i est sélectionnée avec la probabilité $p_{ij} > 0$. Comme cette probabilité est non-nulle, chaque solution peut être sélectionnée pour faire partie de la génération suivante. La sélection des solutions se fait suivant les probabilités p_{ij} à la ligne 19. Enfin, la nouvelle population $Q(t+1)$ est générée à la ligne 20. Dans la fonction *générer-population*, la sélection est non-déterministe et la probabilité de croisement est strictement inférieure à un. Ainsi, toute solution peut potentiellement se retrouver inchangée à l'étape suivante.

2.4.5 Conditions de convergence ⋆

Nous prouvons qu'il y a convergence vers un ensemble de solutions Pareto-optimales lorsque l'algorithme décrit dans la section 2.4.4 est utilisé conjointement avec les techniques évoquées dans la section 2.4.3 pour le problème d'ordonnancement.

Algorithme 2.3 Version modifiée de la boucle principale de NSGA-II

1: $R(t) = P(t) \cup Q(t)$ {Combine la population parente et celle générée}
2: $O = \text{tri-non-dominé}(R(t))$ {$O = (O_1, O_2, \ldots)$ sont les fronts non-dominés de $R(t)$}
3: **pour chaque** $o \in O_1$ **faire** {Pour chaque solution non-dominée o}
4: $D_o = \{s \in S(t) : f(o) \prec f(s)\}$ {D_o : ensemble de solutions dans S dominées par o}
5: $N_o = \{s \in S(t) : f(s) \prec f(o)\}$ {N_o : ensemble de solutions de S dominant o}
6: $S(t) = S(t) \setminus D_o$ {Enlève les solutions dominées de S}
7: **si** $N_o = \emptyset$ **et** $|S(t)| < N$ **alors**
8: $S(t) = S(t) \cup \{o\}$ {Ajoute o à S en garantissant que $|S| \leq N$)}
9: **fin si**
10: **fin pour**
11: $S(t+1) = S(t)$
12: **pour chaque** $O_i \in O$ **faire**
13: $p_i = \frac{1}{i^l \times \sum_{k=1}^{|O|} \frac{1}{k^l}}$ {Calcule la probabilité de sélection du front O_i}
14: distance-promiscuité(O_i) {Calcule les distances de promiscuité c_{ij} de chaque solution s_j du front O_i}
15: **pour chaque** $o_j \in O_i$ **faire** {Pour chaque solution o_j du front O_i triée par c_{ij} décroissant}
16: $p_{ij} = p_i \times \frac{1}{j^l \times \sum_{k=1}^{|O_i|} \frac{1}{k^l}}$ {Calcule la probabilité de sélection de la solution o_j du front O_i}
17: **fin pour**
18: **fin pour**
19: $P(t+1) = \text{séléction}(N, R(t), \{p_{ij}\})$ {Sélectionne N éléments distincts de $R(t)$ tirés aléatoirement selon les probabilités p_{ij}}
20: $Q(t+1) = \text{générer-population}(P(t+1))$ {Utilise les opérateurs de mutation et de croisement pour créer une nouvelle population}
21: $t = t + 1$ {Incrémente le compteur de génération}

Rudolph [126, définition 3] définit la condition pour qu'un algorithme évolutionnaire multicritère converge. Soit O^* l'ensemble des solutions Pareto-optimales. Nous notons $S(t)$ la population générée par l'algorithme évolutionnaire à l'étape t. Enfin, $\delta_O(O')$ est le nombre de solutions de l'ensemble O' qui ne sont pas présentes dans l'ensemble O. Un algorithme évolutionnaire converge presque sûrement vers un ensemble de solutions Pareto-optimales si $\delta_{O^*}(S(t)) \to 0$ lorsque $t \to \infty$. Il est évident que la population $S(t)$ n'est jamais vide sauf s'il n'existe aucune solution admissible. Nous en déduisons que lorsque $t \to \infty$, $|S(t)| = \min(|O^*|, N)$.

Nous donnons des conditions suffisantes pour assurer la convergence de notre algorithme en considérant la définition précédente de la convergence. Pour cela, nous étendons les résultats existants pour prendre en compte les opérateurs de mutations locales.

Pour prouver la convergence, il est nécessaire de prouver la proposition suivante qui énonce que toutes les solutions de l'ensemble S tendent à être Pareto-optimales tandis que t augmente.

Proposition 2.1. *Soit $P(t)$ la population parente de taille N à l'étape t et $\kappa > 0$ un entier. Soit M la matrice stochastique homogène*[1] *décrivant les transitions entre l'étape t et l'étape $t + \kappa - 1$ (de $P(t)$ à $Q(t + \kappa - 1)$). Soit S l'ensemble des solutions résultant de l'algorithme 2.3. Si la matrice M est positive, alors $\delta_{O^*}(S(t)) \to 0$ et $|S(t)| \to \min\{N, |O^*|\}$ presque sûrement lorsque $t \to \infty$.*

Démonstration. La preuve s'inspire de [126, Proposition 4]. Si à l'étape t, $S(t)$ contient une solution non-optimale s, alors l'algorithme évolutionnaire génère presque sûrement et en temps fini une solution o qui domine s. En effet, la matrice M est positive et cela implique que toute solution admissible peut être obtenue à partir de n'importe quelle autre solution. Les lignes 3 à 11 garantissent que l'algorithme évolutionnaire remplace s par o dans S. De plus, ces mêmes lignes assurent qu'une solution Pareto-optimale présente dans S y reste indéfiniment. □

Cela signifie que l'algorithme 2.3 converge si la matrice de transition M est positive (i.e., tous ses éléments sont strictement positifs). Plus précisément, il converge si l'ensemble S tend à n'être composé que de solutions Pareto-optimales lorsque $t \to \infty$.

Nous montrons qu'il existe un entier positif κ tel que la matrice de transition entre l'étape t et l'étape $t + \kappa - 1$ est positive. Nous utilisons l'opérateur de mutation décrit dans la section 2.4.3. Cette mutation est locale, c'est-à-dire qu'elle ne permet que de générer un sous-ensemble des solutions admissibles à partir d'une solution donnée. Cet ensemble de solutions est d'ailleurs appelé *voisinage*. Le lemme 2.2 affirme cependant qu'appliquer κ fois cet opérateur aboutit à un opérateur de mutation globale.

Proposition 2.2. *L'opérateur de mutation locale proposé par Wang et al. [149] est équivalent à un opérateur de mutation globale s'il est appliqué $\kappa = \max(n, led(E)) \leq \max(n, \frac{n(n-1)}{2})$ fois successivement, avec n le nombre de tâches et $led(E)$ le diamètre de l'extension linéaire [66] du graphe de tâches.*

Démonstration. Chaque ordonnancement est représenté par l'intermédiaire de deux chaînes : la chaîne de correspondance qui spécifie le processeur auquel est assignée chaque tâche ; et la chaîne d'ordonnancement qui précise l'ordre d'exécution des tâches.

Le nombre de mutations successives requises pour passer d'une chaîne de correspondance quelconque à une autre s'obtient en considérant chaque élément de la chaîne. Chaque fois qu'une tâche est sélectionnée, elle est assignée à un processeur choisi aléatoirement sans contrainte additionnelle. La probabilité d'obtenir une chaîne de correspondance donnée à partir d'une chaîne

1. La dimension de M est C_N^η où η est le nombre de solutions admissibles.

quelconque après n mutations est bornée par $\delta_{m_1} = \left(\frac{p_m}{nm}\right)^n n! > 0$ où n est le nombre de tâches, m le nombre de processeurs et $p_m > 0$ la probabilité qu'une mutation se produise.

La chaîne d'ordonnancement est une extension linéaire d'un ordre partiel E obtenu à partir du graphe de tâches G (en l'occurrence l'ordre partiel correspond effectivement aux contraintes de dépendance définies dans l'ensemble E). Toute mutation doit modifier cette chaîne en respectant l'ordre partiel (i.e., les contraintes de dépendance). Le nombre maximal de permutations requises pour obtenir une extension linéaire donnée à partir d'une autre est appelé le diamètre de l'extension linéaire $\mathrm{led}(E)$. Felsner et al. [66] ont montré qu'une borne supérieure de ce diamètre est donnée par $\mathrm{Inc}(E)$, le nombre de paires d'éléments incomparable dans E. Ainsi, $\mathrm{led}(E) \leq \frac{n(n-1)}{2}$. La probabilité d'obtenir une chaîne d'ordonnancement donnée à partir d'une chaîne quelconque après $\mathrm{led}(E)$ mutations est bornée par $\delta_{m_2} = \left(\frac{p_m}{n^2}\right)^{\mathrm{led}(E)} > 0$.

Après $\kappa = \max(n, \mathrm{led}(E)) \leq \max(n, \frac{n(n-1)}{2})$ mutations, la probabilité d'obtenir un ordonnancement donné à partir d'un ordonnancement quelconque est bornée par $\delta_m = \delta_{m_1}\delta_{m_2} > 0$. Nous obtenons donc un opérateur de mutation globale en appliquant successivement des mutations locales. □

Le lemme précédant suppose que les mutations sont appliquées successivement. Cependant, une mutation se produit entre les opérations de croisement et de sélection. Le lemme 2.3 montre que κ étapes successives (croisement, mutation et sélection) n'impactent pas la convergence de l'algorithme évolutionnaire si l'on suppose que toute solution d'une population parente peut survivre aux croisements et aux sélections pour se retrouver dans la population générée.

Nous introduisons quelques notations et définitions avant d'énoncer le lemme 2.3. Soit $\mathcal{H}$ l'ensemble des états possibles d'un processus stochastique. L'état du processus à l'étape t est noté X_t. Un noyau markovien K se définit par $K(x, H) = \Pr[X_{t+1} \in H \mid X_t = x]$, c'est-à-dire la probabilité que l'état du processus stochastique soit dans $H \subseteq \mathcal{H}$ à l'étape $t+1$ lorsque son état est x à l'étape t. Chaque état x correspond à une configuration de la population $P(t)$ donnée. Le produit des noyaux (K_c, K_m, K_s) est le noyau markovien représentant une étape de l'algorithme évolutionnaire. L'opérateur de croisement correspond à K_c, celui de mutation est K_m et celui de sélection K_s. Enfin, le noyau K itéré κ fois se note $K^{(\kappa)}$.

Lemme 2.3. *Soit $K_c(x, \{x\}) \geq \delta_c$ et $K_s(x, \{x\}) \geq \delta_s$ pour tout $x \in \mathcal{H}$. Alors, pour tout $H \subseteq \mathcal{H}$*

$$\left(K_c\, K_m\, K_s\right)^{(\kappa)}(x, H) \geq \left(\delta_c\delta_s\right)^{\kappa} K_m^{(\kappa)}(x, H)$$

Démonstration. (Par induction) Nous notons $1_H(x)$ la fonction indicateur pour un ensemble H ($1_H(x) = 1$ si $x \in H$, 0 sinon). D'après les hypothèses du lemmes, $K_c(x, H) \geq \delta_c 1_H(x)$ et $K_s(x, H) \geq \delta_s 1_H(x)$ pour tout $x \in \mathcal{H}$ et pour tout $H \subseteq \mathcal{H}$. Nous initialisons l'induction pour $\kappa = 1$ en développant $\left(K_c\, K_m\, K_s\right)(x, H)$

$$\begin{aligned}
\left(K_c\, K_m\, K_s\right)(x, H) &= \int_{\mathcal{H}} \left(\int_{\mathcal{H}} K_c(x, dz) K_m(z, dy) \right) K_s(y, H) \\
&\geq \int_{\mathcal{H}} \left(\int_{\mathcal{H}} \delta_c 1_{dz}(x) K_m(z, dy) \right) \delta_s 1_H(y) \\
&\geq \delta_c\delta_s \int_H \left(\int_{\mathcal{H}} 1_{dz}(x) K_m(z, dy) \right) \\
&\geq \delta_c\delta_s \int_H K_m(x, dy) \\
&\geq \delta_c\delta_s K_m(x, H) \qquad (2.4.1)
\end{aligned}$$

Nous montrons désormais que l'hypothèse est correcte pour $\kappa > 1$. L'équation 2.4.1 induit que

$$\begin{aligned}(K_c\, K_m\, K_s)^{(\kappa+1)}(x, H) &= \int_{\mathcal{H}} (K_c\, K_m\, K_s)^{(\kappa)}(y, H)\,(K_c\, K_m\, K_s)(x, dy) \\ &\geq \delta_c\delta_s \int_{\mathcal{H}} (K_c\, K_m\, K_s)^{(\kappa)}(y, H) K_m(x, dy)\end{aligned}$$

Par hypothèse d'induction,

$$\int_{\mathcal{H}} (K_c\, K_m\, K_s)^{(\kappa)}(y, H) K_m(x, dy) \geq \int_{\mathcal{H}} (\delta_c\delta_s)^{\kappa} K_m^{(\kappa)}(y, H) K_m(x, dy)$$

et par définition, $K_m^{(\kappa+1)}(x, H) = \int_{\mathcal{H}} K_m^{(\kappa)}(y, H) K_m(x, dy)$. Ainsi, l'hypothèse est vraie pour $\kappa \geq 1$. □

Nous énonçons le résultat principal de cette section.

Théorème 2.4. *L'algorithme 2.3 converge presque sûrement vers un ensemble de solutions Pareto-optimales en temps fini.*

Démonstration. Soit $M = (m_{ij})$ la matrice des probabilités de transition entre une population (correspondant à un état i) à l'étape t et une autre (correspondant à un état j) à l'étape $t+\kappa-1$ avec $\kappa > 0$. D'après la proposition 2.1, il faut prouver que M est positive. Le lemme 2.2 montre que si $\kappa \geq \max(n, \mathrm{led}(E))$, alors $K_m^{(\kappa)}(i, \{j\}) > 0$ pour tous les états i et j. Soit m_{ij} la probabilité de transition entre l'état i et l'état j quand le processus stochastique est appliqué κ fois. Par définition, $m_{ij} = (K_c\, K_m\, K_s)^{(\kappa)}(i, \{j\})$. Le lemme 2.3 montre que $m_{ij} \geq (\delta_c\delta_s)^{\kappa} K_m^{(\kappa)}(i, \{j\}) > 0$ si δ_c et δ_s sont strictement positifs. Comme l'algorithme spécifie que la probabilité de croisement est strictement inférieure à 1 et que la phase de sélection est stochastique, alors $\delta_c > 0$ et $\delta_s > 0$. □

Pour résumer, la version modifiée de NSGA-II converge vers un ensemble de solutions optimales, mêmes avec des mutations locales.

2.5 Conclusion

Nous présentons dans ce chapitre un ensemble de concepts liés à l'ordonnancement bicritère en nous abstrayant des critères considérés. Deux techniques d'optimisation génériques sont proposées à ce titre et sont évaluées dans le chapitre 3.

La première est une heuristique gloutonne qui étend une méthode de la littérature en remplaçant chaque décision monocritère par une décision bicritère. La technique finale fait appel à cette heuristique plusieurs fois en utilisant un principe de dichotomie. Quelques mécanismes pratiques sont utilisés pour éviter de gérer des espaces de valeurs objectives indésirables. En revanche, toutes les situations pathologiques ne sont pas prises en compte. Nous pensons qu'il serait intéressant de formaliser les décisions multicritères successives que l'on rencontre dans les stratégies de construction gloutonnes.

Dans la dernière section, une métaheuristique est introduite et sa convergence est prouvée. Bien que cette propriété soit intéressante, elle ne dénote pas la qualité de l'approche en terme d'efficacité. Estimer la rapidité de convergence aurait un impact plus fort en pratique. L'étude des marches aléatoires (*random walks*) dans les graphes a permis d'obtenir certains résultats à ce

regard (un algorithme évolutionnaire est considéré comme une marche aléatoire dans [83]). Plus précisément, nous pourrions étendre nos résultats en nous intéressant au temps moyen nécessaire pour couvrir l'ensemble des états possibles d'un processus stochastique (voir [111, 108] pour un aperçu du domaine et des concepts de temps de couverture, de frappe et de mixage).

Deuxième partie

Algorithmes

Chapitre 3

Ordonnancement statique de tâches à durée aléatoire

Sommaire

3.1 Introduction

L'ordonnancement de tâches à durée aléatoire s'inscrit dans le contexte des environnements de calcul distribué que nous abordons dans un premier temps. Nous caractérisons l'incertitude considérée et donnons enfin un aperçu du plan de ce chapitre.

3.1.1 Description du contexte

Les environnements de calcul distribué évoluent vers des systèmes toujours plus complexes. Les environnements actuels sont dynamiques, larges et reposent sur des architectures matérielles sophistiquées. Cette tendance multiplie les sources possibles d'incertitude. Cela rend imprévisibles la charge de calcul et la disponibilité des ressources calculatoires par exemple.

Cette complexité grandissante complique l'estimation des durées d'exécution à cause de la difficulté à prédire le comportement des ressources calculatoires ou des applications. La sensibilité des applications aux données en entrée amplifie le problème. En conséquence, l'estimation des durées d'exécution nécessite de simplifier les modèles de coûts ce qui a pour effet d'aboutir sur des prédictions s'éloignant de la réalité dans une proportion indéterminée.

Ceci nous amène naturellement à nous intéresser à la robustesse du système, c'est-à-dire sa capacité à maintenir ses performances en présence d'aléas. En l'occurrence, ce concept est lié à la stabilité du système lorsque les durées d'exécution ne sont pas déterministes. La robustesse n'est pas une mesure mais un critère qui s'applique sur une mesure de performance comme la durée totale d'un ordonnancement, la répartition de la charge d'une application, ou encore le temps d'attente moyen dans une file d'attente.

Notre étude se focalise sur l'ordonnancement statique de tâches dont les durées d'exécution sont incertaines. Cela signifie que la ressource sur laquelle sera exécutée chaque tâche est fixée avant que l'exécution ne commence. En présence d'incertitude, une approche dynamique possède l'avantage de prendre des décisions adaptées aux durées effectives des exécutions. Dans certains cas, cette approche est complémentaire. Elle peut par exemple enrichir un ordonnancement statique déjà calculé en l'adaptant aux circonstances. Cependant, l'approche dynamique se limite aux techniques gloutonnes et interdit les stratégies d'optimisation globale qu'ils seraient trop coûteux d'utiliser dynamiquement. Nous limitons notre étude à des ordonnancements statiques que viendrait enrichir complémentairement une approche dynamique.

3.1.2 Typologie

Étant donné un graphe de tâches et une plateforme hétérogène, nous voulons assigner de façon ordonnée chaque tâche à une ressource calculatoire de manière à minimiser la durée totale de l'ordonnancement, c'est-à-dire la durée que prend l'ensemble des tâches à être exécutées. Ce problème se note $QR|prec|C_{\max}$ suivant la notation $\alpha|\beta|\gamma$ [76]. Ce problème qui se focalise sur l'efficacité (la durée totale) a déjà été étudié [62, 107] et de nombreuses heuristiques comme HEFT ou CPOP [143] ont été proposées.

L'incertitude liée aux durées des calculs est modélisée en remplaçant chaque durée par une variable aléatoire. Cette modélisation est notamment proposée dans [119]. Dans notre étude, cette incertitude est de nature méthodologique car elle est due aux méthodes utilisées pour estimer les durées d'exécution. En effet, la complexité des architectures et des applications motive l'utilisation de méthodes heuristiques plutôt que de méthodes exactes. D'autre part, la source de l'incertitude est matérielle ou/et logicielle. Les variables aléatoires associées aux durées modélisent aussi une incertitude de type aléatoire dans le cas d'applications probabilistes (comme un algorithme de tri probabiliste) dont le déroulement de l'exécution dépend d'un tirage aléatoire.

Symbole	Définition
$T = \{t_i : i \in [1..n]\}$	Ensemble des tâches
n	Nombre de tâches ($n = \|T\|$)
$G = (T, E)$	Graphe orienté acyclique contenant les tâches et les contraintes de dépendance
$\text{Pred}(t_i)$	Ensemble des prédécesseurs de la tâche t_i ($\text{Pred}(t_i) \subset T$)
$P = \{p_j : j \in [1..m]\}$	Ensemble des processeurs
m	Nombre de processeurs ($m = \|P\|$)
w_i^j	Durée d'exécution de la tâche t_i sur le processeur p_j
$d_{ii'}^{jj'}$	Durée de la communication $(t_i, t_{i'}) \in E$ entre les tâches t_i et $t_{i'}$ exécutées respectivement sur les processeurs p_j et $p_{j'}$
S_i	Date du début d'exécution de la tâche t_i
C_i	Date de fin d'exécution de la tâche t_i
$C_{\max}$	Durée totale de l'ordonnancement (maximum des dates de fin C_i)
$\mathcal{O}$	Ensemble des ordonnancements valides
O	Ensemble d'ordonnancements non-dominés
$I_\epsilon : \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O}) \mapsto \mathbb{R}$	ϵ-indicateur qui compare deux ensembles d'ordonnancements non-dominés
$I_C : \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O}) \mapsto \mathbb{R}$	Indicateur de couverture qui compare deux ensembles d'ordonnancements non-dominés
θ	Paramètre de compromis entre l'efficacité et la robustesse

TABLE 3.1 – Notations du chapitre

3.1.3 Description de l'étude

Nous organisons notre étude en deux parties principales : la caractérisation d'une mesure de robustesse pertinente et la recherche d'ordonnancements efficaces et robustes.

Une *mesure* de robustesse est une fonction qui associe à un ordonnancement une valeur numérique représentative de sa robustesse. Nous parlerons donc de la mesure (de robustesse) d'un ordonnancement pour désigner la valeur résultante. Il est important de distinguer le principe de robustesse (qui est un critère) de la valeur produite par une mesure (qui est une valeur objective). Comme il n'existe pas de consensus sur une mesure de robustesse définitive, nous comparons empiriquement dans la section 3.5 les mesures de robustesse décrites dans la section 3.3. La section 3.4 présente le plan d'expérience mis au point pour cette étude empirique.

Le problème d'optimisation est quant à lui difficile. Nous verrons que l'évaluation des mesures d'efficacité et de robustesse de chaque solution est un problème #P'-Complet. L'étape d'ordonnancement est par ailleurs NP-Complet en mono-critère [107]. Nous proposons dans la section 3.6 plusieurs méthodes d'ordonnancement bicritère (efficacité et robustesse) regroupant des heuristiques et une métaheuristique. La section 3.7 montre empiriquement que chacune de ces méthodes possède un compromis unique en terme de temps et de qualité.

3.2 Modèles et définitions

Les notations utilisées dans ce chapitre sont résumées dans le tableau 3.1.

3.2.1 Application et plateforme

Nous considérons une application modélisée par un graphe de tâches $G = (T, E)$. Le graphe G est orienté acyclique. L'ensemble T contient les tâches et E est l'ensemble des contraintes de dépendance entre ces tâches. Chaque arc est une communication entre deux tâches : la communication peut commencer lorsque la tâche source est calculée et doit se terminer avant que la tâche destination ne commence.

Nous cherchons à exécuter l'application sur une plateforme composée d'un ensemble P de ressources calculatoires (ou processeurs). Chaque machine peut communiquer avec toutes les autres pour transmettre des données. Nous supposons qu'il n'y a pas de contention. Plusieurs transferts simultanés n'entrent donc pas en conflit. Les ressources de calcul et de communication sont hétérogènes en terme d'efficacité. En toute généralité, exécuter la tâche $t_i \in T$ sur le processeur $p_j \in P$ prend w_i^j unités de temps. D'autre part, la communication $(t_i, t_{i'}) \in E$ entre les tâches t_i et $t_{i'}$ exécutées respectivement sur les processeurs p_j et $p_{j'}$ nécessite $d_{ii'}^{jj'}$ unités de temps.

Dans le cas d'une plateforme uniforme [76], les machines sont soumises à des vitesses et nous avons $w_i^j = c_i \times \tau_j$ où c_i est le coût de la tâche t_i et τ_j est le temps de cycle du processeur p_j. Par ailleurs, nous pouvons procéder de même pour les communications : $d_{ii'}^{jj'} = l_{jj'} + c_{ii'} \times b_{jj'}$ où $l_{jj'}$ est la latence entre les processeurs p_j et $p_{j'}$, $c_{ii'}$ est la quantité de données à transmettre entre les tâches t_i et $t_{i'}$, et $b_{jj'}$ est le temps nécessaire pour envoyer un élément entre les deux processeurs.

L'étape d'ordonnancement consiste à assigner chaque tâche à un processeur tout en respectant les contraintes de dépendance spécifiées par le graphe de tâches et les contraintes liées aux ressources (un seul calcul peut être réalisé à un instant donné sur un processeur et les exécutions ne s'interrompent pas). Nous définissons dans la section suivante la façon dont les tâches sont exécutées sur chaque processeur.

3.2.2 Modèle d'exécution

Nous modélisons les variations possibles dans les durées d'exécution à l'aide de variables aléatoires. Chaque variable aléatoire donne ainsi la probabilité qu'une exécution ou qu'une communication finisse avant une date donnée. Nous ne contraignons pas le type des variables aléatoires. En revanche, nous supposons que chaque espérance et que chaque variance est définie. Pour caractériser une variable aléatoire, nous nous référons à sa distribution ou à sa densité de probabilité, ou encore à la loi de probabilité qu'elle suit.

Par convention, nous notons les variables aléatoires avec des capitales. Les durées w_i^j et $d_{ii'}^{jj'}$ sont donc issues des variables aléatoires W_i^j et $D_{ii'}^{jj'}$ associées. Dans le cas d'une plateforme uniforme, nous pouvons considérer que les coûts c_i et $c_{ii'}$ sont aléatoires.

Comme toutes les durées sont aléatoires, les dates de début et de fin d'exécution des tâches sont aussi aléatoires. Lors d'une exécution, toutes ces dates prennent des valeurs déterministes. En répétant l'exécution d'un ordonnancement, de nouvelles valeurs peuvent être obtenues. C'est pourquoi nous considérons une classe d'ordonnancements spécifique : seul l'ordre d'exécution des tâches sur chaque processeur est donné. Dans ce cas, chaque tâche commence dans l'ordre spécifié au plus tôt (dès que ses contraintes de dépendance sont satisfaites). De cette manière, aucune date de début ou de fin d'exécution ne requiert d'être définie dans l'ordonnancement. En revanche, nous n'autorisons pas l'insertion de temps libre dans l'ordonnancement. De nombreuses heuristiques produisent de tels ordonnancements (HEFT et CPOP [143] par exemple). Notons également que les heuristiques de listes ne rajoutent pas de temps libre non plus.

3.2.3 Évaluation de la durée totale d'un ordonnancement

Soit $C_{\max}$ la durée totale d'un ordonnancement. Dans le cas où les durées sont aléatoires, $C_{\max}$ est une variable aléatoire. Nous avons donc besoin d'une méthode pour caractériser la distribution de $C_{\max}$. Cette caractérisation est un problème #P'-Complet (cela découle du résultat de complexité énoncé dans la section 1.4 du chapitre 1). Les méthodes exactes connues sont donc coûteuses. Nous utilisons les heuristiques proposées dans la section 1.5 en suivant la démarche décrite dans la section 1.2.4 pour réduire notre problème à l'évaluation d'un graphe stochastique.

Nous détaillons désormais quelques notations liées à la durée totale d'un ordonnancement. Nous dénotons la densité de probabilité de $C_{\max}$ par la fonction $f_{C_{\max}}$ définie sur $\mathbb{R}^+$. Grâce à cette fonction, nous sommes en mesure de calculer la probabilité que $C_{\max}$ soit compris dans un intervalle $[C_1; C_2]$ avec $\int_{C_1}^{C_2} f_{C_{\max}}(x)dx$. La fonction de répartition de $C_{\max}$ est notée $F_{C_{\max}}$. Cette fonction est la primitive de la fonction de densité $f_{C_{\max}}$ telle que $F_M(0) = 0$. Ainsi, la probabilité que la durée totale de l'ordonnancement soit inférieure à une valeur C est $\Pr[C_{\max} \leq C] = F_{C_{\max}}(C)$.

3.2.4 Ordonnancement robuste

Notre objectif est de proposer des méthodes qui optimisent à la fois l'efficacité et la robustesse d'un ordonnancement en nous focalisant sur la durée totale. Nous devons pour cela définir la façon dont ces deux critères sont quantifiés à partir de la distribution de $C_{\max}$ pour un ordonnancement donné :

- Pour l'efficacité d'un ordonnancement, nous minimisons l'espérance de $C_{\max}$, notée $E(C_{\max})$. Cela permet de minimiser la durée totale de l'ordonnancement dans le cas moyen.
- Il existe de nombreuses méthodes dans la littérature pour évaluer la robustesse. Nous préciserons donc le concept de robustesse dans le cas de l'ordonnancement de graphes de tâches à durée aléatoire. Cependant, dans tous les cas, la mesure s'exprimera en fonction de la distribution de $C_{\max}$.

En définitive, le problème consiste à ordonnancer un graphe de tâche sur une plateforme hétérogène avec des durées aléatoires. Les critères à optimiser concernent la distribution de $C_{\max}$. Comme l'optimisation de l'efficacité dans le cas déterministe est un problème NP-Difficile [107] et que le calcul de la distribution de $C_{\max}$ est #P'-Complet, ce problème est donc dans la classe [1] NP-Complet$^{\#\text{P'-Complet}}$.

Enfin, en toute généralité, les deux critères ne sont pas équivalents. Plusieurs solutions Pareto-optimales peuvent donc exister, ce qui nécessite des méthodes bicritères.

3.3 Mesures de robustesse

Quantifier la robustesse est un problème qui n'a pas abouti à l'adhésion de la communauté scientifique sur une mesure spécifique. Pour commencer, nous présentons les travaux antérieurs relatifs aux mesures de robustesse. L'absence de consensus dans ce domaine nous motive à comparer empiriquement plusieurs approches que nous détaillons dans un second temps.

3.3.1 État de l'art

Les travaux de Wu et al. [152] constituent un point de repère dans l'étude des mesures de robustesse et présentent un intérêt méthodologique pour notre problème. Ils traitent de l'ordon-

1. un problème dans la classe A^B se résout par un algorithme dans la classe A avec un oracle qui résout un problème de la classe B [25].

nancement d'atelier lorsque les ressources peuvent devenir indisponibles aléatoirement pour une durée indéterminée. Bien que les durées des tâches soient connues, la durée totale de l'ordonnancement est une variable aléatoire. Les auteurs organisent leur étude en proposant plusieurs mesures qui correspondent à leur définition de la robustesse. Ils les comparent ensuite empiriquement similairement à l'approche que nous appliquons dans la section 3.5. Ils définissent la robustesse d'un système par sa capacité à conserver des performances élevées lors d'indisponibilités ponctuelles des ressources. Ils commencent donc par définir la mesure de performance qui est dans leur cas la même que la nôtre, i.e., la durée totale. L'objectif est de minimiser le retard par rapport à une valeur nominale, c'est-à-dire la durée totale en l'absence d'indisponibilité. Parmi les trois mesures de robustesse proposées, l'une se base sur l'hypothèse que chaque ressource ne devient indisponible qu'une seule fois. Dans ce cas, l'espérance du retard peut être évaluée analytiquement de façon efficace et cette valeur doit être minimisée. Dans notre cas, les durées sont aléatoires et considérer l'absence de perturbation n'est pas pertinent. Cette mesure n'est donc pas directement applicable car nos ordonnancements n'ont pas de durées totales nominales. Les autres mesures proposées par Wu et al. s'appuient sur ce même principe. En revanche, la façon dont nous abordons la caractérisation de la robustesse d'un ordonnancement et celle dont nous comparons un ensemble de mesures concordent avec l'étude qu'ils proposent.

Ali et al. décrivent dans [14] une méthode pour mesurer la robustesse qui consiste en une succession de quatre étapes :

1. définir quel type de performances nécessite d'être robuste,
2. identifier les paramètres impactant la robustesse,
3. identifier comment la variation de ces paramètres influence les performances,
4. identifier le plus petit ensemble de variations qui aboutit à des performances inacceptables.

Avec cette méthode, les auteurs définissent le *rayon de la robustesse* qui est la plus petite variation des paramètres telle que les valeurs de performances excèdent les limites tolérables. Dans notre cas (l'ordonnancement de tâches sur plateformes hétérogènes), la durée totale d'un ordonnancement généré caractérise la performance de nos solutions. Les paramètres qui impactent $C_{\max}$ sont les durées de chaque tâche et de chaque communication. Ainsi, un ordonnancement est plus robuste qu'un autre s'il requiert des changements dans les durées d'exécution de ses tâches plus importants pour excéder une limite donnée. Cette définition ne prend pas en compte les probabilités avec lesquelles les variations se produisent. De plus, le calcul du rayon de la robustesse est coûteux.

Pour simplifier l'évaluation de la robustesse, England et al. [63] proposent d'utiliser la distance de Kolmogorov-Smirnov entre la fonction de répartition de la mesure de performance dans des conditions normales de fonctionnement et la fonction de répartition de la mesure de performance lorsque des perturbations se produisent. Si la distance de Kolmogorov-Smirnov est large (proche de 1), alors les deux distributions sont suffisamment différentes et l'on peut affirmer que les perturbations ont un impact significatif sur le système étudié. Cependant, le système que nous avons modélisé ne possède pas de valeur nominale. Considérer l'absence d'incertitude revient à rendre déterministe chaque durée. La mesure de performance dans ces conditions ne prend qu'une seule valeur. La fonction de densité est alors une fonction δ de Dirac et la fonction de répartition est une fonction échelon. De plus, si la valeur de $C_{\max}$ sans perturbation est obtenue en considérant la valeur minimale de chaque durée, alors la distance de Kolmogorov-Smirnov est toujours égale à 1 quelque soit l'ordonnancement. Cela signifie que cette mesure de robustesse n'est pas adaptée lorsque la mesure de performance ne prend qu'une seule valeur possible dans des conditions normales. En plus de cela, l'incertitude est liée à l'estimation des durées d'exécution et non pas à des perturbations dans notre problème.

Shestak et al. [134] définissent la *métrique de robustesse stochastique*, que nous appelons *métrique stochastique*. Il s'agit de la probabilité qu'une valeur de performance soit comprise dans un intervalle donné. Ils comparent cette mesure au rayon de robustesse (dénotée *métrique de robustesse déterministe* dans leur article) et montrent que la métrique stochastique est préférable dans le cas de l'ordonnancement de tâches indépendantes.

D'autres définitions de la robustesse existent dans la littérature. Bölöni et Marinescu [31] proposent d'utiliser la *marge* comme mesure de robustesse. La marge d'une tâche représente le délai dont son exécution peut être retardée sans allonger la durée totale de l'ordonnancement. Ces auteurs suggèrent aussi d'utiliser l'entropie de la distribution de $C_{\max}$ pour comparer deux ordonnancements. Ils avancent que celui possédant la plus petite entropie est le plus robuste. Une autre définition de la marge, équivalente à celle de [31], est donnée dans [135]. Dans ce dernier article, deux nouvelles mesures sont d'ailleurs proposées pour le problème de l'ordonnancement. L'une se base sur l'espérance de la différence entre l'espérance de $C_{\max}$ et les valeurs que prend $C_{\max}$. L'autre est la proportion de scénarios qui sont en retard par rapport à l'espérance de $C_{\max}$. De plus, les auteurs montrent que minimiser la durée totale d'un ordonnancement est un objectif antagoniste avec l'optimisation de la robustesse.

3.3.2 Description des mesures

Nous détaillons dans cette section les mesures de robustesse qui seront comparées entre elles. Certaines mesures, comme le rayon de robustesse [14], ne sont toutefois pas adaptées à notre problème.

Rappelons que la robustesse se définit par la stabilité des performances du système quelque soient les aléas qu'il subit. Dans notre problème, il s'agit de la stabilité de la durée totale $C_{\max}$ de l'ordonnancement quelque soient les durées prises par les tâches.

Nous nous proposons d'étudier les mesures de robustesse suivantes :

Écart type de la durée totale L'écart-type de la distribution de la durée totale représente sa dispersion statistique. Plus cette valeur est faible, plus les valeurs possibles pour la durée totale sont proches les unes des autres. Cette mesure caractérise donc la robustesse. Sa formulation mathématique est

$$\sigma_{C_{\max}} = \sqrt{E(C_{\max}^2) - E(C_{\max})^2}$$

Entropie différentielle de la durée totale Mesurer l'entropie d'une distribution pour apprécier l'incertitude liée à une distribution est proposé dans [31]. Moins il existe d'incertitude, plus les valeurs que prend la variable $C_{\max}$ sont rapprochées, impliquant ainsi que l'ordonnancement est robuste.

$$h(C_{\max}) = -\int_{-\infty}^{+\infty} f_{C_{\max}}(x) \log f_{C_{\max}}(x) dx$$

Espérance de la marge Utiliser la marge (*slack* en anglais) est également proposé dans [31]. La marge d'une tâche est le délai qu'elle peut subir sans pour autant augmenter la durée totale. Le calcul de la marge d'une tâche nécessite de déterminer la marge libre associée à chaque communication $(t_i, t_{i'}) \in E$, c'est-à-dire la quantité de temps additionnelle qu'aurait pu prendre une communication sans que la tâche destinataire ne prenne de retard

$$T_{\text{marge}}(t_i, t_{i'}) = S_{i'} - C_i - D_{ii'}$$

où $S_{i'}$ est la date de début d'exécution de la tâche $t_{i'}$, C_i est la date de fin d'exécution de t_i et $D_{ii'}$ est la durée de la communication entre les tâches t_i et $t_{i'}$. À partir d'une tâche

donnée, nous considérons tous les chemins possibles qui vont jusqu'à la tâche finale et sur chaque chemin nous sommons les marges libres des communications rencontrées. La marge d'une tâche est le minimum de toutes ces sommes. La marge d'un ordonnancement est la somme des marges de toutes les tâches. Intuitivement, plus la marge d'un ordonnancement est grande, plus celui-ci est en mesure d'absorber l'incertitude et donc de conserver les mêmes performances. Comme les durées sont toutes aléatoires, la marge est aussi une variable aléatoire et nous ne retenons que son espérance en tant que mesure de robustesse.

Métrique stochastique La mesure de robustesse définie dans [134] est la probabilité que $C_{\max}$ soit dans un intervalle donné. Plus cette probabilité est élevée, plus l'ordonnancement est robuste. Nous proposons deux versions de cette mesure qui diffèrent par la façon de caractériser l'intervalle en question. La *métrique stochastique absolue* considère l'intervalle $[E(C_{\max}) - \delta, E(C_{\max}) + \delta]$ où δ est une constante positive donnée par l'utilisateur. La *métrique stochastique relative* utilise l'intervalle $[E(C_{\max}) \times \frac{1}{\gamma}, E(C_{\max}) \times \gamma]$ où la valeur de γ est supérieure à 1. Formellement, la métrique stochastique absolue se définit par

$$A(\delta) = \Pr\left[E(C_{\max}) - \delta \leq C_{\max} \leq E(C_{\max}) + \delta\right]$$

et la métrique stochastique relative par

$$R(\gamma) = \Pr\left[\frac{E(C_{\max})}{\gamma} \leq C_{\max} \leq \gamma \times E(C_{\max})\right]$$

Probabilité de retard Nous disons qu'un ordonnancement est en retard si sa durée totale excède une valeur donnée (l'espérance de $C_{\max}$ par exemple). La probabilité de retard [135] est une valeur que l'on cherche à minimiser. Elle se définit par

$$L = \Pr[C_{\max} > E(C_{\max})]$$

99$^{\text{ième}}$ centile de la durée totale Étant donné un ordonnancement, cette mesure, suggérée par [96], est la pire durée totale qu'il est possible d'obtenir dans 99% des cas les plus favorables. Il s'agit d'une mesure qui dénote à la fois l'efficacité d'un ordonnancement et sa robustesse.

3.4 Plan d'expérience

Notre étude empirique nécessite de caractériser l'ensemble des paramètres intervenant dans la génération du graphe de tâches, la génération de la plateforme et ceux inhérent à la simulation.

Les paramètres qui concernent le graphe de tâches et la plateforme sont donnés dans les tableaux 3.2 et 3.3 avec les valeurs qu'ils prennent. Chaque valeur testée correspond à une simulation tandis que toutes les combinaisons de valeurs par défaut sont utilisées. Plus précisément, nous réalisons le produit cartésien des valeurs par défaut, ce qui aboutit à trois scénarios de base (un pour chaque type de graphes). Pour chacun de ces scénarios, nous sélectionnons un paramètre et lui affectons l'une des valeurs testées.

3.4.1 Paramètres du graphe de tâches

Deux types de graphes sont générés aléatoirement et un type de graphes correspond à une application réelle, à savoir l'algorithme de Strassen qui réalise le produit de deux matrices. Tobita et al. [142] présentent deux manières de générer des graphes aléatoires : *samepred* avec laquelle

Paramètre	Valeurs par défaut	Valeurs testées
Type de graphes	samepred, layrpred, strassen	
Nombre de tâches (n)	1000	10, 100, 1000
Racine du générateur aléatoire	0	1, 2, 3, 4, 5, 6, 7, 8, 9
Espérance du nombre d'arcs par sommet	3	1, 5
Espérance des coûts d'exécution (FLOP)	100 M	10 M, 1 G
Espérance des coûts de communication (octet)	100 ko	10 ko, 1 M
Coefficient de variation des coûts	0,5	0,001, 0,1, 0,3, 1, 2
Distribution	bêta	exponentielle, normale
Espérance des niveaux d'incertitude	1,1	1,0001, 1,2, 1,5, 2, 3, 5
Coefficient de variation des niveaux d'incertitude	0,3	0,001, 0,1, 0,5, 1, 2

TABLE 3.2 – Paramètres et valeurs pour la génération d'un graphe de tâches

il peut exister une communication entre n'importe quelles tâches ; et, *layrpred* où les tâches sont organisées par niveau.

Pour chaque type de graphes, nous varions les coûts des tâches et des communications. Comme les graphes sont hétérogènes, il s'agit de coûts moyens sur l'ensemble des tâches et des communications d'un même graphe. Pour modéliser la disparité de ces coûts, nous utilisons le coefficient de variation qui définit le rapport entre l'écart-type et la moyenne d'un ensemble de valeurs (voir [129, Section 5.3.2.3] pour plus de détail sur cette mesure de dispersion relative). Dans notre cas, nous utilisons une distribution gamma avec une espérance et un coefficient de variation donnés pour générer les coûts d'un graphe spécifique. Par exemple, nous voyons dans le tableau 3.2 que l'espérance du coût des communications est par défaut 100 ko. Le coefficient de variation associé est 0,5. Les coûts de communication sont donc issus d'une loi gamma d'espérance 10^5 et d'écart-type 5×10^4.

Le coût de chaque tâche et de chaque communication est modélisé par une variable aléatoire dont nous fixons l'espérance, la distribution et le niveau d'incertitude, i.e., le rapport entre sa valeur maximale et celle minimale. Si ces valeurs ne sont pas finies, nous les remplaçons respectivement par le $99{,}9^{\text{ième}}$ centile et le $0{,}1^{\text{ième}}$ centile. Ce niveau d'incertitude caractérise la dispersion statistique d'un coût aléatoire. Plus le niveau est élevé, moins on aura de certitude sur la valeur d'un coût lors de l'exécution de l'ordonnancement. Les niveaux d'incertitude ne sont pas identiques pour toutes les tâches, et c'est pourquoi nous spécifions le coefficient de variation de ces niveaux.

La distribution d'un coût suit soit une loi bêta, soit une loi exponentielle, soit une loi normale. Les paramètres α et β de la loi bêta sont choisis de façon à ce que les valeurs faibles soient plus fréquentes que les valeurs élevées. Cela signifie que la distribution doit avoir une asymétrie positive pour la queue de la distribution s'étale à droite et donc $\alpha < \beta$. Pour avoir un mode non nul, il faut que $\alpha > 1$. Nous choisissons donc $\alpha = 2$ et $\beta = 5$.

Enfin, nous faisons varier des paramètres plus classiques comme le nombre de tâches, l'espérance du nombre d'arcs par sommet et la racine du générateur pseudo-aléatoire. Quelques

paramètres sont ignorés pour les graphes de type Strassen car la structure de ces graphes impose certaines propriétés. Les coûts des communications sont par exemple déjà induis par le nombre de tâches et les coûts d'exécutions. Le coefficient de variation de ces deux coûts est alors nul et l'espérance du nombre d'arcs par sommet est ignorée. Finalement, les nombres de tâches sont remplacés par 23, 163 et 1143. Ces nombres sont obtenus en faisant varier le niveau de récursion de 1 à 3 (le nombre de tâches dans le graphe de tâches de l'algorithme de Strassen est $\frac{10\times 7^r-1}{3}$ où r est le niveau de récursion).

3.4.2 Paramètres de la plateforme

Les paramètres pour la plateforme sont représentés dans le tableau 3.3. La plateforme est hétérogène et uniforme, c'est-à-dire que chaque machine possède une vitesse propre et que la durée d'exécution d'une tâche sur une machine est obtenue en divisant le coût de la tâche par la vitesse de la machine. Les espérances des vitesses, des latences et des bandes passantes ne prennent qu'une seule valeur. Nous agissons sur leurs coefficients de variation pour contrôler l'hétérogénéité de la plateforme. Un coefficient différent est utilisé pour les bandes passantes avec une valeur par défaut plus élevée de manière à représenter des vitesses de communication plus disparates.

Paramètre	Valeur par défaut	Valeurs testées
Nombre de processeur (n)	50	25, 100
Racine	0	1, 2
Espérance de la vitesse des processeurs (FLOPS)	2,5 G	
Espérance des latences (ms)	0.1	
Coefficient de variation des performances	0.5	0,001, 0,1, 0,3, 1, 2
Espérance des bandes passantes (o/s)	50 M	
Coefficient de variation des bandes passantes	1	0,001, 0,1, 0,3, 0,5, 2

TABLE 3.3 – Paramètres et valeurs pour la génération d'une plateforme hétérogène

En prenant en compte les paramètres du graphe de tâches et de la plateforme, il y a cinquante valeurs à tester. Comme il y a trois types de graphes de tâches à considérer par défaut, cela fait un total de $50 \times 3 = 150$ simulations.

3.4.3 Paramètres de la simulation

L'évaluation de la distribution de la durée totale $C_{\max}$ est requise pour appliquer la majorité des mesures présentées dans la section 3.3.2. Nous utilisons une méthode de Monte Carlo. Pour chaque variable aléatoire, une valeur est tirée suivant sa loi grâce au Mersenne Twister, un générateur pseudo-aléatoire [115]. Nous devons donc définir la quantité de tirages à réaliser pour obtenir une précision acceptable. La section 1.3.4 du chapitre 1 nous permet de définir ce nombre si l'on suppose que $C_{\max}$ suit une loi normale. Avec 20 000 tirages, l'erreur relative de l'écart-type de la durée totale est inférieure à 5% avec un degré de confiance de 99%. Avec ces 20 000 tirages, nous pouvons construire une fonction de répartition empirique de $C_{\max}$ qui sert de base aux calculs de la plupart des mesures de robustesse définies dans la section 3.3.2.

L'évaluation de l'entropie différentielle présente en revanche quelques écueils. Nous présentons deux approches qui consistent soit à borner la mesure, soit à l'estimer. Une borne inférieure triviale est $-\infty$. Cette borne s'obtient en considérant que l'ensemble des tirages provient d'une loi discrète pour laquelle l'entropie différentielle est $-\infty$. Une borne supérieure est proposée par DeStefano et al. [106]. Cette méthode considère les intervalles de confiance de chaque valeur de la fonction de répartition empirique et utilise une technique basée sur la mise en tension d'une corde imaginaire qui passerait à l'intérieur de chaque intervalle de confiance. Cette technique permet de produire la distribution ayant l'entropie différentielle la plus élevée et qui respecte les intervalles de confiance de la fonction de répartition empirique. Bien que cette technique fournisse une borne formelle, elle se révèle trop coûteuse en pratique. Nous lui préférons une méthode plus rapide qui suppose que chaque point de la fonction de répartition empirique correspond parfaitement à la réalité. Pour un nombre de simulation important, cette approximation est proche de la borne supérieure décrite dans [106].

Pour la métrique stochastique, nous avons choisi $\delta = 0,1$ et $\gamma = 1,005$ de façon à avoir des mesures d'ordonnancements correctement dispersées dans l'intervalle $[0; 1]$. Ces valeurs dépendent fortement des coûts et des niveaux d'incertitude considérés.

3.4.4 Mesures supplémentaires

Pour étudier la structure du problème, nous mesurons des informations supplémentaires pour chaque ordonnancement.

Écart type de la marge La marge est similaire à la durée totale dans le sens où la marge est aussi une variable aléatoire. Nous calculons donc également son écart-type.

Statistique Anderson-Darling Ce test est l'un des meilleurs pour tester la normalité d'un ensemble de valeurs. Nous l'appliquons sur la durée totale $C_{\max}$. Plus sa valeur est élevée, moins $C_{\max}$ est distribuée normalement.

3.4.5 Ordonnancement

Dans l'ensemble, nous avons 150 scénarios avec différents graphes et différentes plateformes. Dans chaque cas, nous construisons 5 000 ordonnancements aléatoires. Chaque ordonnancement est généré en répétant les trois étapes suivantes :

1. choisir aléatoirement une tâche parmi celles qui sont prêtes,
2. l'assigner à un processeur choisi aléatoirement et l'ordonnancer à la suite,
3. mettre à jour la liste des tâches prêtes.

Toutes les mesures sont estimées pour chaque ordonnancement. Pour un scénario donné, les mesures des ordonnancements sont d'abord comparées visuellement, puis nous appliquons le test de corrélation de Spearman.

3.5 Comparaison des mesures de robustesse

Dans cette section, nous étudions empiriquement les mesures présentées dans la section 3.3.2 ainsi que les liens entre celles-ci. En particulier, nous souhaitons déterminer la corrélation entre deux mesures, c'est-à-dire la corrélation entre deux séries de valeurs produites par deux mesures données en utilisant les mêmes ordonnancements.

3.5.1 Résultats empiriques

Les figures 3.5.1 à 3.5.3 montrent les corrélations entre les mesures. Chaque figure correspond au scénario de simulation avec les valeurs par défaut pour un type de graphe donné. Sur ces figures, neuf mesures sont comparées les unes aux autres. Pour chaque scénario, nous obtenons donc une matrice de 81 graphiques. Plutôt que de confronter chaque mesure à elle-même, le nom de chaque mesure est donné sur la diagonale. Les 36 graphiques de la partie inférieure-gauche contiennent toutes les informations car la matrice est symétrique.

Les valeurs produites par l'application de chaque mesure sur les ordonnancements aléatoires sont donc tracées sur la partie inférieure-gauche. Par exemple, l'espérance de la durée totale de chaque ordonnancement aléatoire est représentée en fonction de l'entropie différentielle de la durée totale sur le graphique de la première colonne à la troisième ligne (l'espérance en abscisse et l'entropie en ordonnée).

Pour faciliter la lecture de ces graphiques, les valeurs de trois mesures sont inversées pour que l'optimisation consiste toujours en une minimisation de ces valeurs (les mesures des meilleurs ordonnancements se positionnent donc dans le coin inférieur-gauche de chaque graphique). Ces trois mesures sont les deux métriques stochastiques et l'espérance de la marge car nous faisons l'hypothèse qu'un ordonnancement robuste possède une marge importante. L'inversion est réalisée en multipliant par -1 l'espérance de la marge d'un ordonnancement et en soustrayant la mesure de l'ordonnancement à 1 pour les métriques stochastiques. D'autre part, le $99^{\text{ième}}$ centile de la durée totale est omis car cette mesure est quasiment équivalente à l'espérance de la durée totale.

Par ailleurs, un spline cubique qui approche les valeurs est tracé sur chaque graphique de la matrice. Cette courbe souligne la corrélation entre deux mesures lorsqu'elle existe. Finalement, nous avons ajouté l'ordonnancement produit par l'algorithme HEFT à l'ensemble des ordonnancements aléatoires (les mesures de cet ordonnancement sont caractérisées par un carré rouge). Cela permet de tester si les comportements observés avec des ordonnancements aléatoires sont cohérents avec un ordonnancement proche de l'optimal.

La partie supérieure-droite de la matrice contient les coefficients de Spearman des mesures spécifiées par la ligne et la colonne. Plus une corrélation est forte, plus la valeur absolue du coefficient de Spearman est proche de 1. Un coefficient négatif signifie que les mesures sont négativement corrélées : si une mesure d'un ordonnancement est élevée, alors l'autre mesure de ce même ordonnancement est faible (et inversement). Par exemple, l'espérance de la durée totale et l'espérance de la marge sont négativement corrélées avec une valeur de $-0,92$ sur la figure 3.5.1.

Comme ces coefficients de Spearman caractérisent numériquement les corrélations entre les mesures, nous nous en servons pour agréger nos résultats. La figure 3.5.4 résume l'ensemble des coefficients de Spearman sur les 150 scénarios en suivant la même structure que les figures 3.5.1 à 3.5.3. Les moyennes de chaque coefficient de corrélation entre deux mesures sont représentées dans la partie supérieure-droite de la matrice tandis que les écart-types sont dans la partie inférieure-gauche. Nous voyons par exemple que l'écart-type de la marge et l'entropie différentielle de la durée totale sont fortement corrélés (la moyenne des coefficients de Spearman vaut $1,00$) avec un écart-type négligeable $(0,01)$.

3.5.2 Interprétation

3.5.2.1 Supposition de la normalité

Nous remarquons sur la figure 3.5.4 la forte corrélation entre l'écart-type de la durée totale, l'entropie différentielle de la durée totale et la métrique stochastique absolue. Les faibles écart-

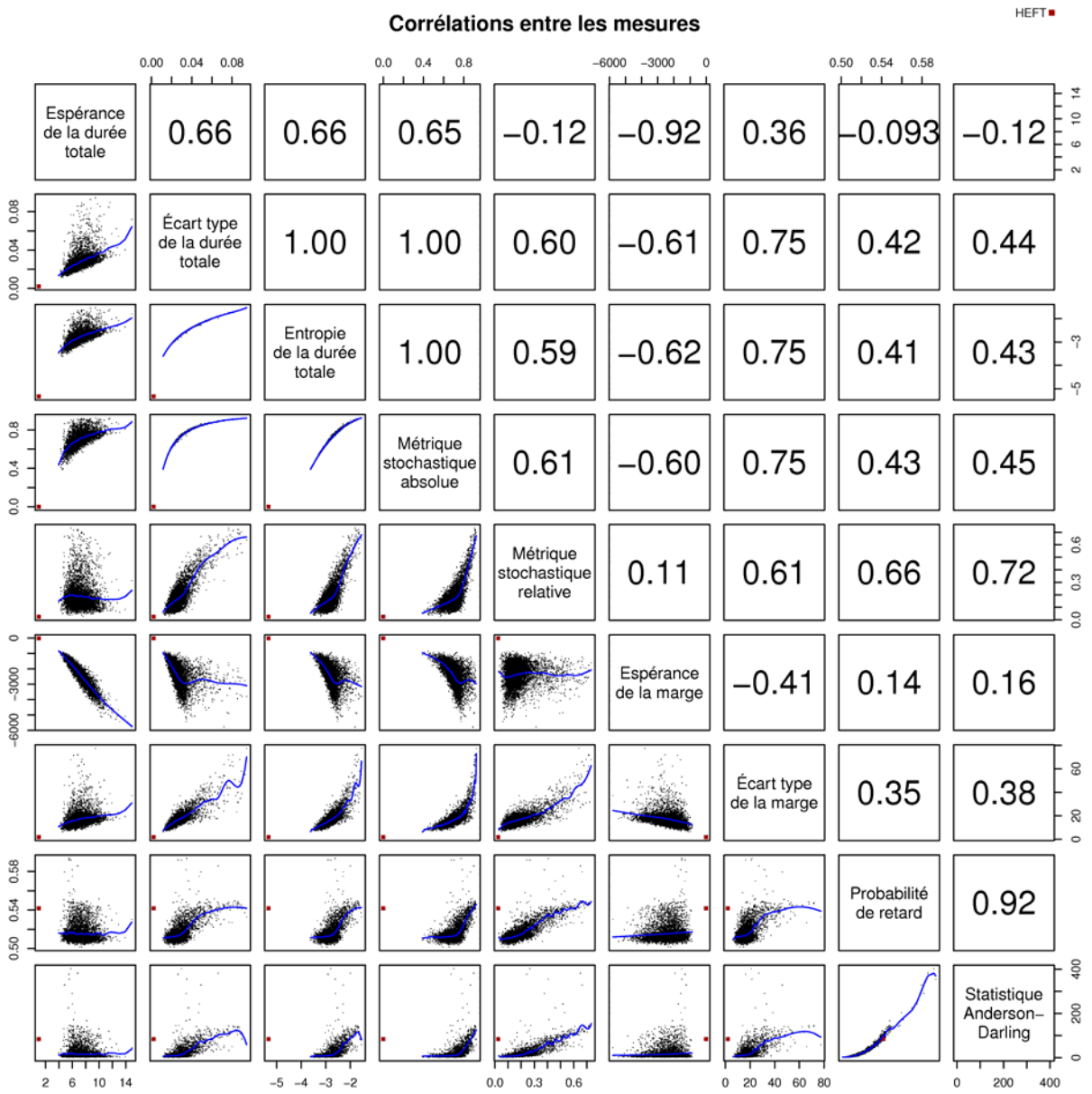


FIGURE 3.5.1 – Corrélations entre les mesures de robustesse pour les graphes de type samepred avec les valeurs par défaut. Partie inférieure-gauche : graphiques avec 5 000 ordonnancements aléatoires. Partie supérieure-droite : valeurs des coefficients de Spearman.

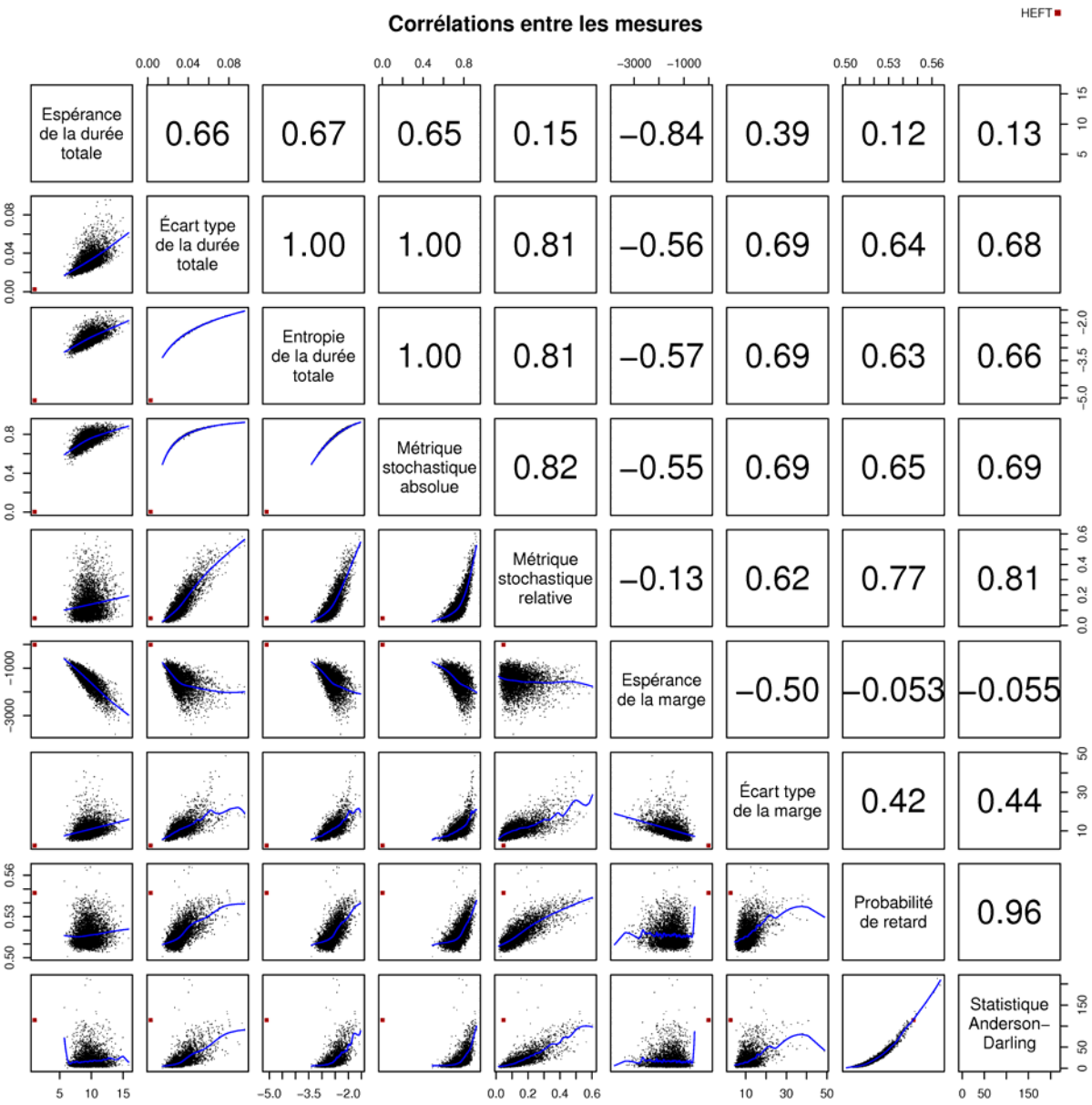


FIGURE 3.5.2 – Corrélations entre les mesures de robustesse pour les graphes de type layrpred avec les valeurs par défaut. Partie inférieure-gauche : graphiques avec 5 000 ordonnancements aléatoires. Partie supérieure-droite : valeurs des coefficients de Spearman.

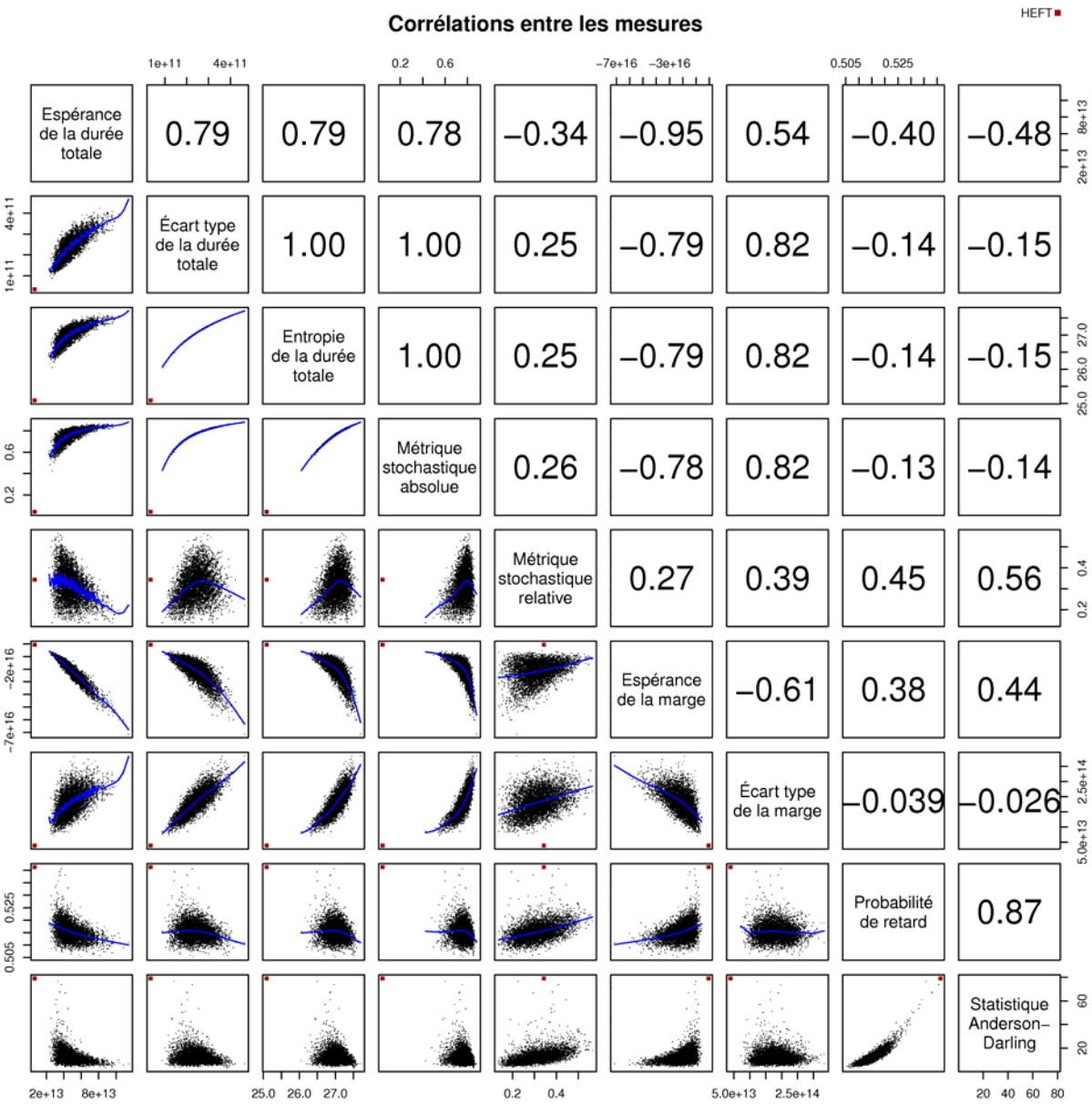


FIGURE 3.5.3 – Corrélations entre les mesures de robustesse pour les graphes de type strassen avec les valeurs par défaut. Partie inférieure-gauche : graphiques avec 5 000 ordonnancements aléatoires. Partie supérieure-droite : valeurs des coefficients de Spearman.

Résumé des corrélations

Espérance de la durée totale	0.73	0.74	0.70	−0.07	−0.91	0.49	−0.12	−0.15
0.12	Écart type de la durée totale	1.00	0.97	0.53	−0.70	0.79	0.23	0.24
0.11	0.01	Entropie de la durée totale	0.97	0.53	−0.70	0.79	0.22	0.23
0.13	0.11	0.11	Métrique stochastique absolue	0.55	−0.67	0.77	0.25	0.27
0.30	0.29	0.29	0.27	Métrique stochastique relative	0.02	0.54	0.56	0.63
0.05	0.13	0.13	0.15	0.29	Espérance de la marge	−0.58	0.12	0.15
0.17	0.10	0.10	0.13	0.22	0.17	Écart type de la marge	0.22	0.24
0.27	0.35	0.34	0.34	0.19	0.27	0.27	Probabilité de retard	0.90
0.30	0.37	0.36	0.35	0.18	0.30	0.28	0.11	Statistique Anderson−Darling

FIGURE 3.5.4 – Moyennes (partie supérieure-droite) et écart-types (partie inférieure-gauche) des coefficients de Spearman sur les 150 scénarios de simulation

types des coefficients de corrélation indiquent que ces relations sont présentes dans l'ensemble des graphes générés quelque soient leurs tailles, leurs types et les niveaux d'incertitude considérés.

Le lien entre l'écart-type, l'entropie et la métrique stochastique absolue suggère que les durées totales de tous les ordonnancements partagent des caractéristiques similaires en terme de densité de probabilité. Par exemple, leurs distributions peuvent suivre les mêmes lois mais avec des paramètres différents. Nous expliquons cette corrélation en nous basant sur le théorème central de la limite. Celui-ci établit que la somme de variables aléatoires d'espérances et d'écart-types finis converge vers une loi normale. Notons que la durée totale s'obtient en réalisant une succession de maximums et de sommes. En supposant que les maximums aient un impact négligeable et que suffisamment de sommes soient réalisées, alors la durée totale suit approximativement une loi normale.

Nous nous proposons de vérifier cette hypothèse en caractérisant la normalité de chaque durée totale avec le test d'Anderson-Darling. L'histogramme des statistiques ainsi obtenues pour les ordonnancements aléatoires est tracé sur la figure 3.5.5. La moitié des distributions possèdent une valeur inférieure à $12,0$. Une loi de Student avec $11,5$ degrés de liberté obtient la même valeur pour ce test et est visuellement très proche d'une loi normale comme l'atteste la figure 3.5.6. D'autre part, 90% des statistiques Anderson-Darling relatives aux durées totales sont inférieures à $79,5$ et sont donc plus proches d'une loi normale que d'une loi de Weibull de paramètres $\lambda = 1$ et $k = 2,17$. En pratique, une loi de Weibull de paramètre $k > 3,4$ peut être considérée comme une loi normale. Bien que les durées totales ne soient pas rigoureusement des lois normales, elles en sont suffisamment proches pour que l'on puisse raisonnablement baser l'interprétation des résultats obtenus dans la section 3.5.1 en supposant que chaque durée totale suit une loi normale. Cependant, l'exactitude de cette supposition dépend fortement de la normalité des durées des tâches. Par exemple, elle est contestable lorsque les durées suivent des lois exponentielles. Dans ce cas, la médiane des statistiques obtenues est $97,6$, ce qui est même supérieur à la statistique de la loi de Weibull mentionnée plus haut.

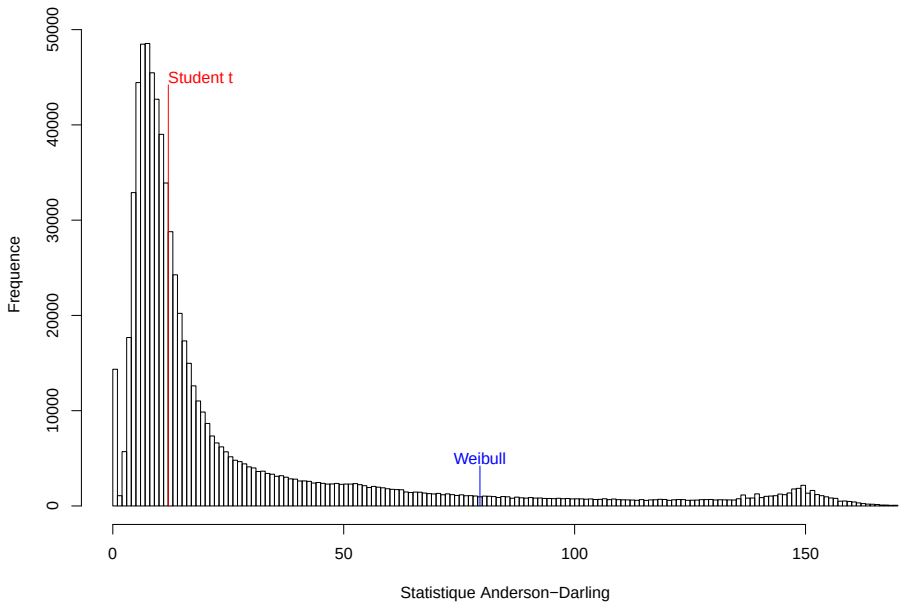


FIGURE 3.5.5 – Histogramme des statistiques Anderson-Darling des distributions des durées totales sur les 150 scénarios

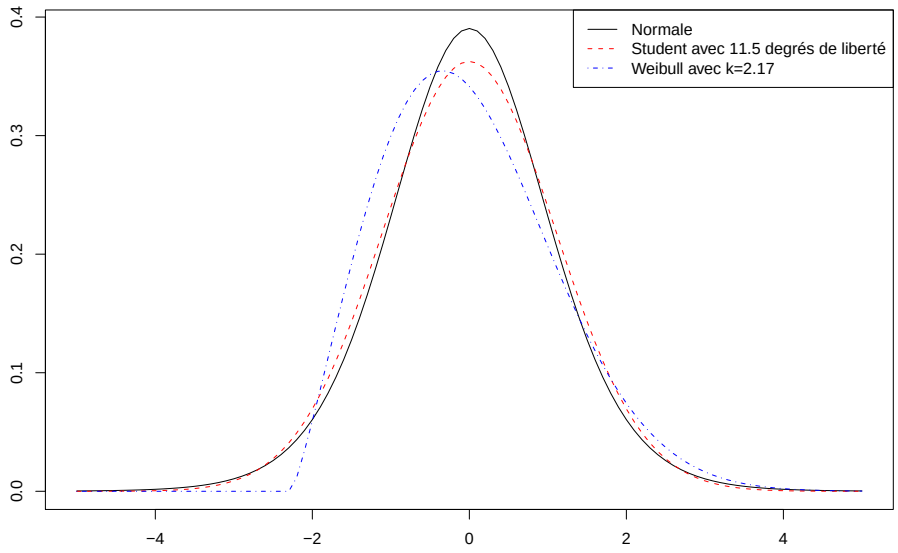


FIGURE 3.5.6 – Trois fonctions de densité de probabilité de lois possédant la même espérance et le même écart-type

Bien qu'il existe quelques cas où la distribution de la durée totale s'écarte significativement d'une loi normale, la corrélation entre l'écart-type, l'entropie et la métrique stochastique absolue s'explique aisément en supposant la normalité de la durée totale. L'entropie différentielle d'une loi normale s'écrit $\log(\sigma\sqrt{2\pi e})$ et dépend donc directement de l'écart-type σ. Il est également possible de formuler la métrique stochastique absolue en fonction de l'écart-type pour une loi normale.

Nous analysons la métrique stochastique relative en supposant la normalité de la durée totale. Cette mesure est liée aux trois précédentes, particulièrement dans le cas des graphes aléatoires (voir les figures 3.5.1 et 3.5.2). Bien que significatifs, les coefficients de corrélation entre cette mesure et l'écart-type sont en moyenne faibles et dispersés. La figure 3.5.4 donne une moyenne valant $0,53$ et un écart-type valant $0,29$. Les plus faibles coefficients de corrélation sont obtenus pour les graphes de type Strassen. L'exploitation des résultats a permis de montrer que cette mesure est équivalente au coefficient de variation de la durée totale, à savoir, le rapport entre l'écart-type et l'espérance de la durée totale. Les coefficients de Spearman obtenus entre la métrique stochastique relative et le coefficient de variation ont pour moyenne $1,00$ et pour écart-type $0,01$. Comme pour la métrique stochastique absolue, cette mesure peut s'exprimer en fonction de l'espérance et l'écart-type dans le cas d'une loi normale, ce qui explique cette corrélation quasiment parfaite.

Finalement, la statistique Anderson-Darling, utilisée pour valider la supposition de la normalité, est corrélée à la probabilité de retard. Cette dernière mesure est en effet liée à la différence entre l'espérance d'une loi et sa valeur médiane, c'est-à-dire son asymétrie. Or ces deux valeurs sont identiques pour une loi normale. Ainsi, plus la probabilité de retard diverge de $0,5$, plus la durée totale se distingue d'une normale et plus la statistique d'Anderson-Darling prend une valeur élevée.

3.5.2.2 Espérance et écart-type de la durée totale

La corrélation que l'on constate sur la figure 3.5.4 entre l'espérance et l'écart-type de la durée totale révèle une propriété importante. Cette corrélation varie en fonction du type de graphes mais reste manifeste. Le tableau 3.4 résume les coefficients de corrélation obtenus sur l'ensemble des simulations entre ces deux mesures pour des ordonnancements aléatoires. Pour un ensemble de valeurs, un résumé en cinq nombres donne les valeurs extrêmes et tous les quartiles (celui du milieu étant aussi la valeur médiane). Nous constatons que 25% des coefficients sont inférieurs à $0,78$ pour les graphes de type Strassen. Pour ces graphes, l'espérance et l'écart-type de la durée totale sont donc fortement corrélés.

Type de graphes	Minimum	25%	Médiane	75%	Maximum
layrpred	0,27	0,65	0,70	0,79	0,94
samepred	0,29	0,66	0,70	0,78	0,95
strassen	0,30	0,78	0,79	0,79	0,97

TABLE 3.4 – Résumé en cinq nombres des coefficients de corrélation par type de graphes

Pour expliquer cette corrélation, nous décrivons un phénomène qui intervient lorsque nous évaluons la densité de probabilité de la durée totale. La variance d'une variable aléatoire qui résulte d'une somme de deux variables aléatoires indépendantes est la somme des variances de ces variables. Si nous ignorons l'influence des opérations de maximum, cela a pour conséquence directe que l'écart-type de la durée totale dépend des variances des durées des tâches présentes sur le chemin le plus critique. Dans un graphe donné, chacune de ces variances est générée par l'intermédiaire du niveau de certitude et est donc plus ou moins proportionnelle à l'espérance du coût de la tâche considérée. Un ordonnancement avec une durée totale courte est susceptible d'avoir peu de tâches et/ou des tâches courtes sur son chemin le plus critique et donc un faible écart-type de la durée totale. L'imperfection de cette corrélation peut provenir de la présence des maximums et aussi de l'hétérogénéité des niveaux d'incertitude.

Les critères évalués par l'espérance et l'écart-type de la durée totale ne sont donc pas antagonistes sans pour autant être équivalents.

3.5.2.3 Espérance de la marge

Nous discutons dans cette section des corrélations entre l'espérance de la marge et deux autres mesures. La première est l'espérance de la durée totale. Un ordonnancement efficace, c'est-à-dire dont la durée totale est courte, aura tendance à avoir une petite marge. Intuitivement, dans un tel ordonnancement, les processeurs ne sont pas souvent inactifs et donc la marge de chaque tâche est réduite au maximum. Inversement, un ordonnancement avec beaucoup de marge n'exploite pas totalement les ressources et possède donc une durée totale sous-optimale. La figure 3.5.4 donne un coefficient de corrélation moyen négatif car chaque espérance de la marge est inversée.

La corrélation entre l'espérance de la marge et l'écart-type de la durée totale est surprenante. Nous avions fait l'hypothèse qu'un ordonnancement avec une marge élevée était robuste (plusieurs tâches pouvant être retardées sans incidence sur l'efficacité globale). Or, il apparaît que plus un ordonnancement possède de marge, plus son écart-type est important. Maximiser la marge entre donc en conflit avec la minimisation de l'écart-type (et donc de l'entropie et de la métrique stochastique absolue). Par conséquent, la marge et l'écart-type ne peuvent pas être toutes les deux des mesures de robustesse appropriées. Nous nous proposons ainsi d'analyser la pertinence de la marge en tant que mesure de robustesse.

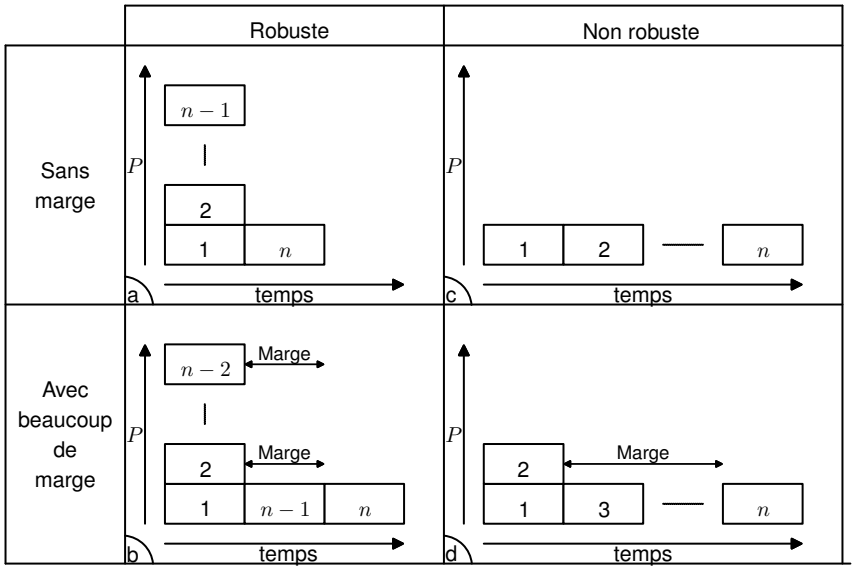


FIGURE 3.5.7 – Quatre ordonnancements plus ou moins robustes et avec différentes marges. Un total de $n-1$ tâches dont dépend une dernière tâche n sont ordonnancées sur un ensemble de processeurs P en considérant que les durées d'exécution sont des variables aléatoires i.i.d.

Nous présentons une étude de cas sur la figure 3.5.7 qui caractérise différentes situations possibles relativement à la marge et la robustesse d'un ordonnancement. Quatre ordonnancements sont proposés, chacun pour le même graphe de tâches, à savoir $n-1$ tâches indépendantes dont dépend une dernière tâche n. Les durées de toutes ces tâches sont identiquement et indépendamment distribuées (i.i.d.). Les ordonnancements c et d sont peu robustes aux aléas dans les durées des tâches car tout retard (à l'exception de celui touchant la tâche 2 pour l'ordonnancement d) se répercute sur la durée totale. L'ordonnancement b est robuste car seulement trois tâches sont sur le chemin critique et ont un impact sur la durée totale en cas de retard d'une de ces tâches. L'interprétation de l'ordonnancement a s'appuie sur les caractéristiques du maximum de plusieurs variables aléatoires similaires. Dans le cas du maximum de deux variables aléatoires i.i.d., l'espérance résultante de l'opération est supérieure à celles des variables initiales, et surtout l'écart-type résultant est inférieur à celui des variables initiales. Plus généralement, le maximum d'un nombre infini de variables aléatoires i.i.d. définies sur un support compact ou semi-infini (avec une borne supérieure finie) converge vers une fonction δ de Dirac (ce qui est parfaitement robuste) dont la valeur est la borne supérieure du support des variables initiales. Ainsi, plus il existe de tâches dont il faut attendre la finalisation, plus nous pouvons être sûrs qu'au moins une d'entre elles sera en retard. Et donc, l'ordonnancement a est robuste car nous avons plus de certitude sur la valeur de la durée totale.

Avec ces quatre exemples, nous constatons que la marge n'est pas nécessairement liée à la robustesse. La description de ces cas justifie notre rejet de la marge comme mesure de robustesse. De plus, nous avons précédemment remarqué que les espérances de la marge et de la durée totale sont négativement corrélées et que l'espérance de la durée totale et son écart-type sont positivement corrélés. Ces résultats sont donc cohérents.

Mesurer l'écart-type de la marge sert à obtenir une information plus complète sur la densité de probabilité de la marge. Sur la figure 3.5.4, la corrélation entre l'espérance et l'écart-type de la marge n'est pas négligeable, sans pour autant être importante. La raison en est la même que pour la durée totale. Une corrélation élevée existe entre les écart-types de la marge et de la durée totale, l'entropie et la métrique stochastique absolue. Cela est dû au calcul de la marge

qui consiste à agréger des intervalles de temps entre des dates de début et de fin d'exécution, l'écart-type de la marge dépendant principalement des écart-types de ces dates.

3.5.2.4 Remarques finales

L'équivalence entre le $99^{\text{ième}}$ centile et l'espérance de la durée totale vient du fait que les écart-types des durées des tâches sont faibles et donc l'écart-type de la durée totale aussi. Pour un ordonnancement donné, ces deux mesures ne diffèrent donc pas significativement.

Les mesures des ordonnancements produits par l'heuristique HEFT (carrés rouges sur les figures 3.5.1 à 3.5.3) corroborent nos observations. Lorsqu'il y a corrélation, les valeurs relatives à l'heuristique HEFT se situent dans le prolongement du spline cubique qui approche les points. C'est le cas notamment du graphique qui confronte l'entropie et la métrique stochastique absolue dans la figure 3.5.3.

3.5.3 Sélection de la mesure de robustesse

En nous basant sur les observations précédentes, nous tirons les conclusions suivantes. D'abord, l'écart-type de la durée totale, l'entropie différentielle de la durée totale et la métrique stochastique absolue ont un lien de dépendance fort. Ensuite, l'espérance de la marge n'est pas une mesure qui dénote la robustesse telle que définie dans la section 3.2.4.

Par ailleurs, l'écart-type de la durée totale est une mesure facile à évaluer lorsque la distribution de la durée totale est estimée par une loi normale. Les tests de normalité réalisés dans la section 3.5.2.1 justifie cette estimation, toute proportion gardée. Ainsi, parmi l'ensemble des mesures de robustesse considérées, l'écart-type est représentatif et s'obtient directement à partir d'une loi normale.

L'idée de la métrique stochastique relative est de tolérer une incertitude plus forte pour les ordonnancements longs que pour ceux qui sont courts. Si cette définition de la robustesse est considérée, alors calculer l'espérance et l'écart-type de la durée totale est suffisant car la métrique stochastique relative peut être remplacée par le coefficient de variation de la durée totale (rapport de l'écart-type sur l'espérance).

3.6 Méthodes d'ordonnancement

Nous présentons quelques travaux traitant de l'optimisation proprement dite, c'est-à-dire de l'ordonnancement efficace et robuste de tâches dans des conditions incertaines.

3.6.1 État de l'art

Plusieurs types d'approches ont été caractérisés et nous présentons des méthodes qui en sont représentatives. Quelques travaux [48, 84, 85] inventorient de façon plus exhaustive les méthodes d'ordonnancement conçues pour gérer l'incertitude.

Il existe de nombreuses méthodes d'ordonnancement statique qui se basent sur l'insertion de marges dans la solution finale [70, 49, 116]. Ce type de techniques procède en augmentant la durée des tâches avant l'étape d'ordonnancement. Les marges ajoutées jouent ainsi le rôle de protection temporelle : chaque tâche peut accuser un retard sans conséquence si celui-ci est inférieur à la marge de la tâche. Obtenir un ordonnancement dont l'exécution entière, y compris chaque date d'exécution, est robuste ne fait pas partie de nos objectifs car notre critère de performance est la durée totale. Ainsi, l'insertion de marge à l'intérieur de l'ordonnancement ne nous apporterait pas de stabilité supplémentaire vis-à-vis de ce critère. Nous estimons d'autre part qu'ajouter une

marge à l'ensemble de l'ordonnancement est une technique qui sacrifie exagérément l'efficacité au profit de la robustesse.

Plusieurs contributions mettant en avant des politiques d'ordonnancement dynamique ont été proposées dans [87, 153, 137]. Les décisions d'ordonnancement (le choix du processeur qui exécute une tâche donnée) sont alors prises pendant l'exécution de l'application. Les décisions sont donc adaptées aux durées d'exécution constatées. On évite ainsi d'handicaper toute une exécution à cause d'une décision statique qui n'est pas pertinente dans certaines situations extrêmes. Certains travaux proposent de se baser sur un ou plusieurs ordonnancements statiques et de composer avec en cours d'exécution. Il s'agit d'une approche complémentaire à la nôtre.

L'analyse de sensibilité consiste à étudier la réaction d'un système à une variation des paramètres d'entrée. Hall et al. [80] appliquent cette démarche dans le cadre de l'ordonnancement et décrivent une méthode pour augmenter la robustesse d'un ordonnancement optimal en utilisant un ensemble d'ordonnancements optimaux. L'analyse de sensibilité de techniques d'ordonnancement à base de partitionnement convexe est réalisée dans [127, Chapitre 4]. Ce type d'analyse est en revanche moins général que l'usage de variables aléatoires car les probabilités associées à chaque durée possible ne sont pas prises en compte.

D'autres travaux [39, 65] considèrent l'utilisation de la logique possibiliste [59] qui permet de modéliser l'imprécision grâce à des distributions possibilistes. Un degré de possibilité peut être vu comme la borne supérieure d'une probabilité. Une distribution possibiliste encode donc une famille de distributions de probabilité. Fargier et al. [65] applique la logique floue à l'ordonnancement de réseaux de projet car l'évaluation de la distribution possibiliste de la durée totale est plus facile que celle de sa distribution de probabilité. Ils proposent d'utiliser des règles de décision pessimistes pour obtenir un ordonnancement robuste. Nous pensons toutefois que cette approche ne peut pas fournir le même niveau de détail que l'usage de distributions de probabilité.

Pour clore cette description des travaux existants, nous abordons des approches plus spécifiques. Kim et al. [96] proposent un algorithme pour l'ordonnancement de réseaux de projet. Avec leur approche, chaque durée aléatoire est réduite à une valeur déterministe et le problème est ensuite résolu. L'aspect stochastique est pris en compte car la valeur déterministe est générée à partir de l'espérance et de la variance de la durée aléatoire. Gerasoulis et al. [71] étudient la stabilité de l'algorithme DSC (Dominant Sequence Clustering [72]) lorsque les durées des tâches et des communications sont sujettes à des incertitudes bornées. Leur étude est spécifique à une application irrégulière unique. Enfin, la robustesse est au centre de l'étude réalisée dans [47] mais cette dernière se focalise sur l'ordonnancement de tâches indépendantes sur un processeur unique.

3.6.2 Description des heuristiques

Les fonctions objectives sont d'abord explicitées, puis une heuristique gloutonne et un algorithme évolutionnaire sont présentés.

3.6.2.1 Fonctions objectives

L'heuristique CorLCA décrite dans la section 1.5.1 du chapitre 1 permet d'obtenir une estimation précise de la distribution de la durée totale $C_{\max}$. Cette heuristique étant peu coûteuse, elle se prête bien aux heuristiques décrites dans cette section car celles-ci invoquent un grand nombre de fois le mécanisme d'évaluation.

La première fonction objective correspond au critère d'efficacité et sa mesure est directement obtenue par l'espérance de $C_{\max}$. La mesure de robustesse retenue dans la section 3.5.3 est l'écart-type de $C_{\max}$. La représentativité de cette mesure et la facilité de son calcul motivent ce

choix. Le problème d'optimisation que nous posons consiste à minimiser à la fois l'espérance et l'écart-type de la durée totale d'un ordonnancement.

3.6.2.2 Heuristiques gloutonnes

La section 2.3.4 du chapitre 2 décrit une heuristique multicritère basée sur HEFT qui ordonnance un graphe de tâches sur une plateforme hétérogène. Le premier objectif concorde avec la mesure de l'efficacité donnée dans ce chapitre. Nous spécifions que le second objectif à optimiser est l'écart-type de la durée totale d'un ordonnancement.

Cette heuristique est nommée Rob-HEFT lorsque l'heuristique CorLCA est utilisée pour estimer la distribution de la durée totale. Une autre version de cette heuristique, appelée Rob-HEFT-MC, réalise des simulations de Monte Carlo (voir section 1.3.4) pour l'évaluation de la distribution de la durée totale.

3.6.2.3 Métaheuristique multicritère

Nous utilisons l'algorithme évolutionnaire convergeant décrit dans la section 2.4.4. Comme l'encodage des solutions et les opérateurs de mutation et de croisement sont spécifiés, il ne reste que les fonctions objectives à définir. Nous utilisons l'espérance et l'écart-type obtenus grâce à CorLCA comme cela est évoqué plus haut.

Rappelons que CorLCA ne fournit pas un résultat exact. Il existe des méthodes qui prennent en compte les incertitudes auxquelles sont soumises les valeurs objectives dans les algorithmes évolutionnaires. Cela permet d'en améliorer l'efficacité (voir [89] pour un état de l'art sur ce sujet). Nous gérons l'inexactitude du mécanisme d'évaluation en conservant dans une archive les trois meilleurs ensembles d'ordonnancements non-dominés (au lieu d'un seul). Ainsi, nous conservons davantage d'ordonnancements et augmentons la probabilité que certains d'entre eux soient véritablement optimaux.

Une autre façon d'aborder ce problème est de remarquer que la métaheuristique va orienter sa recherche vers les ordonnancements pour lesquels le mécanisme d'évaluation fournit des valeurs objectives favorables. Potentiellement, il est possible que ces ordonnancements soient ceux pour lesquels l'évaluation est la moins précise et non pas ceux qui sont les meilleurs vis-à-vis des deux objectifs. En conservant des ordonnancements qui semblent dominés (du point de vue de leurs valeurs objectives estimées), nous avons plus de chance d'aboutir à de bons ordonnancements une fois qu'ils seront évalués plus précisément par des simulations de Monte Carlo.

3.7 Comparaison des méthodes d'ordonnancement

Nous utilisons le même plan d'expérience que celui décrit dans la section 3.4. Pour chacun des 150 scénarios, toutes les heuristiques et la métaheuristique sont utilisées.

3.7.1 Paramètres des méthodes

Pour la métaheuristique, la population se compose de 200 chromosomes et évolue sur 1000 générations (2500 générations lorsque l'objectif est d'approcher le front de Pareto). Les probabilités de croisement et de mutation sont 0,25 et 0,35, respectivement. Nos travaux se basent sur une version de l'implémentation de NSGA-II dans la plateforme ParadisEO [37]. Le paramètre l de la métaheuristique vaut 1.

Pour les heuristiques Rob-HEFT et Rob-HEFT-MC, le paramètre de compromis θ varie de 0 à 1 en utilisant le mécanisme de recherche dichotomique décrit dans la section 2.3.3 du chapitre 2. La différence minimale entre deux valeurs du paramètre θ est $0,01$.

3.7.2 Espace des valeurs objectives

Notre première approche pour étudier la structure du problème consiste à analyser l'espace des valeurs objectives en caractérisant les frontières au delà desquelles les valeurs objectives ne sont atteignables par aucun ordonnancement. On obtient une estimation de ces frontières avec la métaheuristique et elles sont dénotées SO, NO, NE et SE d'après les points inter-cardinaux. Comme nous minimisons les deux critères, la frontière SO correspond à l'estimation du front de Pareto. La figure 3.7.1 illustre l'espace des valeurs objectives pour un scénario de simulation donné. Les mesures des ordonnancements aléatoires décrits dans la section 3.5.1 sont également représentées. La frontière NE n'est pas représentée car elle se situe dans une région trop éloignée.

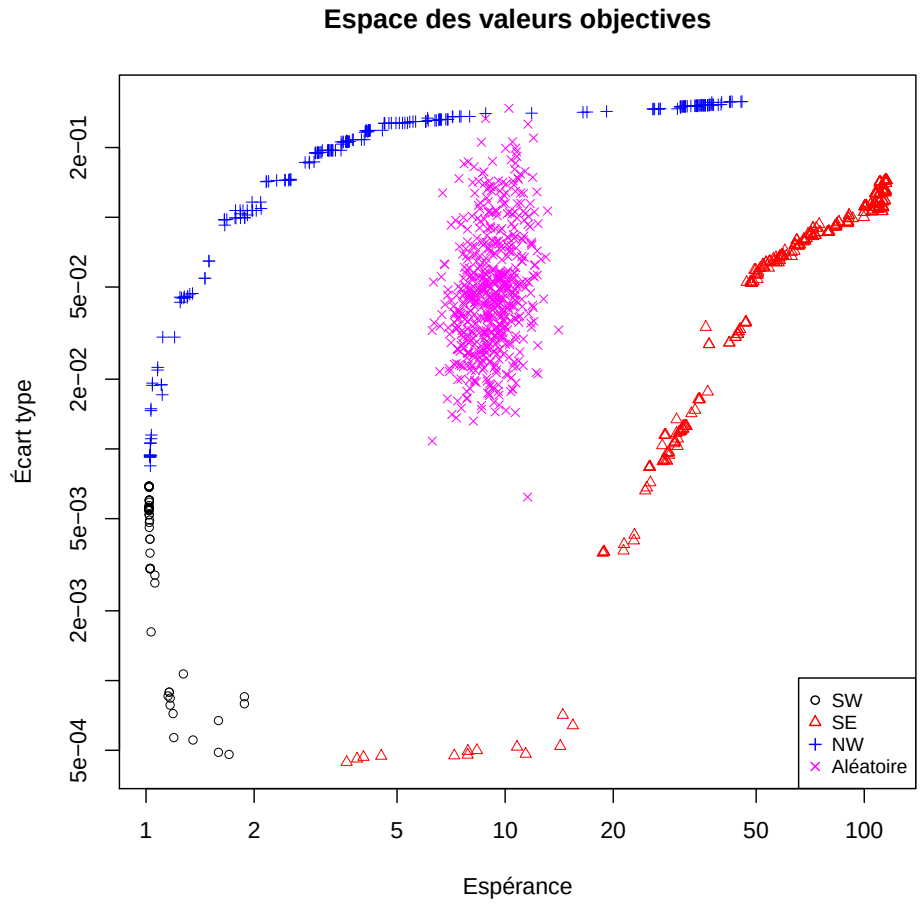


FIGURE 3.7.1 – Espérances et écart-types de la durée totale d'ordonnancements proches des frontières et d'ordonnancements aléatoires pour un graphe de 1000 tâches de type layrpred avec un coefficient de variation du niveau d'incertitude valant 2

Nous notons que l'extension de la frontière SO est faible par rapport aux autres frontières. Nous en déduisons que les solutions Pareto-optimales sont proches les unes des autres relativement aux dimensions de l'espace des valeurs objectives. En considérant les autres scénarios

de simulation, nous constatons que la frontière SO est d'autant plus large que le coefficient de variation du niveau d'incertitude est important. Cela signifie que l'hétérogénéité des incertitudes liées aux durées influence l'extension du front de Pareto.

Remarquons enfin que la figure 3.7.1 confirme les conclusions réalisées en ne considérant que des ordonnancements aléatoires, à savoir que l'espérance et l'écart-type de la durée totale sont corrélés de façon non-négligeable. Bien que l'échelle logarithmique permette de mieux visualiser la répartition des faibles valeurs objectives, elle peut paraître trompeuse sur ce point. En revanche, on distingue clairement que les frontières NO et SE restreignent davantage l'espace que la frontière SO. Par exemple, il n'existe sans doute aucun ordonnancement avec une espérance élevée et un faible écart-type.

3.7.3 Qualité moyenne

Nous souhaitons désormais évaluer la qualité des ordonnancements générés avec les méthodes présentées dans la section 3.6.2 en terme d'efficacité et de robustesse. La figure 3.7.2 montre un exemple représentatif des valeurs objectives des solutions produites par chaque méthode. Les barres d'erreurs dénotent les intervalles de confiance des mesures faites sur chaque ordonnancement. Les relations de dominance prennent en compte ces intervalles. Ainsi, une solution est dominée pour un critère si sa valeur dans le meilleur des cas est supérieure à la valeur dans le pire des cas d'une autre solution. C'est pourquoi les lignes ont parfois une pente positive.

Pour quantifier la qualité d'un ensemble de solutions non-dominées, nous utilisons deux indicateurs décrits dans la section 2.2.3 du chapitre 2. Nous rappelons brièvement l'ϵ-indicateur binaire introduit par Zitzler et al. [165]. Soient O et O' deux ensembles de solutions non-dominées, alors la valeur $I_\epsilon(O, O')$ correspond à un rapport entre une solution de O et une solution de O' pour l'un des critères. Plus précisément, $I_\epsilon(O, O')$ est le facteur ϵ minimal tel que chaque solution de O' dont les valeurs objectives sont multipliées par ϵ est dominée par au moins une solution de O. Cet indicateur peut être vu comme une distance relative entre deux ensembles de solutions non-dominées. Si nous avons $I_\epsilon(O, O') \leq 1$ et $I_\epsilon(O', O) > 1$, alors l'ensemble O est meilleur que O'. Sinon, les ensembles sont incomparables.

Le tableau 3.5 compare les ensembles de solutions non-dominées représentées sur la figure 3.7.2 en fournissant les valeurs de l'ϵ-indicateur binaire entre chaque paire de méthodes. Par exemple, nous avons $I_\epsilon(\text{Métaheuristique}, \text{Rob-HEFT}) \leq 1$ et $I_\epsilon(\text{Rob-HEFT}, \text{Métaheuristique}) > 1$, ce qui signifie que la métaheuristique est meilleure que l'heuristique Rob-HEFT.

$O \backslash O'$	HEFT	Rob-HEFT	Rob-HEFT-MC	Métaheuristique
HEFT	1	1,147	1,187	1,173
Rob-HEFT	0,998	1	1,035	1,024
Rob-HEFT-MC	0,994	0,999	1	1,019
Métaheuristique	0,986	0,991	1,012	1

TABLE 3.5 – Valeurs de l'ϵ-indicateur binaire $I_\epsilon(O, O')$ pour chaque paire d'ensembles

Le fait que $I_\epsilon(\text{Métaheuristique}, \text{Rob-HEFT-MC}) > I_\epsilon(\text{Rob-HEFT-MC}, \text{Métaheuristique})$ ne signifie pas que l'heuristique Rob-HEFT-MC est meilleure que la métaheuristique. L'ϵ-indicateur est un rapport entre deux espérances ou entre deux écart-types. Comme cette dernière mesure subit une variation relative plus importante que celle subite par l'espérance, l'existence d'un ordonnancement robuste aura plus d'impact sur l'ϵ-indicateur que celle d'un ordonnancement efficace. Les valeurs quantitatives de cet indicateur ne sont donc pas systématiquement interprétables.

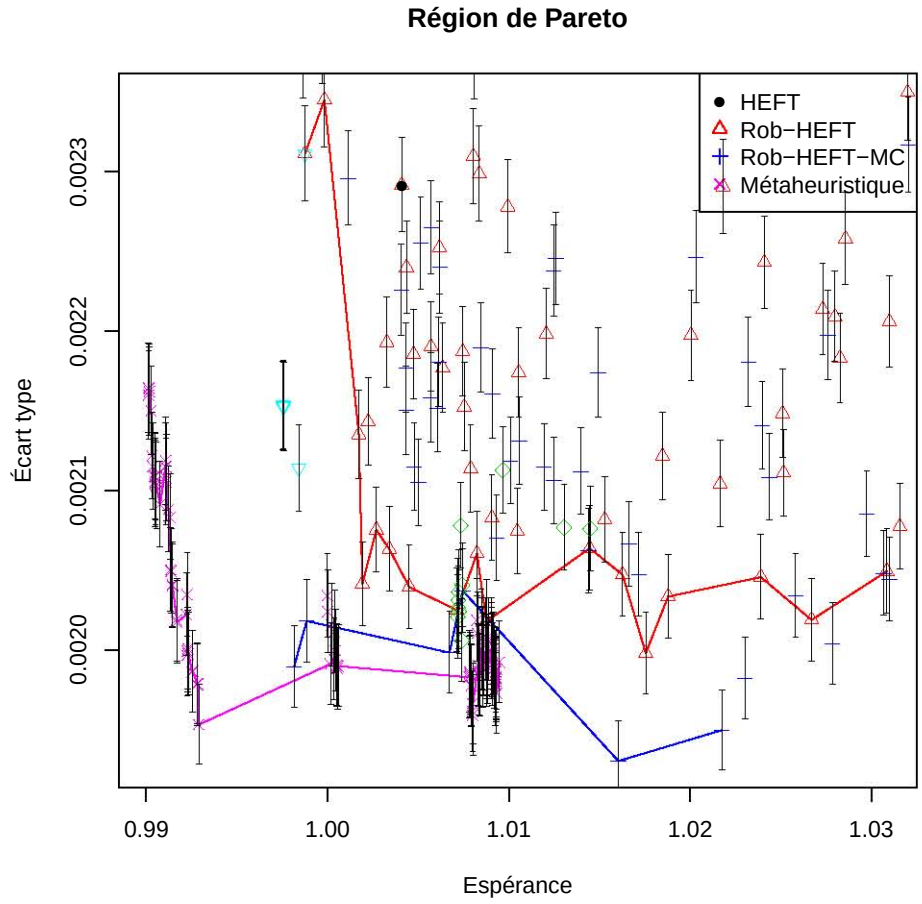


FIGURE 3.7.2 – Espérances et écart-types de la durée totale d'ordonnancements proches du front de Pareto pour un graphe de 1000 tâches de type layrpred. Chaque ligne connecte les valeurs objectives des solutions générées par la même méthode.

Cependant, comme l'ϵ-indicateur est un rapport, donc une mesure relative, nous pouvons agréger l'ensemble des valeurs obtenues pour l'ensemble des scénarios. La figure 3.7.3 représente les indicateurs I_ϵ sous forme de boîtes à moustaches (les ensembles O sont sur les lignes et les ensembles O' sont sur les colonnes). Une boîte à moustaches consiste en un résumé en cinq nombres d'un ensemble de valeurs : la médiane est le trait vertical gras, la boîte s'étend du premier quartile jusqu'au troisième quartile, et la longueur des moustaches vaut 1,5 fois l'intervalle interquartile. Enfin, les valeurs hors normes qui se situent en dehors des moustaches sont représentées par un point. Par exemple, sur la ligne de l'heuristique Rob-HEFT et la colonne de la métaheuristique, ces cinq valeurs sont $1,00$, $1,01$, $1,04$, $1,12$ et $1,28$. L'heuristique HEFT est omise car elle est systématiquement moins performante que les autres méthodes.

Ces résultats montrent que la métaheuristique et les heuristiques Rob-HEFT et ROB-HEFT-MC sont généralement incomparables. Les valeurs de l'ϵ-indicateur proches de 1 entre les heuristiques Rob-HEFT et Rob-HEFT-MC indiquent que les deux ensembles de solutions sont souvent très proches.

Comme il est difficile de conclure sur la qualité des méthodes en utilisant l'ϵ-indicateur, nous utilisons l'indicateur de couverture [163] que nous notons I_C. La valeur $I_C(O, O')$ correspond

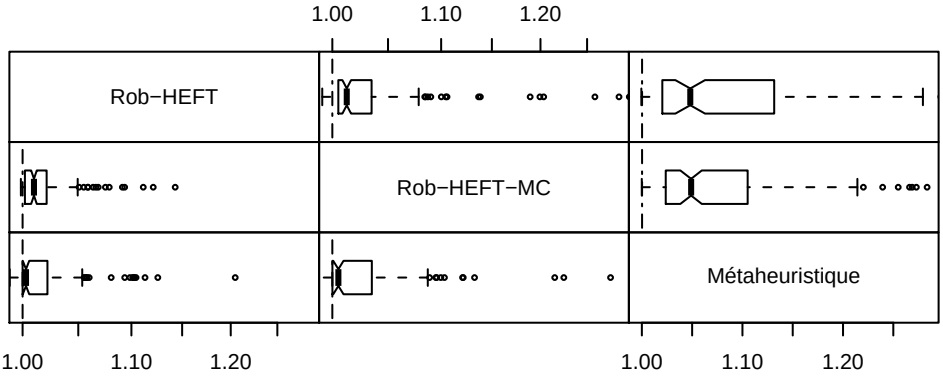


FIGURE 3.7.3 – Boîtes à moustaches des valeurs de l'ϵ-indicateur sur les 150 scénarios

à la proportion des solutions de l'ensemble O' qui sont dominées par au moins une solution de l'ensemble O. Pour une valeur de cet indicateur proche de 1, l'ensemble O est meilleur que l'ensemble O'.

La figure 3.7.4 montre les boîtes à moustaches pour l'indicateur I_C obtenues de la même manière que pour la figure 3.7.3. La métaheuristique se comporte mieux que les autres méthodes relativement à cet indicateur. Dans la moitié des cas, au moins 68% des solutions trouvées par Rob-HEFT-MC sont dominées par au moins une solution trouvée par la métaheuristique. Par ailleurs, dans 75% des cas, moins de 16% des solutions trouvées pas la métaheuristique sont dominées par au moins une solution trouvée par Rob-HEFT-MC. Même avec cet indicateur, les différences générales entre Rob-HEFT et Rob-HEFT-MC ne sont pas significatives.

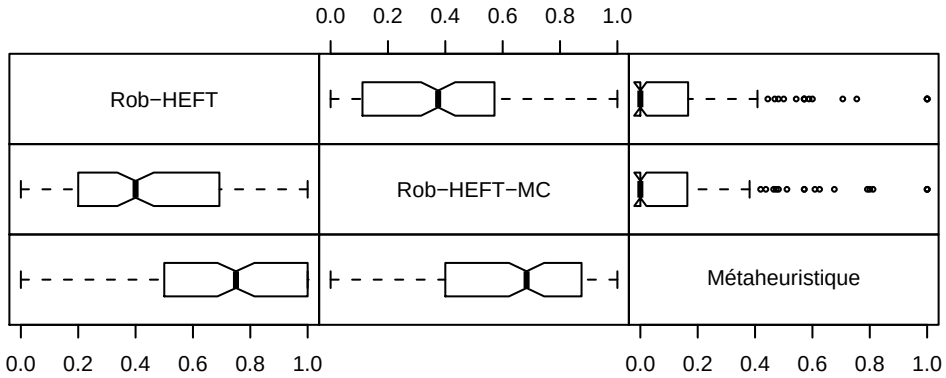


FIGURE 3.7.4 – Boîtes à moustaches des valeurs de l'indicateur de couverture sur les 150 scénarios

3.7.4 Temps de calcul

Nous mesurons le temps nécessaire pour que chaque méthode se termine. Les temps sont des moyennes sur six exécutions sur des graphes de type layrpred. Le tableau 3.6 donne les temps moyens obtenus en faisant varier la taille des graphes. Ces temps concernent la construction des ordonnancements et n'incluent pas l'évaluation de la distribution de la durée totale.

Nombre de tâches	10	100	1000
HEFT	0.2”	2”	23”
Rob-HEFT	0.3”	12”	4’44”
Rob-HEFT-MC	30”	30’37”	6h00
Métaheuristique	17’34”	43’57”	7h39

TABLE 3.6 – Temps d’exécution de chaque méthode

3.8 Conclusion

L’ordonnancement efficace et robuste de tâches à durée aléatoire sur plateforme hétérogène lève des problèmes difficiles (l’un est NP-Complet tandis que l’autre est #P’-Complet). Nous avons présenté dans ce chapitre une approche empirique pour valider nos méthodes.

Face à l’absence de consensus sur une mesure de robustesse unique, nous avons exploré les liens entre un ensemble de mesures représentatives. Ce comparatif empirique est réalisé en utilisant différents types de graphes. Plusieurs mesures se sont révélées équivalentes, ce que nous avons expliqué par l’intermédiaire du théorème central de la limite. L’écart-type s’est distingué en tant que mesure facile à estimer, intuitive et représentative. Enfin, nous avons invalidé la marge comme mesure de robustesse.

Bien que les mesures de performance et de robustesse soient corrélés, le problème reste bicritère car il existe quasiment toujours un ensemble de solutions non-dominées de taille strictement supérieure à 1. Nous avons étudié le comportement de différentes méthodes (heuristique et métaheuristique) dont le fonctionnement général est décrit dans le chapitre 2. Nous pouvons conclure que la métaheuristique est plus performante que les deux heuristiques qui sont elles-mêmes équivalentes. Nous avons donc principalement deux méthodes possédant chacune un compromis en terme de qualité et de temps unique.

Ces travaux peuvent se poursuivre en étudiant l’impact de décisions d’ordonnancement dynamique sur des ordonnancements produits statiquement. Nous reviendrons sur ce point dans la conclusion générale.

Chapitre 4

Évaluation de la fiabilité d'ordonnancements statiques

Sommaire

4.1 Introduction

Le contexte général de l'étude, à savoir la fiabilité dans le cadre du calcul parallèle, est d'abord présenté, suivi d'une description introductive de la terminologie utilisée et du type d'incertitude considéré. Cette section finit sur le plan de ce chapitre.

4.1.1 Description du contexte

L'ordonnancement de tâches sur systèmes distribués en présence de pannes fait l'objet de nombreuses études depuis les travaux majeurs dans ce domaine [19, 133, 95]. Lorsque des pannes affectent isolément les machines, il est naturel de répliquer les tâches sur des processeurs distincts. Ainsi, il suffit qu'une seule copie d'une tâche donnée réussisse son exécution pour que la tâche en question soit correctement exécutée. De cette façon, la probabilité que l'ordonnancement réussisse augmente.

Cela nous amène à considérer ce concept de fiabilité dans le cadre d'un ordonnancement qui spécifie la façon dont les tâches sont exécutées sur les processeurs. Chaque machine est susceptible de tomber en panne pendant l'exécution d'une tâche, auquel cas elle ne réussit pas à calculer le résultat de la tâche. La fiabilité d'un système se définit par sa capacité à conserver son fonctionnement pendant une période donnée. Dans le cas d'un ordonnancement, il s'agit de son aptitude à réussir malgré les pannes.

Nous limitons notre étude aux ordonnancements statiques. Ces ordonnancements définissent les dates de début et de fin d'exécution des copies de chaque tâche ainsi que les processeurs sur lesquels elles sont assignées. Comme les pannes surviennent durant l'exécution, une stratégie d'ordonnancement dynamique est pertinente. Toutefois, elle interdit de mettre en place des stratégies d'ordonnancements globales et se limite donc aux stratégies gloutonnes.

4.1.2 Typologie

Le problème que nous posons consiste à optimiser la durée totale d'un ordonnancement d'un graphe de tâches sur une plateforme de calcul hétérogène. Ce problème se note $R|prec|C_{\max}$ suivant la notation $\alpha|\beta|\gamma$ [76].

Nous raffinons ce problème pour tenir compte de l'incertitude se portant sur le succès du calcul d'une tâche. Cette incertitude est principalement d'origine matérielle et se manifeste par les pannes que subissent les processeurs. Cependant, il est envisageable que le succès d'un calcul soit tributaire d'événements humains ou logiciels. Dans tous les cas, l'échec d'un calcul est causé par une panne dont l'origine ne modifie par le formalisme du problème. Nous concluons cette discussion sur l'incertitude en analysant sa nature. Nous estimons que les mécanismes à l'œuvre lors d'une panne, surtout si elle est d'origine matérielle, sont de nature purement aléatoire et que nous sommes confrontés à un véritable phénomène stochastique.

Les pannes sont soit *franches*, les processeurs les subissant restant alors indisponibles jusqu'à la fin de l'ordonnancement, soit *transitoires*, les processeurs redevenant disponibles dès que la tâche en cours est terminée (et échouée).

Nous pouvons également choisir entre deux types d'ordonnancements. Aucune contrainte n'est soumise aux ordonnancements *généraux*. Cependant, dans un ordonnancement *strict*, aucune tâche ne peut commencer tant que toutes les copies de tous ses prédécesseurs n'ont pas fini leurs exécutions.

4.1.3 Description de l'étude

Le problème levé dans cette introduction est multicritère. Minimiser la durée totale et maximiser la fiabilité d'un ordonnancement en sont les objectifs. Tandis que l'évaluation de la durée totale ne pose pas de difficulté, il n'en est pas de même pour le calcul de la fiabilité à cause de la réplication des tâches sur plusieurs processeurs.

Nous nous focalisons donc sur l'évaluation de la fiabilité dans ce chapitre. Dans la section 4.4, nous montrons que le problème en toute généralité est difficile. Il s'agit d'un problème #P'-Complet en l'occurrence. Nous renvoyons le lecteur à la section 1.4 du chapitre 1 pour une description de cette classe de complexité. Pour une classe restreinte d'ordonnancements (les ordonnancements stricts), nous proposons une méthode approchée dont la complexité est paramétrable (section 4.5). Enfin, nous considérons une catégorie de graphes de tâches spécifiques dans la section 4.6 et proposons un algorithme d'évaluation à coût polynomial pour certains ordonnancements caractéristiques.

4.2 État de l'art

Bien que l'évaluation de la fiabilité d'ordonnancements dans le cas le plus général soit reconnu comme un problème difficile [73], ce sujet n'a pas reçu une attention importante dans la littérature. Distinguons néanmoins les cas avec et sans réplication. Lorsque les tâches ne sont pas répliquées, alors l'évaluation de la fiabilité est triviale.

Nous présentons d'abord quelques travaux qui se concentrent sur l'ordonnancement de graphes de tâches avec pour objectif de maximiser la fiabilité et de minimiser la durée totale. La spécificité des modèles proposés permet une évaluation de la fiabilité facile. Enfin, nous décrirons une méthode pour représenter la fiabilité d'un ordonnancement.

4.2.1 Ordonnancement bicritère

Ordonnancer des graphes de tâches avec l'objectif de maximiser la fiabilité et de minimiser la durée totale d'exécution, sans réplication des tâches, a été étudié dans [55, 88]. Le cas avec réplication est étudié dans [74]. Toutefois, les auteurs se concentrent sur des ordonnancements stricts avec pannes transitoires et peuvent donc utiliser des algorithmes polynomiaux pour évaluer la fiabilité. Ces méthodes sont décrites avec les notations adéquates dans la section 4.5.1.

4.2.2 Diagramme de fiabilité

Évaluer la fiabilité d'un système est souvent réalisé par l'intermédiaire de diagrammes de fiabilité (RBD pour Reliability Block Diagram) [33]. De nombreux travaux comme [74] supposent que l'évaluation de RBD a un coût exponentiel. À notre connaissance, il n'en existe aucune preuve formelle. Nous pouvons aisément dériver nos résultats pour prouver que l'évaluation de RBD est aussi #P'-Complet. Cependant, les travaux de Provan et Ball [120] permettent sans difficulté majeure d'aboutir à la même conclusion. Remarquons enfin que certains RBDs possèdent une structure particulière et qu'il existe un algorithme polynomial pour les résoudre (e.g., les ordonnancements stricts avec pannes transitoires).

4.3 Modèles et définitions

Cette section détaille d'abord le modèle de l'application et celui de la plateforme. Le modèle de pannes est ensuite décrit. Nous introduisons ensuite le mécanisme de réplication au sein de

Symbole	Définition
$T = \{t_i : i \in [1..n]\}$	Ensemble des tâches
n	Nombre de tâches ($n = \|T\|$)
$G = (T, E)$	Graphe orienté acyclique contenant les tâches et les contraintes de dépendance
$\text{Pred}(t_i)$	Ensemble des prédécesseurs de la tâche t_i ($\text{Pred}(t_i) \subset T$)
$P = \{p_j : j \in [1..m]\}$	Ensemble des processeurs
m	Nombre de processeurs ($m = \|P\|$)
$\pi : T \mapsto 2^{P \times \mathbb{N} \times [0;1]}$	Ordonnancement définissant les processeurs, les dates d'exécution et les probabilités de faute associées à chaque tâche
T_j	Ensemble des tâches t_i assignées au processeur p_j ($T_j \subseteq T$)
P_i	Ensemble des processeurs p_j auxquels est assignée la tâche t_i ($P_i \subseteq P$)
t_i^j	Copie de la tâche t_i assignée au processeur p_j
S_i^j	Date du début d'exécution de la copie t_i^j (indéfinie si t_i^j n'existe pas)
w_i^j	Durée d'exécution de la copie t_i^j
C_i^j	Date de fin d'exécution de la copie t_i^j (indéfinie si t_i^j n'existe pas)
$C_{\max}(\pi) = \max_{p_j \in P_n} C_n^j$	Durée totale de l'ordonnancement π (maximum des dates de fin C_n^j qui sont définies)
$\text{fiab}(\pi)$	Fiabilité de l'ordonnancement π

TABLE 4.1 – Notations du chapitre

l'ordonnancement et fournissons les formules exprimant la fiabilité de tels ordonnancements. Nous concluons sur un exemple qui montre la nature combinatoire du calcul de la fiabilité. Les notations utilisées dans ce chapitre sont résumées dans le tableau 4.1.

4.3.1 Application et plateforme

Les modèles d'application et de plateforme sont tirés de la littérature [35]. L'application est représentée par un graphe orienté acyclique (aussi appelé graphe de tâches) $G = (T, E)$, où T est l'ensemble des tâches à exécuter et E est l'ensemble des arcs de dépendance entre ces tâches. Soit $n = |T|$ le nombre de tâches. Nous numérotons les tâches $t_i \in T$, $1 \leq i \leq n$ selon un ordre topologique arbitraire (ce qui signifie que si $(t_i, t_j) \in E$, alors $i < j$). Par commodité, nous supposons l'existence d'une tâche source unique t_1 et d'une tâche puits unique t_n. La plateforme visée consiste en un ensemble P de m processeurs hétérogènes p_j, $1 \leq j \leq m$. L'exécution de la tâche t_i sur le processeur p_j nécessite w_i^j unités de temps. Sans perte de généralité, nous postulons que tous les temps d'exécution w_i^j sont entiers et que chaque date considérée est entière (nous pouvons au besoin changer d'échelle afin que tous les rationnels deviennent entiers).

4.3.2 Modèle de pannes

Les processeurs sont soumis à des pannes lors de l'exécution des tâches qui leur sont assignées. Il existe deux catégories de pannes qui peuvent se produire lorsque la tâche t est exécutée sur le

processeur p :

Transitoire Lors d'une panne transitoire du processeur p, la tâche t échoue. En revanche, le processeur p sera disponible pour exécuter d'autres tâches (les suivantes qui lui sont assignées par l'ordonnanceur, s'il y en a). En d'autres termes, p pourra contribuer au reste de l'exécution à l'issue d'une panne transitoire.

Franche Une panne franche n'est pas récupérable. Dans ce cas, le processeur p restera indisponible jusqu'à la fin de l'exécution de l'application. Toutes les tâches suivantes qui lui sont assignées ne seront alors pas exécutées.

Chacune des catégories regroupe des scénarios bien identifiés. Les pannes transitoires correspondent à des erreurs arithmétiques ou sont d'origines logicielles. Il peut aussi s'agir de défaillances matérielles récupérables (coupure de courant par exemple [132, 160]). Les pannes franches correspondent à des défaillances matérielles plus graves (surchauffe de l'alimentation, destruction physique d'un composant quelconque) ou à la récupération d'une machine par son propriétaire dans le cas d'un épisode de vol de cycle [18, 26, 124].

Nous considérons différentes distributions de probabilité pour modéliser les pannes. Selon une distribution générale, le processeur p_j tombe en panne à la date t (i.e., durant l'intervalle de temps $[t; t+1[$) avec une probabilité $d_j(t)$, où $0 \leq d_j(t) \leq 1$ est un rationnel arbitraire. La probabilité que le processeur p_j tombe en panne entre l'instant t et l'instant t' est donc $1 - \prod_{k=t}^{k=t'}(1 - d_j(k))$. Nous n'imposons pas que $1 - \prod_{t=0}^{+\infty}(1 - d_j(t)) = 1$, bien que ce soit souvent le cas pour les pannes franches car tout processeur est condamné à tomber en panne. Pour les pannes transitoires, les probabilités de panne sont souvent supposées dépendre de la durée de la tâche à exécuter. Nos résultats s'appliquent au cas le plus général où les probabilités de panne sont des rationnels arbitraires.

Sans perte de généralité, nous postulons que les pannes surviennent uniquement lors de l'exécution des tâches, et non pas lors des périodes d'inactivité des processeurs. Ce postulat n'a d'incidence qu'avec les pannes de type franche. Dans ces cas, la probabilité de panne durant une période d'inactivité sur un processeur peut être rajoutée à la probabilité de panne de la prochaine tâche à être exécutée sur le processeur en question sans que cela modifie la fiabilité de l'ordonnancement. Finalement, les pannes (transitoires ou franches) sont supposées être indépendantes, quelles que soient les lois de probabilité qu'elles suivent.

4.3.3 Réplication des tâches

L'objectif est d'exécuter l'application sur la plateforme définie ci-dessus. L'ordonnancement assigne les tâches sur les processeurs. Sans réplication, chaque tâche est assignée à un seul processeur. Dans ce cas, l'ordonnancement spécifie les dates de début et de fin d'exécution de chaque tâche sur le processeur qui lui est assigné. Cependant, pour remédier au problème posé par les pannes, l'ordonnanceur peut décider de répliquer l'exécution des tâches sur différents processeurs : si une tâche échoue sur un processeur donné, il lui reste une chance d'être exécutée avec succès sur un autre processeur, permettant ainsi au reste de l'application d'être exécutée entièrement malgré la panne.

Un ordonnancement est donc défini comme une fonction qui associe chaque tâche à un sous-ensemble de processeurs, chacun d'eux exécutant une copie de la tâche. Pour chaque copie, nous enregistrons un triplet de valeurs : l'indice du processeur, la date de début et la probabilité de panne. Formellement, $\pi : T \mapsto 2^{P \times \mathbb{N} \times [0;1]}$ associe chaque tâche à un ensemble de tels triplets. Nous dénoterons P_i l'ensemble des processeurs qui calculent tous la tâche t_i, et T_j l'ensemble des tâches assignées au processeur p_j.

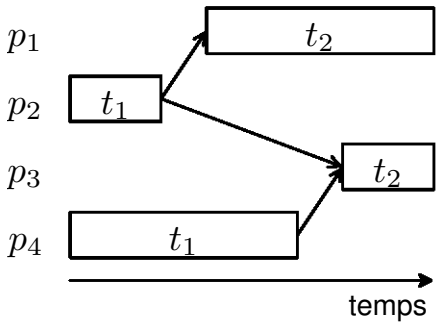


FIGURE 4.3.1 – Exécution possible d'une chaîne de deux tâches ($G = (\{t_1, t_2\}, \{(t_1, t_2)\})$) avec un ordonnancement général. Les flèches correspondent aux paires de copies valides.

Soit t_i^j la copie de la tâche t_i sur le processeur p_j (si cette copie existe). Sa date de début d'exécution est S_i^j et sa date de fin est $C_i^j = S_i^j + w_i^j$. Par convention, si la tâche t_i n'est pas assignée au processeur p_j, alors nous laissons C_i^j et S_i^j indéfinies. Pour simplifier les notations, nous supposerons que la tâche t_i est ordonnancée sur le processeur p_j chaque fois que nous utiliserons les notations C_i^j et S_i^j. D'autre part, ordonnancer la même tâche plusieurs fois sur le même processeur n'est pas autorisé, ce qui garantit l'unicité de chaque date C_i^j et S_i^j. Cette dernière contrainte ne limite cependant pas la généralité de nos résultats.

L'ordonnancement doit respecter les contraintes de dépendance. Sans réplication, il n'y a pas d'ambiguïté possible : s'il existe une contrainte de dépendance entre la tâche t_i et la tâche $t_{i'}$ (i.e., si $(t_i, t_{i'}) \in E$), alors l'ordonnancement doit spécifier que l'exécution de $t_{i'}$ ne commence pas avant que t_i ne soit finie : $C_i^j \leq S_{i'}^{j'}$ (en considérant que t_i soit assignée à p_j et que $t_{i'}$ soit assignée à $p_{j'}$). Quand plusieurs copies d'une même tâche sont exécutées, deux règles possibles peuvent régir les ordonnancements :

Ordonnancement strict L'exécution d'une tâche ne commence que si toutes les copies de chacun de ses prédécesseurs ont terminé leurs exécutions.

Ordonnancement général L'exécution d'une tâche peut commencer dès qu'au moins une copie de chacun de ses prédécesseurs a terminé son exécution.

Les ordonnancements stricts sont un cas particulier des ordonnancements généraux. Bien qu'ils soient moins génériques, ils sont plus simples à analyser dans le cas des pannes transitoires.

Il est important de préciser que ces définitions s'appliquent dans un contexte idéal, i.e., sans panne. Les dates de début et de fin d'exécution de toutes les copies sont déterministes et connues à l'avance grâce aux spécifications données par l'ordonnancement, avant que l'exécution ne démarre. Les pannes peuvent intervenir aléatoirement lors de l'exécution. La figure 4.3.1 présente une exécution possible avec un ordonnancement général : t_2^1, la copie de t_2 sur p_1, peut démarrer dès que t_1^2, la copie de t_1 sur p_2, est terminée. Il n'y a en effet pas besoin que t_2^1 attende la terminaison de l'autre copie t_1^4 de t_1 sur p_4. Cependant, si l'exécution de t_1^2 échoue, alors la copie t_2^1 devient inutile car sa contrainte de dépendance ne pourra être vérifiée ni grâce à t_1^2 (panne), ni grâce à t_1^4 (date de fin postérieure à la date de début de t_2^1).

Nous définissons désormais la notion de *paire de copies*. Pour chaque arc de dépendance $(t_i, t_{i'}) \in E$ et pour chaque paire de processeurs $(p_j, p_{j'}) \in P^2$, il existe deux cas : la date de fin C_i^j de la copie t_i^j de t_i est antérieure ou égale à la date de début $S_{i'}^{j'}$ de la copie $t_{i'}^{j'}$ de $t_{i'}$, auquel cas nous dirons que la paire de copies $(t_i^j, t_{i'}^{j'})$ est *valide* ; dans le cas contraire, nous dirons que $(t_i^j, t_{i'}^{j'})$ est *invalide*. Les flèches sur la figure 4.3.1 correspondent aux paires de copies valides.

Pour un ordonnancement strict, toutes les paires de copies doivent être valides pour chaque

arc de dépendance dans le graphe de tâches $G = (T, E)$. Pour un ordonnancement général, cette contrainte n'est pas nécessaire. Cependant, pour chaque chemin dans le graphe de tâches, il doit y avoir une liste de copies qui correspondent aux tâches sur ce chemin et telle que chaque copie forme une paire de copies valide avec sa voisine dans la liste : si ce n'est pas le cas, l'ordonnancement ne finira jamais son exécution, même en l'absence de panne. Intuitivement, les ordonnancements stricts sont plus *fiables* que les ordonnancements généraux : pour un ordonnancement strict, l'exécution d'une tâche débutera si et seulement si au moins une copie de chacun de ses prédécesseurs a été exécutée avec succès ; pour un ordonnancement général, l'ensemble des copies des prédécesseurs est restreint à celles dont les dates de fin ne dépassent pas la date de début de la tâche considérée. Cependant, la durée totale de l'ordonnancement sera vraisemblablement plus réduite avec un ordonnancement général qu'avec un ordonnancement strict. En effet, il y a moins de paires de copies à considérer, et donc moins de prédécesseurs à attendre.

4.3.4 Fiabilité

Similairement à Barlow et Proschan [20], nous considérons l'exécution d'un ordonnancement jusqu'à sa première panne. Comme on réplique les tâches, une panne lors de l'exécution d'une tâche n'implique pas forcément une panne générale de l'ordonnancement.

La fiabilité $\text{fiab}(\pi)$ d'un ordonnancement π se définit comme la probabilité qu'il réussisse, i.e., qu'il soit exécuté avec succès. Un ordonnancement strict réussit si et seulement si au moins une copie de chaque tâche termine correctement son exécution. Déterminer si un ordonnancement général réussit est plus compliqué : le graphe de tâche est traversé de manière à vérifier si chaque copie est exécutée correctement ou non. Plus précisément, une copie $t_{i'}^{j'}$ de la tâche $t_{i'}$ est exécutée correctement si et seulement si :

- $p_{j'}$ ne subit pas de panne pendant l'exécution de $t_{i'}^{j'}$, et
- pour chaque prédécesseur $t_i \in \text{Pred}(t_{i'})$ (s'il y en a), il existe au moins une paire de copies valide $(t_i^j, t_{i'}^{j'})$ telle que t_i^j est exécutée correctement.

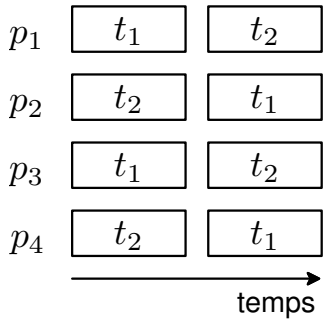
Finalement, un ordonnancement général réussit si au moins une copie de la tâche puit t_n est exécutée correctement. En effet, cela induit forcément que chaque tâche a été calculée au moins une fois.

Formellement, nous avons les définitions suivantes. Soit π un ordonnancement.

- Soit R_{ij} l'événement « le processeur p_j ne tombe pas en panne lors de l'exécution de t_i^j ». Avec des pannes transitoires, cela signifie que p_j n'échoue pas entre la date de début et celle de fin de t_i^j, alors qu'avec des pannes franches, cela signifie que p_j n'échoue pas entre le début de l'ordonnancement et la date de fin de t_i^j. Notons que cet événement est nécessaire mais pas suffisant pour que la copie t_i^j soit exécutée correctement : une paire de copies valide pour chaque prédécesseur de t_i doit pour cela être réussie. Soit $\Pr[R_{ij}] = 0$ si la tâche t_i n'est pas assignée au processeur p_j.
- Soit U_{ij} l'événement « la copie t_i^j de la tâche t_i est exécutée correctement ». Soit $\Pr[U_{ij}] = 0$ si la tâche t_i n'est pas assignée au processeur p_j.
- La fiabilité d'un ordonnancement général π se définit par $\text{fiab}(\pi) = \Pr\left[\bigcup_j U_{nj}\right]$, où $U_{1j} = R_{1j}$ et U_{ij} se définit récursivement pour $i > 1$ par

$$U_{ij} = \left(\bigcap_{t_{i'} \in \text{Pred}(t_i)} \bigcup_{p_{j'} \in P_{i'}, C_{i'}^{j'} \leq S_i^j} U_{i'j'} \right) \cap R_{ij} \tag{4.3.1}$$

(l'ensemble des copies des prédécesseurs est restreint aux paires de copies valides, i.e., à



	Fiabilité
p_1	P_{31}
p_1, p_2	$P_{11}P_{32} + P_{21}(P_{22} + P_{32}) + P_{31}$
p_1, p_2, p_3	$P_{11}(P_{12}P_{33} + P_{22}(P_{23} + P_{33}) + P_{32}) + P_{21}(P_{12}P_{33} + P_{22} + P_{32}) + P_{31}$
p_1, p_2, p_3, p_4	$P_{11}(P_{12}(P_{13}P_{34} + P_{23}(P_{24} + P_{34}) + P_{33}) + P_{22}(P_{13}P_{34} + P_{23} + P_{33}) + P_{32}) + P_{21}(P_{12}((P_{13} + P_{23})(P_{24} + P_{34}) + P_{33}) + P_{22} + P_{32}) + P_{31}$

FIGURE 4.3.2 – Ordonnancement de deux tâches indépendantes sur quatre processeurs

toutes les copies $t_{i'}^{j'}$ telles que $C_{i'}^{j'} \leq S_i^j$).

– La fiabilité fiab(π) d'un ordonnancement strict π se définit par

$$\text{fiab}(\pi) = \Pr\left[\bigcap_i \bigcup_j R_{ij}\right] \tag{4.3.2}$$

(les événements $U_{ij} = \left(\bigcap_{t_{i'} \in \text{Pred}(t_i)} \bigcup_{j'} R_{i'j'}\right) \cap R_{ij}$ ne sont pas nécessaires pour calculer la fiabilité d'un ordonnancement strict).

4.3.5 Communications

Précisons que les pannes survenant sur les liens de dépendance ou pendant les communications peuvent être incluses facilement lors du calcul de la fiabilité d'un ordonnancement : il faut pour cela remplacer chaque lien de dépendance $t_i^j \to t_{i'}^{j'}$ par deux arcs $(t_i^j, d_{ii'}^{jj'})$ et $(d_{ii'}^{jj'}, t_{i'}^{j'})$, où $d_{ii'}^{jj'}$ est une nouvelle tâche dont le temps d'exécution est le temps pris par la communication entre les deux copies concernées et dont la probabilité de panne (transitoire ou franche) peut être fixée arbitrairement. Dans le cas des pannes franches, chaque tâche $d_{ii'}^{jj'}$ doit être ordonnancée sur un processeur spécifique. La probabilité de panne de l'arc peut dépendre du coût de la communication ou/et du taux de panne du lien physique. Si $j = j'$, nous modélisons une panne survenant lors d'un transfert en mémoire ; sinon, nous modélisons une panne survenant sur des liens d'interconnexion physiques.

4.3.6 Exemple

Dans cette section, nous étudions un exemple simple pour montrer la difficulté liée au calcul de la fiabilité dans le cas des pannes franches, même avec des tâches indépendantes. Dans le cas des tâches indépendantes, tout ordonnancement est à la fois strict et général car il n'y a pas de contrainte de dépendance. La figure 4.3.2 illustre un ordonnancement avec deux tâches et quatre processeurs qui exécutent chacun les deux tâches (mais dans des ordres différents). Chaque tâche est donc répliquée quatre fois. Pour chaque processeur p_j : P_{1j} dénote la probabilité que p_j tombe en panne durant l'exécution de la première tâche qui lui est assignée ; P_{2j} est la probabilité que p_j tombe en panne durant l'exécution de sa seconde tâche ; et P_{3j} est la probabilité que p_j ne subisse pas de panne avant le terme des deux tâches. L'approche analytique directe pour évaluer la fiabilité de cet ordonnancement consiste à former chaque produit du type $P_{a1}P_{b2}P_{c3}P_{d4}$ avec $a, b, c, d \in \{1, 2, 3\}^4$. Chaque produit est la probabilité qu'un scénario d'exécution spécifique se réalise et tous ces scénarios sont distincts. Pour calculer la fiabilité de l'ordonnancement, nous

pouvons donc ajouter tous les termes correspondants à un scénario conduisant à une réussite. Par exemple, $P_{11}P_{22}P_{23}P_{14}$ est la probabilité que p_1 et p_2 tombent en panne alors qu'ils calculaient leurs copies de t_1 et que p_3 et p_4 tombent en panne en calculant leurs copies de t_2. Ce scénario conduit à une réussite car t_2 est calculé par p_2 et t_1 par p_3. Le tableau sur la figure 4.3.2 montre les formules obtenues avec cette approche. Chaque formule définit la fiabilité de l'ordonnancement lorsque seul le sous-ensemble de processeurs défini par la première colonne est pris en compte. Nous remarquons que le nombre de termes (même factorisés) augmente exponentiellement avec le nombre de processeurs.

4.4 Évaluation de la fiabilité d'un ordonnancement général

Dans cette section, nous montrons que calculer la fiabilité d'un ordonnancement général est un problème #P'-Complet (voir la section 1.4 du chapitre 1). Ce résultat ne dépend pas du type de pannes considéré et est donc applicable dans les deux cas (pannes transitoires et franches).

Théorème 4.1. *Calculer la fiabilité d'un ordonnancement général est #P'-Complet.*

Nous précisons d'abord le problème d'évaluation.

Définition 4.2. *Le problème dénoté* Fiabilité-Général *consiste à calculer la fiabilité fiab*(π) *d'un ordonnancement général* π *telle que défini dans la section 4.3, avec des probabilités* $\Pr[R_{ij}]$ *rationnelles arbitraires.*

Pour prouver que ce problème est #P'-Complet, nous montrons d'abord l'appartenance de ce problème à la classe #P' dans la section 4.4.1. La complétude est établie au travers d'une réduction partant du problème Connexion qui est prouvé #P-Complet dans la section 1.4.2 du chapitre 1. La réduction de Connexion vers Fiabilité-Général est expliquée dans la section 4.4.2. Finalement, la complétude est prouvée dans la section 4.4.3.

4.4.1 Appartenance à la classe #P'

Pour prouver que ce problème appartient à #P', nous caractérisons le problème de décision NP sous-jacent et la transformation nécessaire pour générer la probabilité de sortie qui est dans notre cas un nombre rationnel.

Proposition 4.3. Fiabilité-Général *est dans #P'.*

Démonstration. Les probabilités de succès de chaque processeur p_j lors de l'exécution de la tâche t_i sont toutes encodées suivant une fraction $\frac{n_{ij}}{d_{ij}}$. Un vecteur $x_i = (x_{ij})_{1\leq j\leq m}$ spécifie le succès de chaque processeur p_j lorsqu'il exécute la tâche t_i. Si $1 \leq x_{ij} \leq n_{ij}$, alors p_j ne tombe pas en panne en calculant t_i. Si $n_{ij} < x_{ij} \leq d_{ij}$, alors p_j tombe en panne en calculant t_i. Sinon, t_i n'est pas assignée à p_j et $x_{ij} = 0$ ($d_{ij} = 1$ et n_{ij} n'est pas défini dans ce cas).

Le problème de décision NP est le suivant : étant donné un ordonnancement, existe t-il un vecteur x_i tel que l'ordonnancement réussisse son exécution ? Ce problème appartient à NP car le certificat est le vecteur $x = (x_i)_{1\leq i\leq n}$ qui est de taille $O(nm)$. Nous testons si un vecteur x encode une exécution réussie en construisant l'ordonnancement ne contenant que les tâches qui sont exécutées correctement. Si l'ordonnancement obtenu est valide, c'est-à-dire si toutes les contraintes de dépendance sont respectées et si toutes les tâches sont correctement exécutées (voir la section 4.3 pour plus de détails sur cette procédure de vérification), alors x encode une exécution réussie d'un ordonnancement.

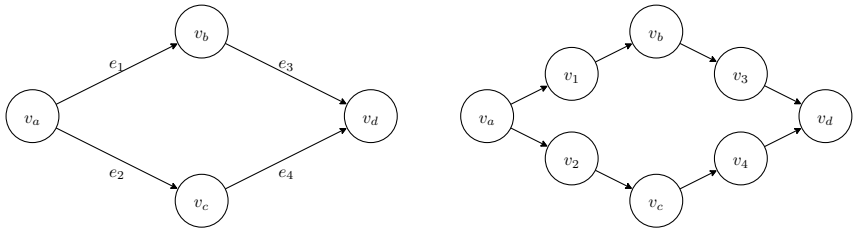


FIGURE 4.4.1 – Une instance du problème *connexion* et sa pré-transformation

Le problème #P sous-jacent à FIABILITÉ-GÉNÉRAL consiste à calculer combien de vecteurs distincts x aboutissent à des exécutions réussies. En d'autres mots, il existe $\prod_{i=1}^{n} \prod_{j=1}^{m} d_{ij}$ vecteurs distincts et chacun définit une exécution possible de l'ordonnancement. Comme toutes ces exécutions sont équiprobables, nous obtenons la fiabilité de l'ordonnancement en divisant le nombre d'exécutions favorables par le nombre total d'exécutions possibles. □

4.4.2 Réduction

Nous utilisons une réduction partant du *problème de connexion*, dénoté CONNEXION. Ce problème est défini et prouvé #P'-Complet dans la section 1.4.2. Pour rappel, il s'agit de calculer la probabilité que le sommet source et le sommet puits soient liés par un chemin d'arcs actifs dans un graphe orienté acyclique $G = (A, B)$ dont les arcs deviennent inactifs avec des probabilités rationnelles arbitraires. La probabilité que nous évaluons est appelée *probabilité de connexion*. S'il existe un chemin d'arcs actifs entre le sommet source et un sommet quelconque, alors ce dernier sommet est dit *accessible*.

4.4.2.1 Pré-transformation des instances

Pour faciliter la réduction, nous transformons légèrement les instances de CONNEXION et fournissons quelques notations formelles. Sans perte de généralité, nous supposons qu'il existe un unique sommet source et un unique sommet puits dans le graphe G. D'abord, nous transférons les probabilités sur les sommets. On obtient donc un ensemble de sommets à ajouter qui seront actifs avec une certaine probabilité. Ces sommets correspondront à des tâches dans la réduction décrite plus bas. Chaque arc (i, j) du somment i vers le sommet j est remplacé par un nouveau sommet k et par deux arcs : (i, k) de i vers k, et (k, j) de k vers j. La figure 4.4.1 illustre ce mécanisme : l'instance du problème de connexion initial possède quatre sommets et l'instance transformée en possède huit. La probabilité que l'arc (i, j) soit inactif est ensuite transférée au sommet introduit k. Tous les arcs sont toujours actifs. La probabilité que le sommet puits soit accessible dans le nouveau graphe est donc identique dans les deux instances (celle de départ et celle après transformation). Le calcul de la probabilité de connexion dans ce problème est donc encore #P'-Complet.

Formellement, soit N_i l'événement « le sommet i est actif » et V_i l'événement « le sommet i est accessible ». La probabilité de connexion est alors $\Pr[V_n]$, où $V_1 = N_1$ et V_i se définit récursivement pour $i > 1$ par

$$V_i = \bigcup_{t_{i'} \in \mathrm{Pred}(t_i)} (V_{i'} \cap N_{i'}) \tag{4.4.1}$$

Nous considérons par la suite que les instances de CONNEXION ont subi cette transformation et utilisons directement les notations précédentes.

Algorithme 4.1 Réduction d'une instance du problème de connexion en une instance du problème du calcul de la fiabilité d'un ordonnancement général.

```
1: partitionner les sommets en niveaux par un parcours en largeur L = (L_1, L_2, ...)
2: temps = 0
3: pour chaque l ∈ L faire
4:    pour chaque t_i ∈ l faire
5:       si t_i ≠ t_1 alors                          {t_i n'est pas la tâche source}
6:          π(t_i) = {(p_prop, temps, 0)}
7:          temps = temps + 1
8:       sinon
9:          π(t_i) = ∅
10:      fin si
11:   fin pour
12:   pour chaque t_i ∈ l faire
13:      pour chaque t_i' ∈ Pred(t_i) faire
14:         si t_i' ∉ T_sat alors                    {t_i' n'est pas ordonnancée sur p_sat}
15:            π(t_i') = π(t_i') ∪ {(p_sat, temps, 0)}
16:            temps = temps + 1
17:         fin si
18:      fin pour
19:   fin pour
20:   pour chaque t_i ∈ l faire
21:      π(t_i) = π(t_i) ∪ {(p_i, temps, 1 − Pr[N_i])}
22:   fin pour
23:   temps = temps + 1
24: fin pour
```

4.4.2.2 Algorithme de réduction

L'algorithme 4.1 décrit la façon dont une instance de CONNEXION est réduite en une instance de FIABILITÉ-GÉNÉRAL. Une tâche est créée pour chaque sommet de l'instance de CONNEXION. Chaque tâche est ordonnancée sur un processeur dont la probabilité de succès est égale à la probabilité que le sommet correspondant dans l'instance de CONNEXION soit inactif. Nous montrerons que la probabilité de connexion de l'instance de CONNEXION est égale à la probabilité que l'ordonnancement créé par l'algorithme de réduction échoue. En fait, l'ordonnancement réussit (le sommet puits est inaccessible) si certaines tâches sont exécutées sur leurs processeurs spécifiques (certains sommets sont inactifs). Si l'ordonnancement est exécuté avec succès, alors il n'existe aucun chemin de sommets actifs.

L'algorithme de réduction commence par partitionner les sommets en niveaux par un parcours en largeur. Tous les sommets de profondeur i sont mis dans le $i^{\text{ème}}$ niveau. Rappelons que la structure du graphe est particulière car il résulte d'une pré-transformation. Aussi, tous les successeurs des prédécesseurs des sommets du niveau l sont aussi dans l. Une tâche est ensuite créée pour chaque sommet de l'instance de base et est ordonnancée trois fois (sauf pour les sommets source et puits qui n'ont que deux copies). Le temps d'exécution de chaque tâche sur chaque processeur est égal à 1.

Les deux processeurs p_{prop} et p_{sat} jouent des rôles spéciaux et exécutent toutes les tâches (à l'exception des tâches source et puits). Ces processeurs ne tombent jamais en panne. Chaque tâche est aussi ordonnancée sur un processeur dont l'indice est le même que celui de la tâche en question

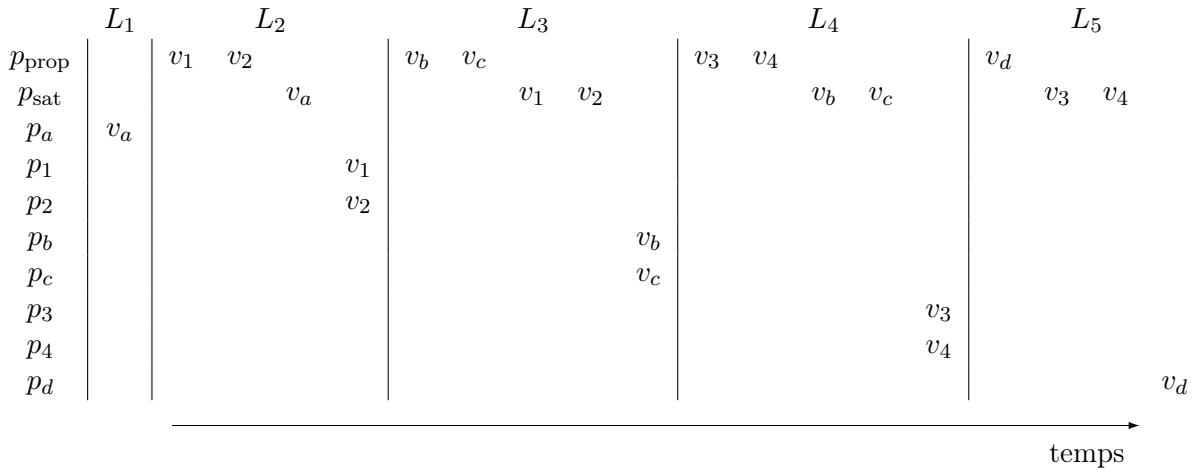


FIGURE 4.4.2 – Ordonnancement construit par l'algorithme de réduction pour l'instance de la figure 4.4.1.

(i.e., chaque tâche t_i est assignée au processeur p_i). Intuitivement, le rôle du processeur p_{prop} est de « propager » le succès d'une tâche à ses successeurs. Cette notion de « propagation » se comprend mieux en considérant le problème de connexion : un sommet peut être valide sans pour autant être accessible, auquel cas l'inaccessibilité de ses ancêtres doit lui être « propagée ». Garder trace de ces inaccessibilités (équivalent à des succès dans l'ordonnancement) est la clé de la réduction. Initialement, les contraintes de dépendance des tâches ordonnancées sur p_{prop} ne peuvent être satisfaites par les tâches assignées aux processeurs p_{prop} ou p_{sat}. Mais si la copie d'une tâche t réussit sur son processeur spécifique, les contraintes de dépendance entre t et ses successeurs ordonnancés sur p_{prop} sont satisfaites. Si toutes les contraintes de dépendance de ces successeurs sont satisfaites, alors ces successeurs sont exécutés avec succès sur p_{prop}. Le succès d'une tâche est donc « propagé » à ses successeurs qui peuvent alors être exécutés correctement même si leurs copies assignées à leurs processeurs spécifiques échouent. L'idée réside dans le fait que chaque tâche ordonnancée sur son processeur spécifique finit avant que ses successeurs ne débutent sur p_{prop}.

De plus, les contraintes de dépendance de chaque tâche ordonnancée sur son processeur spécifique sont satisfaites grâce à p_{sat}. En effet, nous voulons que les tâches assignées à leurs processeurs spécifiques soient exécutées correctement indépendamment de leurs contraintes de dépendance. Ainsi, tous les ancêtres de la tâche t_i doivent avoir réussi avant la date S_i^j. Le processeur p_{sat} joue donc ce rôle en calculant avec succès toutes les tâches (à l'exception de la tâche puits) dans un ordre topologique. Remarquons que nous utilisons deux processeurs identiques et d'une fiabilité parfaite, p_{prop} et p_{sat}, car le modèle interdit d'ordonnancer deux fois une tâche donnée sur le même processeur.

Dans l'exemple de la figure 4.4.2, le parcours en largeur génère cinq niveaux : $\{v_a\}$, $\{v_1, v_2\}$, $\{v_b, v_c\}$, $\{v_3, v_4\}$ et $\{v_d\}$. Tous les successeurs des prédécesseurs d'un niveau sont dans le même niveau. Pour le niveau L_4, les trois étapes de la boucle principale de l'algorithme de réduction consiste à :

1. ordonnancer les tâches du niveau en question, v_3 et v_4, sur p_{prop} ;
2. ordonnancer les prédécesseurs de ces tâches, v_b et v_c, sur p_{sat} ; et,
3. ordonnancer les tâches dans L_4 sur leurs processeurs spécifiques, p_3 et p_4.

Si v_1 est calculée correctement sur p_1, alors v_3 réussit sur p_{prop} ce qui illustre le principe de

Symbole	Définition
R_{ij}	Événement « le processeur p_j n'échoue pas avant la fin de l'exécution de la copie t_i^j »
U_{ij}	Événement « la copie t_i^j est exécutée correctement »
N_i	Événement « le sommet i est actif » (problème de connexion)
V_i	Événement « le sommet i est accessible » (problème de connexion)
p_{prop}	Processeur spécial qui « propage » le succès d'une tâche à ses successeurs
p_{sat}	Processeur spécial qui satisfait les contraintes de dépendance de chaque tâche ordonnancée sur son processeur spécifique

TABLE 4.2 – Liste des notations utilisées par les lemmes 4.4 et 4.5

« propagation » des succès des tâches (qui correspondent à des inaccessibilités dans l'instance de CONNEXION). Dans le cas contraire, v_3 peut encore être calculée sur p_3. Dans les deux cas, l'ordonnancement est exécuté avec succès si v_d réussit, c'est-à-dire, si les tâches v_3 et v_4 sont toutes les deux exécutées correctement. Dans l'instance de CONNEXION, cela équivaut à énoncer que v_d est inaccessible, si v_3 et v_4 sont inaccessibles.

4.4.3 Complétude du problème

Pour prouver la complétude de FIABILITÉ-GÉNÉRAL, nous proposons d'abord une simplification de l'équation 4.3.1 (lemme 4.4) qui définit récursivement la fiabilité d'un ordonnancement général. Cette simplification s'obtient en remarquant que le succès d'une tâche sur p_{prop} dépend du succès des copies de tous ses prédécesseurs, soit parce qu'ils ont été exécutés correctement sur leurs processeurs spécifiques, soit parce que ces copies sur p_{prop} réussissent. Cela signifie que le succès de la tâche (ce qui correspond à l'inaccessibilité du sommet correspondant) est « propagé » à ses successeurs. Nous introduisons ensuite la correspondance entre les événements de l'instance de CONNEXION et ceux de l'instance réduite (lemme 4.5) : nous montrons en particulier que toutes les contraintes de dépendance d'une tâche ordonnancée sur son processeur spécifique sont satisfaites par les copies assignées à p_{sat}.

Les notations utilisées dans les lemmes énoncés dans cette section sont résumées dans le tableau 4.2.

4.4.3.1 Simplification de la formulation de la fiabilité

Nous rappelons l'équation 4.3.1 que nous simplifions dans le cas des ordonnancements produits par l'algorithme de réduction :

$$U_{ij} = \left(\bigcap_{t_{i'} \in \text{Pred}(t_i)} \bigcup_{p_{j'} \in P_{i'}, C_{i'}^{j'} \leq S_i^j} U_{i'j'} \right) \cap R_{ij}$$

Lemme 4.4. *Soit un ordonnancement résultant d'une réduction d'une instance du problème de connexion grâce à l'algorithme 4.1. Le succès d'une tâche quelconque $t_i \in T$ sur le processeur p_{prop} se formule*

$$U_{iprop} = \bigcap_{t_{i'} \in Pred(t_i)} (U_{i'prop} \cup U_{i'i'}).$$

Démonstration. En utilisant l'équation 4.3.1 et en considérant le processeur p_{prop}, nous obtenons

$$U_{i\text{prop}} = \left(\bigcap_{t_{i'} \in \text{Pred}(t_i)} \bigcup_{p_{j'} \in P_{i'}, C_{i'}^{j'} \leq S_i^{\text{prop}}} U_{i'j'} \right) \cap R_{i\text{prop}}.$$

Par construction, les processeurs p_{prop} et p_{sat} ne tombent jamais en panne. Ainsi, $R_{i\text{prop}}$ se produit avec une probabilité 1 et nous éliminons donc ce terme. Nous simplifions davantage en développant l'union interne. Chaque prédécesseur $t_{i'}$ d'une tâche t_i est ordonnancé trois fois : sur le processeur p_{prop} (sauf pour la tâche source) ; sur le processeur p_{sat} ; et sur son processeur spécifique. Nous caractérisons désormais quelles copies $t_{i'}^j$ du prédécesseur $t_{i'}$ se terminent avant que t_i ne commence sur p_{prop}, i.e., quels sont les processeurs p_j tels que $C_{i'}^j \leq S_i^{\text{prop}}$.

Chaque tâche $t \in T$ (sauf la tâche source) du $k^{\text{ième}}$ niveau est ordonnancée sur p_{prop} et sur son processeur spécifique lors de la $k^{\text{ième}}$ itération. Ainsi, lorsqu'un successeur de t dans le $k'^{\text{ième}}$ niveau (avec $k' > k$) est ordonnancé sur p_{prop} lors de la $k'^{\text{ième}}$ itération, t a déjà été terminée sur p_{prop} et sur son processeur spécifique. Formellement, si $t_{i'}$ est un prédécesseur de t_i, alors $C_{i'}^{\text{prop}} \leq S_i^{\text{prop}}$ et $C_{i'}^{i'} \leq S_i^{\text{prop}}$.

Nous montrons désormais par l'absurde que chaque prédécesseur d'une tâche t finit son exécution sur p_{sat} après que t ne démarre sur p_{prop}, i.e., $C_{i'}^{\text{sat}} > S_i^{\text{prop}}$. Cela permet de développer l'union interne sans devoir considérer le succès des prédécesseurs ordonnancés sur p_{sat}.

Dans l'algorithme 4.1, nous considérons une tâche $t \in T$ de profondeur k. Si t n'est pas la tâche source, alors t est ordonnancée sur p_{prop} (ligne 6) à la $k^{\text{ième}}$ itération car le parcours en largeur insère t dans le $k^{\text{ième}}$ niveau. Par ailleurs, t est ordonnancée sur p_{sat} (ligne 15) après la $k^{\text{ième}}$ itération car tous les successeurs de t sont dans les niveaux suivants. Supposons maintenant que t finisse sur p_{sat} avant qu'au moins un de ses successeurs t' dans le $k'^{\text{ième}}$ niveau (avec $k' > k$) ne commence son exécution sur p_{prop}. Dans ce cas, t est ordonnancée sur p_{sat} avant que t' ne soit ordonnancé sur p_{prop} car les coûts et les incréments de temps sont tous unitaires. À la $k'^{\text{ième}}$ itération, t' est ordonnancé sur p_{prop} avant qu'une quelconque tâche ne soit ordonnancée sur p_{sat}. La tâche t a donc forcément un autre successeur dont la profondeur est plus faible que k' (t ne serait pas ordonnancée sur p_{sat} avant la $k'^{\text{ième}}$ itération dans le cas contraire). Ce qui implique que $k' > k+1$, i.e., il existe un niveau qui contient cet autre successeur entre le $k^{\text{ième}}$ et le $k'^{\text{ième}}$ niveau. Le successeur t' a donc un prédécesseur dans le $(k'-1)^{\text{ième}}$ niveau car la profondeur de t' est k'. Ce prédécesseur ne peut par être t car t est dans le $k^{\text{ième}}$ niveau et $k < k'-1$. Nous constatons donc que t possède deux successeurs parmi lesquels t' qui a lui-même aussi deux prédécesseurs. Il y a donc au final deux possibilités : soit t correspond à un sommet dans le graphe initial de CONNEXION (avant pré-transformation), ou bien t correspond à un arc transformé en sommet. Dans le premier cas, cela implique que t' correspond à un arc. Cependant, un sommet résultant d'un arc n'a qu'un seul prédécesseur, ce qui est antagoniste avec le fait que t' en a au moins deux. Dans le second cas, c'est la tâche t qui provient d'un arc. Mais dans ce cas, elle ne devrait avoir qu'un seul successeur et non pas deux comme c'est le cas. Donc, il n'existe aucune tâche finissant sur p_{sat} avant que l'un de ses successeurs ne démarre sur p_{prop}. □

4.4.3.2 Correspondance des événements

Nous supposons que le processeur spécifique de chaque tâche $t_i \in T$ tombe en panne (i.e., l'événement $\overline{R_{ii}}$ se produit) si et seulement si son sommet correspondant dans le problème de connexion est actif (i.e., l'événement N_i se produit). Cette hypothèse est vérifiée par construction : tous les événements N_i sont indépendants, tous les événements R_{ij} sont aussi indépendants, et

les probabilités sont identiques, i.e., $\Pr[R_{ii}] = 1 - \Pr[N_i]$ pour chaque tâche $t_i \in T$. Nous avons donc que $R_{ii} = \overline{N_i}$.

Lemme 4.5. *Soit un ordonnancement résultant d'une réduction d'une instance du problème de connexion grâce à l'algorithme 4.1. Chaque tâche $t_i \in T$ est exécutée correctement sur son processeur spécifique si et seulement si son sommet correspondant est inactif, i.e., $U_{ii} = \overline{N_i}$.*

Démonstration. Nous prouvons d'abord que tous les ancêtres de la tâche t_i sont ordonnancés sur le processeur p_{sat} dans un ordre topologique. Plus précisément, nous montrons par induction sur les niveaux que chaque tâche présente dans les k premiers niveaux commence son exécution sur son processeur spécifique une fois que tous ses ancêtres sont ordonnancés dans un ordre topologique sur p_{sat}.

L'initialisation s'effectue pour $k = 1$. La tâche source n'a pas d'ancêtre, l'hypothèse est donc vérifiée. Supposons désormais que l'hypothèse est vérifiée pour un k donné. Soit t une tâche du $(k+1)^{\text{ième}}$ niveau. À la $(k+1)^{\text{ième}}$ itération, t est ordonnancée sur son processeur spécifique (ligne 21) une fois que ses prédécesseurs sont ordonnancés sur p_{sat} s'ils ne l'étaient pas déjà (ligne 15). Ces prédécesseurs appartiennent aux k premiers niveaux. Ainsi, leurs ancêtres sont ordonnancés dans un ordre topologique sur p_{sat} lors des k premières itérations (par hypothèse d'induction). Comme les coûts et les incréments de temps sont tous unitaires, les tâches ordonnancées à la $(k+1)^{\text{ième}}$ itération commencent une fois que toutes les tâches précédemment ordonnancées sont terminées. Les ancêtres des prédécesseurs de t sont donc ordonnancés dans un ordre topologique sur p_{sat} et les prédécesseurs de t qui n'ont pas été ordonnancés sur p_{sat} le sont dans un ordre arbitraire lors de la $(k+1)^{\text{ième}}$ itération. Comme ces prédécesseurs de t appartiennent au même niveau, ils ont la même profondeur et ne nécessitent pas d'être ordonnancés dans un ordre spécial pour que leurs contraintes de dépendance soient satisfaites. Ainsi, t débute son exécution sur son processeur spécifique alors que tous ses ancêtres ont fini leurs exécutions sur p_{sat}.

Conséquemment, tous les ancêtres de la tâche t_i sont ordonnancés sur p_{sat} et finissent avant que t_i ne commence son exécution sur son processeur spécifique p_i. De plus, les ancêtres de t_i réussissent avec une probabilité 1 car les tâches assignées à p_{sat} n'échouent jamais (ligne 15). Ainsi, lorsque t_i débute son exécution sur p_i, toutes ses contraintes de dépendance sont satisfaites. Par ailleurs, il n'y a aucune autre tâche ordonnancée sur p_i. Le succès de la tâche t_i dépend donc uniquement de son exécution, i.e., $U_{ii} = R_{ii}$ et donc $U_{ii} = \overline{N_i}$. □

4.4.3.3 Preuve de la complétude

Les deux lemmes précédents facilitent la démonstration de la proposition qui suit.

Proposition 4.6. Fiabilité-Général *est #P'-Difficile.*

Démonstration. Nous souhaitons montrer que la probabilité qu'il existe un chemin de sommets actifs dans une instance du problème de connexion est égale à la probabilité que l'ordonnancement obtenu par l'algorithme de réduction échoue. Formellement, il s'agit de prouver que $V_n = \overline{\bigcup_j U_{nj}} = \overline{U_{n\text{prop}}}$ (rappelons que la tâche t_n échoue avec une probabilité 1 sur son processeur spécifique et que t_n n'est pas ordonnancée sur le processeur p_{sat}). Cela induira directement que la probabilité de connexion d'une instance Connexion est égale à la probabilité que l'ordonnancement créé par l'algorithme de réduction échoue.

La relation entre les définitions de V_i et de U_{ij} (données dans la section 4.3) est obtenu en utilisant la loi de Morgan $\overline{X \cup Y} = \overline{X} \cap \overline{Y}$ et grâce aux lemmes 4.4 et 4.5.

Sans perte de généralité, supposons que les tâches sont triées dans un ordre topologique. Nous procédons par induction et montrons que pour chaque tâche t_i, l'inaccessibilité du sommet i équivaut au succès de t_i sur le processeur p_{prop}, i.e., $\forall i, V_i = \overline{U_{i\text{prop}}}$.

Pour $i = 1$, la tâche source n'est pas assignée au processeur p_{prop} car elle ne possède pas de prédécesseur. Aussi, $U_{i\text{prop}}$ ne se produit jamais. Par contre, le sommet source est présent dans l'instance de CONNEXION et est toujours actif, ce qui implique que V_i se produit toujours. L'initialisation de l'induction est donc vérifiée, i.e., $V_1 = \overline{U_{1\text{prop}}}$.

Pour une tâche t_i, nous supposons que $V_k = \overline{U_{k\text{prop}}}$ est vrai pour $1 \leq k < i$. Nous montrons que dans ce cas, $V_i = \overline{U_{i\text{prop}}}$ est également vrai.

$$
\begin{aligned}
V_i &= \bigcup_{t_{i'} \in \text{Pred}(t_i)} V_{i'} \cap N_{i'} && \text{équation 4.4.1} \\
&= \bigcup_{t_{i'} \in \text{Pred}(t_i)} \overline{U_{i'\text{prop}}} \cap N_{i'} && \text{hypothèse d'induction} \\
&= \bigcup_{t_{i'} \in \text{Pred}(t_i)} \overline{U_{i'\text{prop}}} \cap \overline{U_{i'i'}} && \text{lemme 4.5} \\
&= \overline{\bigcap_{t_{i'} \in \text{Pred}(t_i)} U_{i'\text{prop}} \cup U_{i'i'}} && \text{loi de Morgan} \\
&= \overline{U_{i\text{prop}}} && \text{lemme 4.4}
\end{aligned}
$$

Dans la seconde ligne, nous avons utilisé le fait que $i' < i$ car les tâches sont parcourues dans un ordre topologique.

Nous avons montré que l'algorithme de réduction est correct. Pour mesurer sa complexité spatiale, nous comptons le nombre de processeurs utilisés, le nombre de copies ordonnancées et l'espace nécessaire pour stoker les probabilités. L'algorithme ordonnance chacune des $n = |A| + |B|$ tâches au plus trois fois sur $n+2$ processeurs. Les probabilités sont calculées et stockées en utilisant des opérations arithmétiques basiques ($y \leftarrow 1 - x$). Pour la complexité temporelle, le nombre d'appels aux lignes 6 et 21 est linéaire en n. Finalement, en utilisant une structure de données adéquate, la condition ligne 14 peut être vérifiée en temps constant, et la ligne 15 est appelée un nombre de fois linéaire en n. Cela conclut l'ensemble de la preuve. □

Le théorème 4.1 découle directement des propositions 4.3 et 4.6. Ce théorème ne dépend pas du type de pannes utilisées (transitoires ou franches) et est valide pour n'importe quel ordonnancement général. D'autre part, chaque probabilité de panne peut être un rationnel arbitraire. Enfin, ce résultat est aussi valide pour les ordonnancements ne possédant pas de temps libre avant l'exécution de chaque copie. Ainsi, ce résultat de complexité s'applique à une large classe de problèmes d'ordonnancement de graphes de tâches avec réplication.

Corollaire 4.7. *Approximer le résultat de* FIABILITÉ-GÉNÉRAL *à une quantité ε près ou à un facteur α près est un problème #P'-Complet.*

Démonstration. D'après le corollaire 1.3, approximer le résultat de CONNEXION à une quantité ε près ou à un facteur α près est un problème #P'-Complet. La preuve du théorème 1.6 montre que le résultat d'une instance de CONNEXION est le même que celui de l'instance de FIABILITÉ-GÉNÉRAL générée par l'algorithme de réduction (à une opération de complémentarité près, ce qui n'invalide pas le principe de parcimonie). □

Symbole	Définition
Q_i	Événement « toutes les tâches possédant un indice inférieur à i ont au moins une copie correctement exécutée »
$\mathcal{E}$	Intersection arbitraire d'événements $\overline{R_{ij}}$
$\mathcal{E}'$	Intersection d'événements $\overline{R_{ij}}$ et d'un événement $\mathcal{E}$

TABLE 4.3 – Liste des notations utilisées dans la section 4.5.2

4.5 Évaluation de la fiabilité d'un ordonnancement strict

Dans cette section, nous rappelons que le calcul de la fiabilité d'un ordonnancement strict a une complexité polynomiale avec des pannes transitoires [74]. Dans le cas des pannes franches, la complexité est ouverte. Comme nous conjecturons que le problème est aussi #P'-Complet, nous proposons un schéma d'évaluation exponentiel dont la complexité peut être réduite autant que nécessaire si une estimation de la fiabilité est suffisante.

4.5.1 Pannes transitoires

Pour les pannes transitoires, une formule close est fournie dans [74] pour calculer la fiabilité. Cette formule peut être évaluée en temps polynomial.

L'événement R_{ij}, tel que défini dans la section 4.3.4, est « le processeur p_j ne tombe pas en panne lors de l'exécution de la copie t_i^j de la tâche t_i ». L'équation 4.3.2 définit la fiabilité d'un ordonnancement strict π comme la probabilité qu'au moins une copie de chaque tâche soit exécutée correctement : $\text{fiab}(\pi) = \Pr\left[\bigcap_i \bigcup_j R_{ij}\right]$.

Pour simplifier $\text{fiab}(\pi)$, nous appliquons la loi de Morgan, $\overline{X \cup Y} = \overline{X} \cap \overline{Y}$, et obtenons : $\text{fiab}(\pi) = \Pr\left[\bigcap_i \overline{\bigcap_j \overline{R_{ij}}}\right]$. Les événements $\overline{R_{ij}}$ sont indépendants : ils correspondent à des pannes survenant sur des processeurs distincts. De plus, les événements $\overline{\bigcap_j \overline{R_{ij}}}$ sont indépendants car les pannes sont transitoires et le succès d'une copie pour une tâche n'impacte pas le reste de l'exécution. Après dérivation, nous obtenons

$$\text{fiab}(\pi) = \Pr\left[\bigcap_i \overline{\bigcap_j \overline{R_{ij}}}\right] = \prod_i \left(1 - \prod_j (1 - \Pr[R_{ij}])\right) \tag{4.5.1}$$

Cette équation est un résultat connu de la littérature [74].

4.5.2 Pannes franches

Nous traitons désormais du cas des pannes franches. Sans réplication, l'évaluation est polynomiale. Dans le cas contraire, le problème est ouvert. À notre connaissance, il n'existe pas de procédure polynomiale pour évaluer la fiabilité d'un ordonnancement strict (même en restreignant les graphes de tâches à des chaînes ou des tâches indépendantes). Nous conjecturons que l'évaluation de tels ordonnancements possède la même complexité que celle des ordonnancements généraux. Nous proposons donc un schéma d'évaluation exponentiel dont la complexité peut être réduite autant que nécessaire si une estimation de la fiabilité est suffisante.

Les notations introduites dans cette section sont résumées dans le tableau 4.3.

L'équation définissant la fiabilité d'un ordonnancement strict (voir section 4.3.4) ne se développe pas directement lorsqu'il s'agit de pannes franches car les événements R_{ij} ne sont alors plus

indépendants. C'est pourquoi nous proposons une formulation alternative de fiab(π) qui utilise les événements Q_i « toutes les tâches possédant un indice inférieur à i ont au moins une copie correctement exécutée ». L'événement Q_0 se produit toujours et Q_i se définit récursivement pour $i > 0$ par $Q_i = \overline{\bigcap_j \overline{R_{ij}}} \cap Q_{i-1}$: Q_i se produit si et seulement si au moins un processeur $p_j \in P$ ne tombe pas en panne pendant l'exécution de la copie t_i^j et si chacune des $i-1$ premières tâches a été correctement exécutée. Comme les tâches sont numérotées dans un ordre topologique, toutes les contraintes de dépendance de t_i sont satisfaites si Q_{i-1} se produit. Cette dernière formulation nous permet d'obtenir une expression récursive pour évaluer la fiabilité.

Comme la complexité de la méthode proposée est exponentielle dans le nombre de processeurs m, nous proposons un moyen de contrôler cette complexité en limitant la portée des évaluations récursives. En contrepartie, nous avons une estimation de la fiabilité à la place de la valeur exacte.

4.5.2.1 Méthode d'évaluation

La fiabilité d'un ordonnancement est donnée par fiab(π) = $\Pr[Q_n]$. Pour obtenir des dérivations utiles, nous introduisons un événement $\mathcal{E}$ qui est une intersection arbitraire d'événements $\overline{R_{ij}}$. Soit $\mathcal{E}' = \bigcap_j \overline{R_{ij}} \cap \mathcal{E}$. Alors, le calcul de $\Pr[Q_i \mid \mathcal{E}]$ dépend de $\Pr[Q_{i-1} \mid \mathcal{E}]$, $\Pr[Q_{i-1} \mid \mathcal{E}']$ et de quelques autres probabilités élémentaires, i.e., $\Pr\left[\overline{R_{ij}} \mid \mathcal{E}\right]$:

$$
\begin{aligned}
\Pr[Q_i \mid \mathcal{E}] &= \Pr\left[\overline{\bigcap_j \overline{R_{ij}}} \cap Q_{i-1} \mid \mathcal{E}\right] \\
&= \Pr\left[\overline{\bigcap_j \overline{R_{ij}}} \mid Q_{i-1} \cap \mathcal{E}\right] \times \Pr[Q_{i-1} \mid \mathcal{E}] \\
&= \left(1 - \Pr\left[\bigcap_j \overline{R_{ij}} \mid Q_{i-1} \cap \mathcal{E}\right]\right) \times \Pr[Q_{i-1} \mid \mathcal{E}] \\
&= \left(1 - \frac{\Pr\left[\bigcap_j \overline{R_{ij}} \cap Q_{i-1} \mid \mathcal{E}\right]}{\Pr[Q_{i-1} \mid \mathcal{E}]}\right) \times \Pr[Q_{i-1} \mid \mathcal{E}] \\
&= \left(1 - \frac{\Pr\left[Q_{i-1} \mid \bigcap_j \overline{R_{ij}} \cap \mathcal{E}\right]}{\Pr[Q_{i-1} \mid \mathcal{E}]} \Pr\left[\bigcap_j \overline{R_{ij}} \mid \mathcal{E}\right]\right) \times \Pr[Q_{i-1} \mid \mathcal{E}] \\
&= \left(1 - \frac{\Pr[Q_{i-1} \mid \mathcal{E}']}{\Pr[Q_{i-1} \mid \mathcal{E}]} \Pr\left[\bigcap_j \overline{R_{ij}} \mid \mathcal{E}\right]\right) \times \Pr[Q_{i-1} \mid \mathcal{E}] \\
&= \left(1 - \frac{\Pr[Q_{i-1} \mid \mathcal{E}']}{\Pr[Q_{i-1} \mid \mathcal{E}]} \prod_j \Pr\left[\overline{R_{ij}} \mid \mathcal{E}\right]\right) \times \Pr[Q_{i-1} \mid \mathcal{E}]
\end{aligned}
$$

La dernière ligne est obtenue en observant que tous les événements de l'intersection $\bigcap_j \overline{R_{ij}}$ concernent des processeurs distincts et sont donc indépendants. Notons également que $\Pr[Q_0 \mid \mathcal{E}] = 1$ quel que soit $\mathcal{E}$ car Q_0 se produit toujours. Ainsi, $\Pr[Q_n]$ peut être calculée récursivement.

Avant d'analyser la complexité de cette méthode d'évaluation, un mécanisme pour simplifier les intersections d'événements $\overline{R_{ij}}$ est introduit. En effet, tout événement $\mathcal{E} = \overline{R_{ij}} \cap \overline{R_{i'j}} \cap \ldots$ qui est une intersection d'au moins deux événements $\overline{R_{\cdot j}}$ qui concernent le même processeur p_j peut

être réduit en une définition plus concise. Seul un événement par processeur est requis : avec les pannes franches, dès qu'un processeur est tombé en panne, il reste dans le même état jusqu'à la fin de l'ordonnancement. Il est donc suffisant de considérer l'événement $\overline{R_{\cdot j}}$ qui concerne la première tâche ordonnancée sur p_j pour chaque processeur $p_j \in P$. Conséquemment, nous ne calculons aucune probabilité $\Pr[Q_i \mid \mathcal{E}]$ où $\mathcal{E}$ est une intersection de plus de m événements.

La complexité de ce mécanisme d'évaluation récursif est $O(n^{m+1})$. En effet, il y a n événements Q_i, et pour chacun il y a $O(n^m)$ intersections distinctes $\mathcal{E}$ (une intersection contenant au plus m éléments, chacun d'eux pouvant concerner l'une des n tâches). Nous proposons de contrôler l'exposant dans le coût de la complexité en faisant certaines estimations. Nous limitons la taille de chaque intersection $\mathcal{E}$ à k événements. Ceci est réalisé en enlevant des événements $\overline{R_{ij}}$ de $\mathcal{E}$ lorsque la taille de l'intersection dépasse le seuil k. Formellement, nous estimons que chaque intersection $\mathcal{E}'$ est équivalente à une intersection d'au plus k événements parmi $\bigcap_j \overline{R_{ij}} \cap \mathcal{E}$. Plusieurs choix sont possibles. Soit nous sélectionnons arbitrairement les k événements à conserver, ou nous appliquons une procédure heuristique. Par exemple, nous pourrions favoriser le sous-ensemble de taille k qui donne la plus faible probabilité pour $\Pr[Q_i \mid \mathcal{E}']$. Cette heuristique est motivée par la borne $\Pr\left[Q_i \mid \mathcal{E} \cap \overline{R_{ij}}\right] \leq \Pr[Q_i \mid \mathcal{E}]$ et aurait pour objectif de minimiser localement l'erreur réalisée par cette estimation. La complexité résultante est alors $O(n^{k+1})$ où $k \in [0..m]$ ($O(n^k)$ intersections distinctes $\mathcal{E}$ pour chacun des n événements Q_i).

Cette reformulation de la fiabilité et les dérivations proposées aboutissent invariablement à une complexité exponentielle pour notre mécanisme d'évaluation. Une autre approche, également exponentielle, consiste à considérer tous les scénarios possibles (voir la preuve de la proposition 4.3 pour plus de détails sur cette approche).

4.5.2.2 Validation empirique

Des simulations furent conduites pour valider cette méthode. Pour chaque simulation, un graphe de tâches aléatoire est généré avec 20 tâches aux coûts unitaires et 30 arcs de dépendance (plus un nombre arbitraire d'arcs qui assurent que les tâches source et puits sont chacune unique). La plateforme est composée de cinq processeurs homogènes avec des modèles de pannes Poissonnien et des taux de pannes tirés uniformément dans l'intervalle $[0; 0,05]$. Chaque tâche est ordonnancée deux fois sur des processeurs choisis aléatoirement. Au total, 300 ordonnancements sont obtenus et leurs fiabilités exactes se repartissent dans l'intervalle $[0,2; 1]$ en suivant approximativement une loi normale de moyenne $0,6$ et d'écart type $0,14$.

À cause de sa complexité, tester l'approche avec chaque valeur de k demanderait trop de ressources calculatoires si l'on augmentait le nombre de tâches ou le nombre de processeurs.

La figure 4.5.1 montre les temps pris par notre méthode pour chaque valeur de k. Comme attendu, ces durées augmentent exponentiellement avec k. Pour chaque valeur de k, les différences absolues entre les vraies fiabilités et celles estimées sont représentées par l'intermédiaire d'une boîte à moustaches. Dans une telle boîte, le ligne épaisse est la valeur médiane, la boîte représente les quartiles, les barres s'étendent jusqu'aux moustaches ($1,5$ fois l'intervalle interquartile) et les points additionnels sont des valeurs non-représentatives. Nous constatons que l'augmentation de k s'accompagne d'une amélioration de la précision de notre méthode (sauf pour $k = 2$). Pour $k = 3$, la précision médiane est de $0,86\%$.

La méthode proposée dans cette section permet donc de fournir une estimation de la fiabilité d'un ordonnancement strict lorsque les pannes sont franches. Cette estimation tend à s'améliorer lorsque le paramètre k augmente. Pour une valeur suffisamment grande, celle-ci devient alors précise.

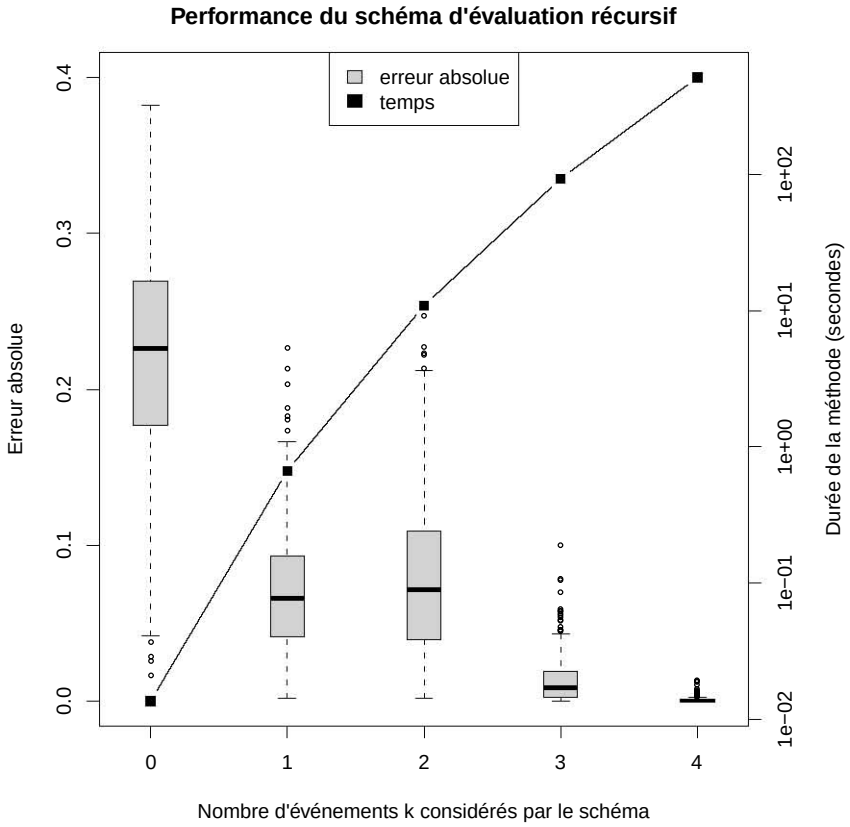


FIGURE 4.5.1 – Résultats des simulations avec 20 tâches sur cinq processeurs

Symbole	Définition
J_{ij}	Événement « la tâche t_i est exécutée correctement sur au moins l'un des j premiers processeurs »
Γ_i^j	Ensemble des processeurs sur lesquels le prédécesseur de la tâche t_i est ordonnancé et se termine avant que t_i ne commence sur le processeur p_j (vide si $i = 1$)
$\sigma(p_j)$	Ensemble des tâches ordonnancées sur le processeur p_j tel que t_{i-1} n'est pas ordonnancées sur p_j (contient t_1 si la copie t_i^j existe)
$\rho(i, j)$	Indice du plus proche ancêtre de t_i qui soit dans $\sigma(p_j)$

TABLE 4.4 – Liste des notations utilisées dans la section 4.6

4.6 Évaluation de la fiabilité d'un ordonnancement d'une chaîne de tâches avec des pannes franches

Dans cette section, nous restreignons les graphes de tâches aux chaînes et considérons que les pannes sont franches. Les ordonnancements de chaînes de tâches présentent parfois des particularités qui permettent d'en évaluer la fiabilité en temps polynomial. Nous décrivons d'abord une propriété présente dans certains ordonnancements (stricts ou généraux) ainsi qu'un algorithme qui permet de vérifier si un ordonnancement donné la possède. Nous proposons ensuite une méthode récursive à coût polynomial pour évaluer la fiabilité des ordonnancements possédant cette propriété. Enfin, nous verrons qu'il est possible d'obtenir des ordonnancements stricts optimaux qui suivent une structure spécifique que nous détaillerons. Par ailleurs, ces ordonnancements optimaux respectent la propriété proposée précédemment et sont donc évaluables en temps polynomial.

Les notations introduites dans cette section sont résumées dans le tableau 4.4.

4.6.1 Propriété de monotonie

La propriété définie dans cette section caractérise un sous-ensemble d'ordonnancements. Pour de tels ordonnancements et avec un indiçage des processeurs adéquat, le succès de la copie d'une tâche sur un processeur ne dépend que du succès de ce processeur et des précédents dans l'ordre donné. Intuitivement, le succès d'une copie sur le processeur p_j n'est alors conditionné que par les j premiers processeurs.

Sans perte de généralité, nous supposons que les tâches sont indicées suivant leurs profondeurs. Ainsi, la tâche t_1 est la tâche source et t_n la tâche puits. Soit Γ_i^j l'ensemble des processeurs qui calculent le prédécesseur de la tâche t_i et le terminent avant que t_i ne commence sur le processeur p_j, i.e., $\Gamma_i^j = \{p_{j'} \in P_{i-1} : C_{i-1}^{j'} \leq S_i^j\}$. Cela signifie que $p_{j'} \in \Gamma_i^j$ si et seulement si la paire de copies $(t_{i-1}^{j'}, t_i^j)$ est valide (voir la section 4.3.3 pour la définition d'une paire de copies valide). Pour la tâche t_1, les ensembles Γ_1^j sont vides pour tous les processeurs $p_j \in P_1$. Si le processeur p_j n'exécute pas t_i, alors Γ_i^j n'est pas définie. Par exemple, sur la figure 4.6.2b, $\Gamma_2^2 = \{p_1\}$ et $\Gamma_2^3 = \{p_1, p_3\}$. Nous nous intéressons à chaque tâche t_i ordonnancée sur le processeur p_j tel que son prédécesseur t_{i-1} n'est pas ordonnancé sur p_j, i.e., les tâches $t_i \in T_j$ pour lesquelles $p_j \notin \Gamma_i^j$. Soit $\sigma(p_j)$ l'ensemble de ces tâches pour le processeur p_j, i.e., $\sigma(p_j) = \{t_i \in T_j : p_j \notin \Gamma_i^j\}$. Par exemple, sur la figure 4.6.4a, $\sigma(p_2) = \{t_1, t_3\}$ et $\sigma(p_1) = \sigma(p_3) = \sigma(p_4) = \{t_1\}$.

Définition 4.8. *L'ordonnancement d'une chaîne de tâches est dit* monotone *si et seulement si les processeurs sont indicés tels que :*

(a) Condition violée par t_2^1 et l'existence de p_2 qui finit t_1 trop tôt

(b) Condition respectée

FIGURE 4.6.1 – Illustration de la condition 1 de la monotonicité

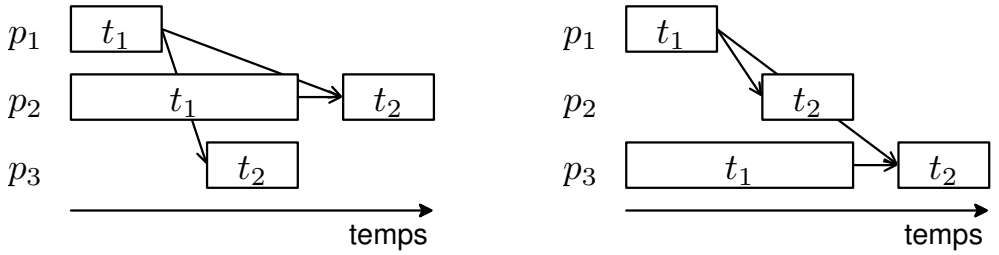
- *pour chaque processeur p_j et pour chaque tâche t_i dont le prédécesseur n'est pas ordonnancé sur p_j ($\forall p_j \in P, \forall t_i \in \sigma(p_j) \setminus \{t_1\}$) :*
  1. *il n'existe pas de processeur $p_{j'}$ avec un indice $j' > j$ qui termine le prédécesseur de t_i avant ou en même temps que la date de début de la copie t_i^j, i.e., $\nexists p_{j'} \in P_{i-1}, j' > j, C_{i-1}^{j'} \leq S_i^j$ (ce qui implique aussi que si une paire de copies $(t_{i-1}^{j'}, t_i^j)$ est valide, alors $j' \leq j$)*
  2. *il n'existe pas de processeur $p_{j'}$ avec un indice $j' \leq j$ qui termine le prédécesseur de t_i après la date de début de la copie t_i^j (ou, $\nexists p_{j'} \in P_{i-1}, j' \leq j, C_{i-1}^{j'} > S_i^j$)*
- *pour chaque processeur p_j et pour chaque tâche t_i ordonnancée sur p_j à l'exception de t_1, t_2 et des tâches $\sigma(p_j)$ ($\forall p_j \in P, \forall t_i \in T_j \setminus \{t_1, t_2\} \cup \sigma(p_j)$) :*
  3. *il n'existe pas de processeur qui termine la tâche t_{i-2} après la date de début de t_{i-1}^j mais avant ou en même temps que la date de début d'une copie $t_{i-1}^{j'}$ où $j' > j$ et où $p_{j'}$ finit la tâche t_{i-1} avant ou en même temps que la date de début de t_i^j, i.e., $\nexists p_{j'} \in P_{i-1}, j' > j, C_{i-1}^{j'} \leq S_i^j, \Gamma_{i-1}^{j'} \not\subseteq \Gamma_{i-1}^j$.*

Lorsque les processeurs sont indicés de telle manière à ce que l'ordonnancement soit monotone, nous dirons qu'ils suivent un ordre monotone.

La définition 4.8 impose trois types de contraintes sur l'indiçage des processeurs. Nous les illustrons par des exemples minimalistes sur les figures 4.6.1, 4.6.2 et 4.6.3. Sur ces figures, les flèches représentent les paires de copies valides. À chaque fois, nous présentons d'abord un ordonnancement où l'ordre des processeurs viole l'une des conditions, puis l'ordre monotone. La condition 1 n'est pas respectée sur la figure 4.6.1a car le processeur p_2 termine le prédécesseur t_1 de t_2 avant le début de t_2^1. L'indiçage des deux processeurs est inversée sur la figure 4.6.1b. Nous constatons alors que l'ordre permet de satisfaire toutes les conditions et que l'ordonnancement est monotone. Sur la figure 4.6.2a, le problème vient du processeur p_2 qui termine l'exécution de t_1 après le début de t_2 sur p_3 (violation de la condition 2). Sur la figure 4.6.3a, la copie t_3^1 dépend des copies t_2^1 et t_2^2. Cette dernière copie, t_2^2, dépend de davantage de copies que t_2^1, ce qui contredit la condition 3.

Notons que la condition 2 de la définition 4.8 implique que les tâches sont ordonnancées dans un ordre topologique sur chaque processeur.

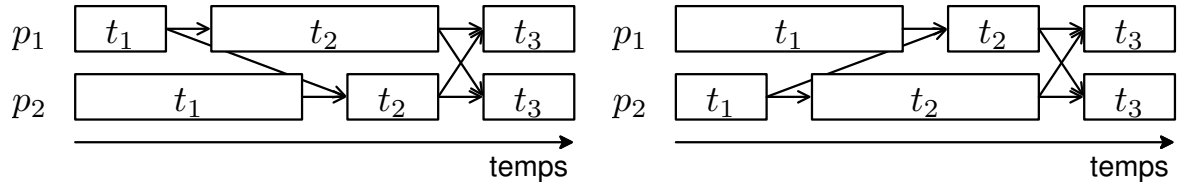
Ainsi que cela sera montré dans la section 4.6.3, certains ordonnancements stricts optimaux sont monotones. La portée de cette propriété est cependant plus large et permet au mécanisme d'évaluation polynomial présenté dans la section 4.6.2 d'être appliqué à un plus grand nombre d'ordonnancements (c'est-à-dire, aux ordonnancements généraux monotones).



(a) Condition violée par t_2^3 et l'existence de p_2 qui finit t_1 trop tard

(b) Condition respectée

FIGURE 4.6.2 – Illustration de la condition 2 de la monotonicité



(a) Condition violée par t_3^1 et l'existence de p_2 qui finit t_1 trop tard

(b) Condition respectée

FIGURE 4.6.3 – Illustration de la condition 3 de la monotonicité

Nous proposons désormais un algorithme à coût polynomial pour trier les processeurs dans un ordre monotone. L'idée est d'exprimer chaque condition sous forme d'arc dans un graphe orienté où les sommets correspondent aux processeurs. Dans ce graphe, un arc signifie que le processeur source doit être positionné avant le processeur destination.

L'algorithme 4.2 construit le graphe de la figure 4.6.4b lorsqu'il traite l'ordonnancement de la figure 4.6.4a. Les quatre sommets correspondent aux quatre processeurs et les arcs aux contraintes liées à la définition 4.8. L'arc a est ajouté ligne 7 lorsque $S_i^j = S_3^2$ et $C_{i-1}^{j'} = C_2^1$, c'est-à-dire lorsque l'on considère la paire de copies valide (t_2^1, t_3^2). Les arcs b et c sont rajoutés ligne 9 lorsque les paires de copies invalides sont respectivement (t_2^3, t_3^2) et (t_2^4, t_3^2). Enfin, l'arc d est inséré ligne 13 en considérant les deux derniers processeurs (de la même manière que sur la figure 4.6.3).

La justesse de l'algorithme 4.2 est énoncée dans la proposition qui suit.

Proposition 4.9. *Un ordonnancement est monotone si et seulement l'algorithme 4.2 n'échoue pas. De plus, l'ordre des processeurs produit par l'algorithme est monotone.*

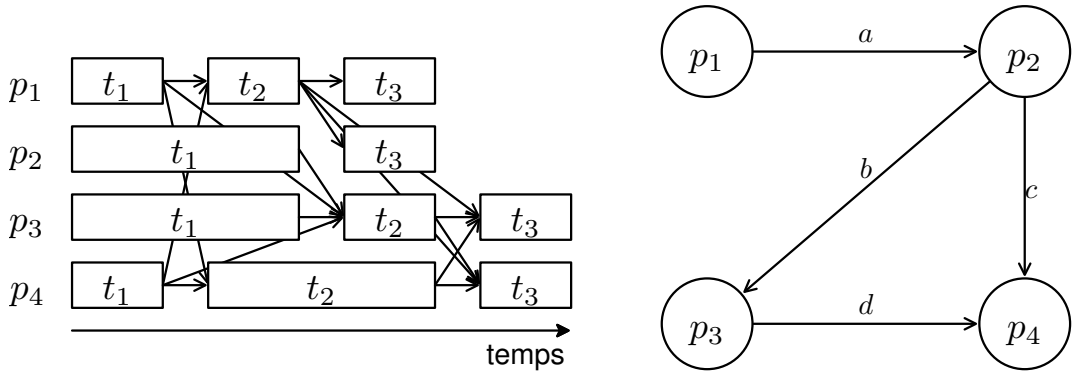
Démonstration. Supposons qu'il existe un arc $v_j \to v_{j'}$ dans le graphe construit par l'algorithme 4.2. Il est inséré soit à la ligne 7, 9 ou 13 qui correspondent respectivement aux contraintes imposées par les conditions 1, 2 et 3 de la définition 4.8. Il existe donc un arc $v_j \to v_{j'}$ si et seulement si l'indice de $p_{j'}$ ne peut pas être inférieur à celui de p_j dans l'ordre monotone.

La démonstration s'appuie sur ce fait pour prouver par l'absurde les deux affirmations suivantes : si le graphe est cyclique alors l'ordonnancement n'est pas monotone ; sinon, le graphe admet un tri topologique qui est un ordre monotone d'après la définition 4.8.

Nous considérons un ordonnancement pour lequel le graphe construit contient le cycle $v_j \to \ldots \to v_{j'} \to v_j$. Supposons qu'il existe un ordre des processeurs monotone. Sans perte de généralité, nous considérons que le premier processeur (par rapport à cet ordre monotone) qui soit

Algorithme 4.2 Algorithme de tri des processeurs

1: créer un sommet v_j pour chaque processeur p_j
2: **pour chaque** $p_j \in P$ **faire**
3: **pour chaque** $t_i \in T_j \setminus \{t_1\}$ **faire** $\{t_i$ est assignée à $p_j\}$
4: **pour chaque** $p_{j'} \in P_{i-1}$ **faire** $\{t_{i-1}$ est assignée à $p_{j'}\}$
5: **si** $t_i \in \sigma(p_j)$ **alors**
6: **si** $C_{i-1}^{j'} \leq S_i^j$ **alors**
7: créer l'arc $v_{j'} \to v_j$
8: **sinon**
9: créer l'arc $v_j \to v_{j'}$
10: **fin si**
11: **sinon**
12: **si** $C_{i-1}^{j'} \leq S_i^j$ et $\Gamma_{i-1}^{j'} \not\subseteq \Gamma_{i-1}^{j}$ **alors**
13: créer l'arc $v_{j'} \to v_j$
14: **fin si**
15: **fin si**
16: **fin pour**
17: **fin pour**
18: **fin pour**
19: **si** le graphe est acyclique **alors**
20: **retourner** un tri topologique
21: **sinon**
22: **afficher** échec de la procédure de tri
23: **fin si**



(a) Ordonnancement monotone

(b) Graphe obtenu pour l'ordre des processeurs

FIGURE 4.6.4 – Illustration du mécanisme de tri des processeurs

présent dans le cycle est p_j. Alors, p_j précède $p_{j'}$ ce qui contredit la sémantique de l'arc $v_{j'} \to v_j$, à savoir, l'indice de p_j ne peut être inférieur à celui de $p_{j'}$.

Nous considérons désormais que le graphe est acyclique. Supposons qu'il n'existe pas d'ordre de processeurs monotone. Comme le graphe est acyclique, nous en générons un tri topologique. Sans perte de généralité, nous considérons que p_j est positionné avant $p_{j'}$ dans ce tri et que cet agencement entre en conflit avec la définition de la monotonicité. Dans ce cas, il doit y avoir un arc entre $v_{j'}$ et v_j, ce qui contredit le fait que le tri permettant de placer p_j avant $p_{j'}$ soit topologique. □

4.6.2 Évaluation d'ordonnancements monotones

Avant d'introduire la formule récursive permettant d'évaluer la fiabilité d'un ordonnancement monotone en temps polynomial, nous proposons trois lemmes nécessaires à la démonstration de notre résultat.

Soit J_{ij} l'événement « la tâche t_i est exécutée correctement sur au moins l'un des j premiers processeurs ». Pour simplifier les formulations des conditions initiales dans les lemmes suivants, nous supposerons que les événements J_{0j} et J_{i0} se produisent toujours.

Lemme 4.10. *Soit un ordonnancement monotone d'une chaîne de tâches. Le succès de chaque copie t_i^j ne dépend que du succès des copies de son prédécesseur sur les j premiers processeurs et du succès de p_j, i.e., $U_{ij} = J_{i-1\,j} \cap R_{ij}$.*

Démonstration. D'après l'équation de la fiabilité 4.3.1, nous obtenons directement l'égalité $U_{ij} = \left(\bigcup_{p_{j'} \in \Gamma_i^j} U_{i-1\,j'}\right) \cap R_{ij}$. Nous montrons d'abord que l'inclusion

$$\bigcup_{p_{j'} \in \Gamma_i^j, j' > j} U_{i-1\,j'} \subseteq \bigcup_{p_{j'} \in \Gamma_i^j, j' \leq j} U_{i-1\,j'}$$

est vérifiée pour toute copie t_i^j.

Supposons que l'un des $m - j$ derniers processeurs $p_{j'}$ calcule correctement la tâche t_{i-1}. Nous considérons deux cas :

1. soit $t_i \in \sigma(p_j)$, auquel cas $p_{j'} \notin \Gamma_i^j$ par la condition 2 de la définition 4.8.
2. sinon, $t_i \notin \sigma(p_j)$. Comme l'ordonnancement est monotone, il vérifie la condition 3 de la définition 4.8. Cela signifie que les contraintes de dépendance satisfaites pour la copie $t_{i-1}^{j'}$ le sont aussi pour t_{i-1}^j.

Enfin, la condition 1 de la définition 4.8 assure que l'ensemble des processeurs $\{p_{j'} \in \Gamma_i^j : j' \leq j\}$ est identique à l'ensemble $\{p_{j'} \in P_{i-1} : j' \leq j\}$. La définition $J_{i-1\,j} = \bigcup_{p_{j'} \in P_{i-1}, j' \leq j} U_{i-1\,j'}$ termine la preuve. □

Lemme 4.11. *Soit un ordonnancement monotone d'une chaîne de tâches. La tâche t_i est exécutée correctement sur au moins l'un des j premiers processeurs si au moins l'un des descendants de t_i est exécuté correctement sur ces mêmes processeurs, i.e., $\forall i' \geq i, J_{i'j} \subseteq J_{ij}$.*

Démonstration. Nous montrons ce résultat par induction sur les indices des tâches. L'initialisation est obtenue pour $i' = i$. Dans ce cas, $J_{i'j} = J_{ij}$ et le lemme 4.11 est vérifié.

Nous supposons désormais que $J_{i'j} \subseteq J_{ij}$ est vrai (hypothèse d'induction) et montrons que $J_{i'+1\,j} \subseteq J_{ij}$ l'est aussi.

$$\begin{aligned}
J_{i'+1\,j} &= \bigcup_{j' \leq j} U_{i'+1\,j'} && \text{définition de } J_{i'+1\,j} \\
&= \bigcup_{j' \leq j} \left(J_{i'j'} \cap R_{i'+1\,j'}\right) && \text{lemme 4.10} \\
&\subseteq \bigcup_{j' \leq j} J_{i'j} && \text{élimination des intersections} \\
&\subseteq J_{i'j} && \text{définition de } J_{i'j} \\
&\subseteq J_{ij} && \text{hypothèse d'induction}
\end{aligned}$$

Dans la première ligne, nous avons utilisé la définition de l'événement J_{ij} : à chaque fois que l'événement J_{ij} se produit, il existe un processeur $p_{j'}$ parmi les j premiers processeurs qui exécute correctement la tâche t_i, i.e., $J_{ij} = \bigcup_{j' \leq j} U_{ij'}$. Dans l'avant-dernière ligne, nous considérons que pour tout processeur $p_{j'}$ avec pour indice $j' \leq j$, l'inclusion $J_{ij'} \subseteq J_{ij}$ découle naturellement. Cela conclut la démonstration. □

Nous notons $\rho(i,j)$ l'indice du plus proche ancêtre de t_i qui soit dans $\sigma(p_j)$, i.e., $\rho(i,j) = \max\{i' \leq i : t_{i'} \in \sigma(p_j)\}$.

Lemme 4.12. *Soit un ordonnancement monotone d'une chaîne de tâches sur des processeurs subissant des pannes franches. Chaque copie t_i^j est exécutée correctement si et seulement si p_j ne tombe pas en panne avant la fin de l'exécution de t_i et si au moins l'un des $j-1$ premiers processeurs exécute correctement la tâche $t_{\rho(i,j)-1}$, i.e., $U_{ij} = J_{\rho(i,j)-1\,j-1} \cap R_{ij}$.*

Démonstration. Nous prouvons d'abord que pour chaque tâche $t_i \in \sigma(p_j)$, $U_{ij} = J_{\rho(i,j)-1\,j-1} \cap R_{ij}$.

$$\begin{aligned}
U_{ij} &= J_{i-1\,j} \cap R_{ij} && \text{lemme 4.10} \\
&= J_{i-1\,j-1} \cap R_{ij} && t_i \in \sigma(p_j) \text{ implique } t_{i-1} \notin T_j \\
&= J_{\rho(i,j)-1\,j-1} \cap R_{ij} && t_i \in \sigma(p_j) \text{ implique } \rho(i,j) = i
\end{aligned}$$

Nous procédons par induction pour chaque tâche t_{i+1} assignée à p_j telle que $t_{i+1} \notin \sigma(p_j)$. Supposons que $U_{ij} = J_{\rho(i,j)-1\,j-1} \cap R_{ij}$ soit vrai. Nous allons montrer que l'égalité est aussi valide pour la copie t_{i+1}^j.

$$\begin{aligned}
U_{i+1\,j} &= J_{ij} \cap R_{i+1\,j} && \text{lemme 4.10} \\
&= (U_{ij} \cup J_{i\,j-1}) \cap R_{i+1\,j} && \text{définition de } J_{ij} \\
&= \left(J_{\rho(i,j)-1\,j-1} \cap R_{ij} \cup J_{i\,j-1}\right) \cap R_{i+1\,j} && \text{hypothèse d'induction} \\
&= \left(J_{\rho(i,j)-1\,j-1} \cup J_{i\,j-1}\right) \cap R_{i+1\,j} && R_{i+1\,j} \subseteq R_{ij} \\
&= J_{\rho(i,j)-1\,j-1} \cap R_{i+1\,j} && \text{lemme 4.11} \\
&= J_{\rho(i+1,j)-1\,j-1} \cap R_{i+1\,j} && t_{i+1} \notin \sigma(p_j) \text{ implique } \rho(i+1,j) = \rho(i,j)
\end{aligned}$$

Dans la quatrième ligne, nous utilisons le fait que les pannes sont franches, ainsi $R_{i+1\,j} \subseteq R_{ij}$.

Il reste à montrer que les inductions multiples qui sont réalisées sur chaque processeur p_j couvrent l'ensemble des tâches $t_i \in T_j$. Nous observons pour cela que chaque copie $t_i^j \notin \sigma(p_j)$ est précédée par au moins une copie $t_{i'}^j \in \sigma(p_j)$. □

Nous sommes désormais prêts à énoncer le résultat principal qui permet d'évaluer la fiabilité d'un ordonnancement monotone d'une chaîne de tâches en temps polynomial.

Théorème 4.13. *Soit R_{ij} l'événement « le processeur p_j n'échoue pas avant la fin de l'exécution de la copie t_i^j » et J_{ij} « la tâche t_i est exécutée correctement sur au moins l'un des j premiers processeurs ». Soit π un ordonnancement monotone d'une chaîne de tâches sur des processeurs subissant des pannes franches.*

La fiabilité de π se définit par $fiab(\pi) = \Pr[J_{nm}]$ où

$$\Pr[J_{i1}] = \Pr[R_{i1}] \tag{4.6.1}$$
$$\Pr[J_{1j}] = \Pr[J_{1\,j-1}] + (1 - \Pr[J_{1\,j-1}]) \Pr[R_{1j}] \tag{4.6.2}$$
$$\Pr[J_{ij}] = \Pr[J_{i\,j-1}] + (\Pr[J_{\rho(i,j)-1\,j-1}] - \Pr[J_{i\,j-1}]) \Pr[R_{ij}] \tag{4.6.3}$$

Démonstration. L'un des j premiers processeurs calcule correctement la tâche t_i si et seulement si l'un des $j-1$ premiers processeurs calcule correctement t_i ou si le processeur p_j calcule correctement t_i, i.e., $J_{ij} = J_{i\,j-1} \cup U_{ij}$. En utilisant le lemme 4.12, nous obtenons

$$\begin{aligned} J_{ij} &= J_{i\,j-1} \cup J_{\rho(i,j)-1\,j-1} \cap R_{ij} \\ \Pr[J_{ij}] &= \Pr[J_{i\,j-1}] + \Pr[J_{\rho(i,j)-1\,j-1} \cap R_{ij}] - \Pr[J_{i\,j-1} \cap J_{\rho(i,j)-1\,j-1} \cap R_{ij}] \end{aligned}$$

L'événement R_{ij} concerne le processeur p_j et est indépendant des événements $J_{i\,j-1}$ et $J_{\rho(i,j)-1\,j-1}$.

$$\Pr[J_{ij}] = \Pr[J_{i\,j-1}] + \Pr[J_{\rho(i,j)-1\,j-1}] \Pr[R_{ij}] - \Pr[J_{i\,j-1}] \Pr[J_{\rho(i,j)-1\,j-1} \mid J_{i\,j-1}] \Pr[R_{ij}]$$

Supposons que t_i soit assignée au processeur p_j (sinon, $\Pr[R_{ij}] = 0$ et l'équation se simplifie considérablement). Dans ce cas, le lemme 4.11 permet d'obtenir l'égalité $\Pr[J_{\rho(i,j)-1\,j-1} \mid J_{i\,j-1}] = 1$, ce qui prouve le théorème. □

Intuitivement, l'équation 4.6.3 signifie que la tâche t_i est exécutée correctement sur au moins l'un des $j-1$ premiers processeurs (ce qui correspond à l'événement $J_{i\,j-1}$) ou si elle est exécutée sur le processeur p_j (c'est-à-dire si les événements $J_{\rho(i,j)-1\,j-1}$ et R_{ij} se produisent). Comme ces deux possibilités sont dépendantes, il faut retrancher leur intersection.

Les équations 4.6.1, 4.6.2 et 4.6.3 du théorème 4.13 définissent un schéma d'évaluation récursif pour déterminer la fiabilité d'un ordonnancement monotone d'une chaîne de tâches. L'évaluation utilise uniquement les probabilités des événements R_{ij} qui font partie de la définition du problème. Pour vérifier que la récursion est polynomial, nous considérons le nombre maximal d'événements J_{ij} dont les probabilités nécessitent d'être évaluées. Comme chaque processeur peut exécuter chaque tâche au plus une fois, la complexité est en $O(nm)$.

Notons que le type de pannes considéré est essentiel pour prouver le lemme 4.12. Dans le cas des pannes transitoires, le schéma d'évaluation récursif défini n'est pas applicable.

Remarquons enfin que la propriété de monotonie est suffisante mais pas nécessaire pour utiliser les équations du théorème 4.13.

4.6.3 Ordonnancement strict sur machines uniformes

Nous décrivons un problème qui illustre la portée du schéma d'évaluation proposée dans la section 4.6.2. Le problème consiste à optimiser la fiabilité et la durée totale d'un ordonnancement strict d'une chaîne de tâches. Comme le problème est multicritère, nous recherchons un ensemble

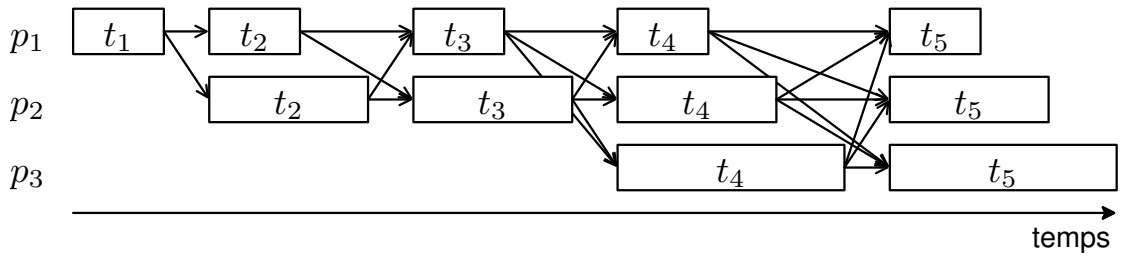


FIGURE 4.6.5 – Exemple d'ordonnancement triangulaire sur trois processeurs et avec cinq tâches

d'ordonnancements optimaux qui fournissent chacun un compromis différent pour les deux objectifs. Le problème consiste donc à générer des solutions Pareto-optimales (voir le chapitre 2 pour des détails sur la notion de Pareto-optimalité).

Nous continuons à considérer que les pannes sont franches. Comme les ordonnancements sont stricts, l'évaluation de la fiabilité avec des pannes transitoires possède une solution déjà décrite dans la section 4.5.1.

Nous postulons que les machines sont uniformes [76]. Rappelons que la copie t_i^j nécessite w_i^j unités de temps. Dans le cas d'une plateforme uniforme, nous avons $w_i^j = c_i \times \tau_j$, où c_i est le coût de la tâche t_i et τ_j est le temps de cycle du processeur p_j. Cela signifie que l'on peut trier les processeurs suivant leurs vitesses. Par la suite, nous indiçons les processeurs par temps de cycle croissant. Le processeur p_1 est donc la machine la plus rapide et p_m la plus lente.

Grâce à cette hypothèse, nous sommes en mesure de caractériser un sous-ensemble d'ordonnancements stricts qu'il nous suffit de considérer dans notre contexte d'optimisation bicritère.

Définition 4.14. *L'ordonnancement d'une chaîne de tâches est dit* triangulaire *si et seulement si*

1. *chaque processeur qui exécute au moins une tâche exécute également tous ses descendants dans un ordre topologique*
2. *et si un processeur exécute une tâche, alors tous les processeurs plus rapides l'exécutent aussi.*

Dans un ordonnancement de chaîne de tâches triangulaire, la première tâche exécutée sur le processeur p_j est alors dénotée $\rho(n, j)$ selon la condition 1. Nous avons d'autre part que $\forall p_j \in P_n, \forall p_{j'} \in P, j' < j \Rightarrow \rho(n, j') \leq \rho(n, j)$ d'après la condition 2. Ce qui équivaut à dire que l'indice de la première tâche ordonnancée sur chaque processeur est croissant monotone lorsque les processeurs sont triés par vitesse croissante. Un exemple d'un ordonnancement strict de chaîne de tâches vérifiant cette propriété est représenté sur figure 4.6.5.

Proposition 4.15. *Considérons un problème d'ordonnancement strict d'une chaîne de tâches sur des machines uniformes subissant des pannes franches. Pour chaque solution, il existe un ordonnancement triangulaire qui possède des valeurs objectives meilleures ou égales.*

Démonstration. Soit un ordonnancement π donné. Nous montrons que nous pouvons construire un ordonnancement triangulaire qui possède des valeurs objectives meilleures ou égales à celles de π.

Supposons que π ne vérifie pas la condition 2 de la définition 4.14. Alors, pour chaque tâche t_i, nous pouvons compléter l'ordonnancement en assignant des copies de t_i aux processeurs plus rapides que le processeur le plus lent qui exécute t_i. Comme l'ordonnancement est strict, cette opération n'ajoute pas de temps à la durée totale de π car l'intervalle de temps consacré à t_i est

déterminé par le processeur le plus lent qui l'exécute. D'autre part, la fiabilité ne diminue pas car le succès de chaque copie déjà ordonnancée ne dépend que de ses dates de début et de fin d'exécution (voir section 4.3.2).

Nous pouvons donc considérer que l'ordonnancement π est complété de manière à respecter la condition 2. Supposons que π ne vérifie pas la condition 1. Alors, il existe une tâche t_i qui est assignée au processeur p_j sans que la tâche t_{i+1} ne lui soit assignée, i.e.,$\exists p_j \in P, \exists t_i \in T_j, t_{i+1} \notin T_j$. Soit $p_{j'}$ le processeur le plus lent sur lequel est ordonnancée la tâche t_{i+1}. Comme π vérifie la condition 2 et que t_{i+1} n'est pas assignée à p_j, ce processeur est plus rapide que p_j. La tâche t_i est donc également ordonnancée sur tous les processeurs plus rapide que $p_{j'}$. Comme les pannes sont franches, le succès de la copie t_i^j n'a pas d'incidence sur le succès des copies de la tâche t_{i+1} assignée à $p_{j'}$ et aux processeurs plus rapides que $p_{j'}$. La copie t_i^j est donc inutile et peut être supprimée pour diminuer la durée totale de π. En réalisant de même pour chaque tâche t_i assignée au processeur p_j sans que la tâche t_{i+1} ne lui soit assigné, nous obtenons un ordonnancement vérifiant la condition 1. □

Nous présentons un corollaire direct de la proposition 4.15 : à partir d'un ordonnancement Pareto-optimal, nous pouvons obtenir un ordonnancement triangulaire qui possède les mêmes valeurs objectives. Cette propriété permet de restreindre la classe des ordonnancements considérés à celle des ordonnancements triangulaires.

Proposition 4.16. *Soit π un ordonnancement triangulaire strict d'une chaîne de tâches sur des machines uniformes subissant des pannes franches. Alors, l'ordonnancement π est monotone.*

Démonstration. Pour chaque processeur p_j qui exécute au moins une tâche, nous avons $\sigma(p_j) = \{t_{\rho(n,j)}\}$. D'autre part, il n'existe pas de processeur parmi les $m - j$ derniers auquel est assigné la tâche $t_{\rho(j,n)-1}$ (implication de la condition 2 de la définition 4.14). L'unique tâche de $\sigma(p_j)$ ne dépend donc que des $j - 1$ premiers processeurs. Un ordonnancement triangulaire strict vérifie donc la condition 1 de la définition 4.8.

Par ailleurs, dans un ordonnancement strict, une tâche débute son exécution lorsque toutes les copies de son prédécesseur sont terminées. Ainsi, les conditions 2 et 3 de la définition 4.8 sont toujours vérifiées. □

Par conséquent, nous pouvons évaluer en temps polynomial la fiabilité d'un ordonnancement Pareto-optimal pour le problème considéré dans cette section, à savoir l'optimisation bicritère de la fiabilité et de la durée totale d'ordonnancements stricts de chaînes de tâches sur machines uniformes subissant des pannes franches. Notons que si un ordonnancement Pareto-optimal n'est pas triangulaire, nous pouvons le compléter de manière à ce qu'il le devienne.

4.7 Conclusion

La figure 4.7.1 dresse un bilan de la complexité des problèmes traités dans ce chapitre. Les flèches représentent les réductions polynomiales possibles entre les problèmes. Un ordonnancement sur une plateforme avec des pannes transitoires peut être transformé en temps polynomial en un ordonnancement sur une plateforme avec des pannes franches. Il faut pour cela positionner chaque copie sur un processeur séparé (s'il n'y a qu'une seule tâche par processeur, alors les pannes transitoires et franches sont équivalentes). Le nombre de processeurs requis dans le nouvel ordonnancement est plus important (nous passons de m processeurs à $O(nm)$ processeurs).

La preuve démontrant la #P'-Complétude de l'évaluation de la fiabilité dans le cas général constitue la contribution majeure de ce chapitre. Dans le cas où la complexité est inconnue,

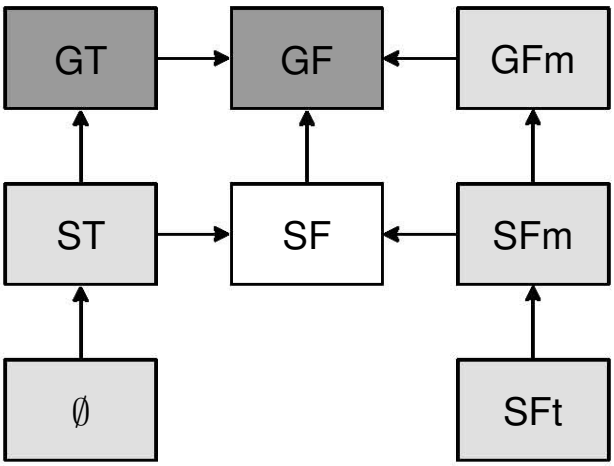


FIGURE 4.7.1 – Bilan de la complexité des problèmes d'évaluation (gris clair pour les problèmes polynomiaux, blanc pour le problème ouvert, gris foncé pour les problèmes #P'-Complet). Les lettres caractérisent les différentes problématiques : G pour les ordonnancements généraux ; S pour les ordonnancements stricts ; T pour les pannes transitoires ; F pour les pannes franches ; m pour les ordonnancements monotones ; t pour les ordonnancements triangulaires ; $\emptyset$ pour les ordonnancements sans réplication.

des développements analytiques aboutissent sur une méthode heuristique. La section présentant l'étude d'un cas particulier de graphe de tâches souligne la difficulté que présente l'évaluation de la fiabilité même dans un cas spécifique. Cela conforte notre conjecture selon laquelle le problème ouvert est aussi #P'-Complet.

Chapitre 5

Caractérisation dynamique de la fiabilité des ressources calculatoires dans les plateformes volontaires

Sommaire

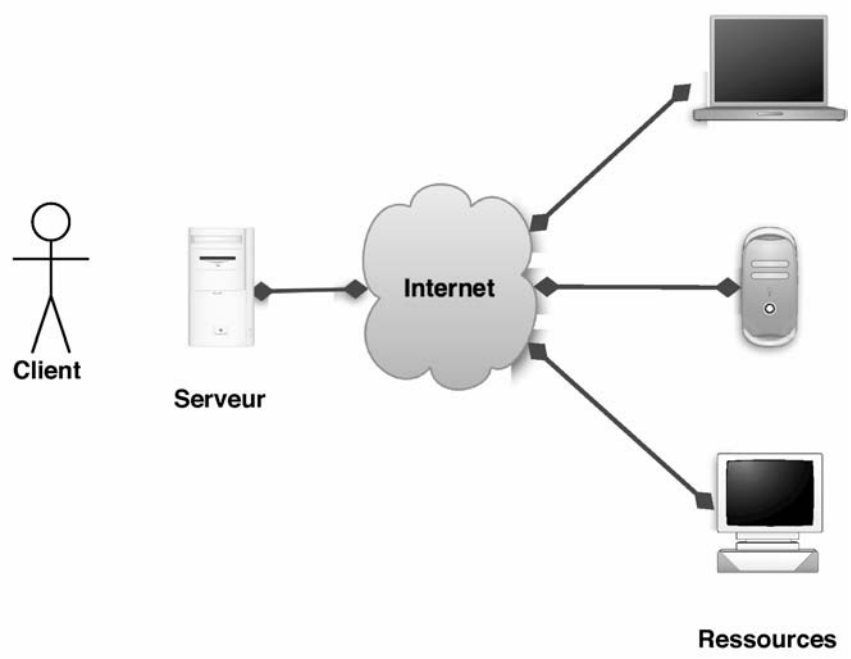


FIGURE 5.1.1 – Une plateforme de calcul distribué

5.1 Introduction

Nous introduisons d'abord le contexte des plateformes de calcul distribué et le risque présenté par le manque de contrôle dans ces plateformes, surtout les plateformes volontaires. Nous évoquons le lien entre la nature de l'incertitude considérée et notre approche au problème, et terminons sur le plan de ce chapitre.

5.1.1 Description du contexte

Les plateformes de calcul distribué telles que BOINC [16] (voir la figure 5.1.1) offrent une capacité de calcul importante mais sont sujettes à de nombreuses incertitudes.

Ces plateformes sont composées d'un ensemble de *ressources calculatoires*. Dans le cas des plateformes volontaires, celles-ci sont mises à disposition par des participants bénévolement. Dans ce cas, chaque participant, ou *entité*, offre une partie de la capacité de calcul de sa ou ses machines à une plateforme de calcul distribué en utilisant une *identité* virtuelle qui permet de l'authentifier.

Ce type d'infrastructures rend possible le déroulement de simulations scientifiques intensives [17, 15, 105]. De nombreuses applications à objectif académique ou médical sont développées afin de profiter de la puissance de calcul ainsi disponible. Dans ces projets, la charge de travail est divisée en un ensemble de tâches qui sont distribuées aux ressources après une étape d'ordonnancement.

Les avantages des plateformes de calcul distribué sont multiples. Reposant souvent sur le volontariat, le coût économique est fortement réduit pour une grande puissance de calcul. Par exemple, le déploiement, la gestion et la maintenance des ressources calculatoires devient un problème négligeable par rapport à la complexité de ces opérations dans des environnements de calcul centralisés.

En contrepartie, la quantité, la dispersion et le manque de contrôle des ressources posent

problèmes. Dans ces plateformes, les temps de calcul sont incertains, la disponibilité d'une ressource à un instant donné n'est pas garantie et tout résultat obtenu à l'issu d'un calcul peut être incorrect. Cela rend la plateforme imprévisible et potentiellement instable.

Il en découle que la fiabilité affichée par le système est à prendre en considération. La triche est un cas de figure illustratif des risques existants. Lorsqu'un participant malveillant produit des *résultats incorrects*, il est nécessaire d'avoir un mécanisme de certification pour valider les résultats corrects et invalider ceux qui ne le sont pas. À cette fin, il existe plusieurs approches. Citons par exemple les quizzes pré-calculés [159] ou encore les systèmes de vote utilisés conjointement avec un système de réputation [138].

Procéder à un vote est efficace si chaque résultat incorrect est unique (en considérant que les collisions sont improbables). Cependant, ce n'est plus le cas si les participants décide de tricher de façon organisée en faisant preuve de collusion ou s'il existe une erreur matérielle ou logicielle reproductible sur certaines ressources spécifiques. Citons aussi le cas d'une infection par un virus ou le cas de l'attaque Sybil [56]. Toutes ces situations rendent les plateformes de calcul distribué vulnérables aux *fautes collectives*, c'est-à-dire la production du même résultat incorrect par plusieurs ressources distinctes.

5.1.2 Typologie

D'après la taxonomie proposée dans [40], notre étude se positionne dans le contexte des plateformes de calcul distribué centralisées, à savoir les plateformes où chaque décision est prise par un serveur unique.

Le problème de l'ordonnancement sur plateforme de calcul distribué consiste à distribuer des tâches dont les durées sont inconnues sur des machines hétérogènes dont les disponibilités sont inconnues. Ce problème se note $R|online - time - nclv|\sum C_i$ suivant la notation $\alpha|\beta|\gamma$ [76] et celle décrite dans [121]. L'objectif est alors de minimiser la latence moyenne, c'est-à-dire la durée moyenne pendant laquelle chaque tâche reste dans le système.

L'incertitude concerne la justesse des résultats et nous enrichissons cette problématique d'un second critère relatif à la validité de l'ensemble des résultats générés. Comme évoqué plus haut, il existe plusieurs causes à ces incertitudes. Elles peuvent être la conséquence de comportements humains ce qui est le cas lorsque les participants trichent. Nous n'écartons toutefois pas les erreurs matérielles ou logicielles. Remarquons finalement que le problème existe à cause de l'incapacité à déterminer si une ressource donnée est fautive ou non. Cette ignorance de la part de l'ordonnanceur illustre le caractère épistémique de l'incertitude traitée (tel que défini dans l'introduction de ce manuscrit). Pour mettre cette remarque en perspective, notons que le doute porté sur la validité d'un résultat n'est pas lié à l'insuffisance des méthodes mises en place pour certifier le résultat (incertitude méthodologique). C'est en revanche l'ignorance qui est en jeu et notre solution consistera à la limiter. Le modèle défini plus bas introduit des probabilités de faute insérant ainsi une variabilité aléatoire aux comportements des ressources.

5.1.3 Description de l'étude

Le problème évoqué dans cette introduction se focalise sur deux critères : l'efficacité et la précision. L'incertitude sur la validité des résultats rend ce dernier critère difficile à optimiser. Pour gérer cette incertitude, nous proposons de réduire l'ignorance et pour cela, nous nous intéressons à la caractérisation des ressources fautives (d'un point de vue qualitatif et quantitatif). Notre contribution consiste en un mécanisme de caractérisation qui se greffe sur un algorithme d'ordonnancement existant. À chaque fois que le résultat d'une tâche est reçu par le serveur, le mécanisme proposé considère le résultat comme une nouvelle observation pour améliorer les

estimations qu'il produit sur les propriétés fautives des ressources. Le problème traité n'est donc pas celui de l'ordonnancement, mais celui de la caractérisation des ressources.

Nous décrivons dans la section 5.4 notre méthode pour représenter, détecter et caractériser les fautes collectives. Comme nous manipulons deux structures de données, nous aboutissons sur deux mécanismes de caractérisation. Ils sont par ailleurs comparés empiriquement dans la section 5.5 en utilisant des traces d'exécution réelles.

5.2 État de l'art

Bien que les plateformes de calcul distribués aient été amplement étudiées, il existe peu de travaux sur la caractérisation de la fiabilité des ressources en présence de fautes collectives.

Nous allons donc articuler l'état de l'art autour des travaux principaux en lien avec la problématique générale de la production de résultat fiable sur ces plateformes. Dans un premier temps, nous décrirons une attaque compromettante et postulerons qu'un mécanisme est utilisé pour la limiter. Ensuite, nous aborderons des solutions algorithmiques complètes pour l'ordonnancement et la certification des résultats. Après avoir remarqué que ces systèmes nécessitent souvent un système de réputation, nous présenterons différentes solutions. Bien que ces systèmes de réputation caractérisent la fiabilité des ressources, ils ignorent les fautes collectives. Nous finirons sur un domaine transversal dont la problématique se rapproche de la nôtre.

5.2.1 Attaque Sybil

La mise en place d'une attaque Sybil par un participant malveillant constitue une source dangereuse de collusion. Dans [56], Douceur distingue le concept d'entité (qui correspond à un individu physique) et d'identité (qui correspond à un compte virtuel). L'attaque qu'il présente, nommée *attaque Sybil*, consiste à multiplier démesurément les identités possédées par une unique entité. Cette attaque menace particulièrement les systèmes dans lesquels la création d'un compte (une identité) est facile. Une fois réalisée, l'attaquant dispose alors d'une quantité d'identités qui lui permettent d'obtenir la majorité lors d'un vote par exemple. Dans ce cas, la certification des résultats devient impossible si aucune référence n'est disponible. Dans notre cas, les participants prêtent de la puissance de calcul et il est alors possible de limiter partiellement une attaque Sybil à l'aide de puzzles calculatoires [109] ou par l'intermédiaire d'un système d'invitation et de récompense comme proposé dans [54].

Nous supposons désormais qu'un mécanisme quelconque est mis en place et permet de limiter l'impact de toute attaque Sybil.

5.2.2 Ordonnancement et certification

Différentes solutions existent pour l'ordonnancement de tâches sur des ressources calculatoires et la certification des résultats dans les plateformes de calcul distribué. Ces solutions peuvent se classifier en considérant les propriétés suivantes :

Redondance De nombreuses solutions se basent sur la duplication des tâches sur les ressources calculatoires, i.e., exécuter plusieurs fois la même tâche pour accroître les chances d'obtenir un résultat correct.

Tâche vérifiable ou *quiz*. Des tâches dont les résultats sont connus sont insérées dans la liste de tâches à traiter. Assurer que les tâches vérifiables ne se distinguent pas des autres tâches est un problème difficile.

Ressource calculatoire sûre Cette ressource exécute certaines tâches et les résultats générés ne comportent aucune erreur. Cela peut être utilisé avantageusement en comparant ces résultats corrects à ceux obtenus par d'autres ressources dont la fiabilité est inconnue.

Liste noire Le principe consiste à marquer les ressources qui ont produit un résultat incorrect et à les ignorer. En revanche, cela ne protège pas le système des participants qui se reconnectent avec une nouvelle identité en abandonnant l'ancienne. D'autre part, Kondo et al. montrent dans [100] que cette solution est inefficace à cause de la faible proportion de ressources parfaitement fiables.

Vérification intermédiaire ou *checkpoint*. Il est possible de réaliser des vérifications pendant l'exécution d'une tâche pour accélérer la détection des fautes.

Spécificité Le mécanisme peut être spécifique à une application donnée. En effet, les tâches de certaines applications présentent des particularités exploitables lors de la certification de résultats.

Centralisation Bien que les ressources calculatoires soient reparties, les décisions d'ordonnancement et de certification sont soit centralisées, soit entièrement distribuées. Dans ce dernier cas, le système est alors plus robuste à une attaque ciblée.

5.2.2.1 Faute individuelle

La plateforme de calcul distribué BOINC [16] contient le mécanisme d'ordonnancement et de certification le plus utilisé en pratique. Sur cette plateforme, les tâches sont distribuées de façon redondante aux ressources calculatoires dans l'optique d'atteindre un quorum de résultat. Ainsi, chaque tâche est d'abord distribuée à o ressources. Si un quorum de q résultats est obtenu (à savoir, q ressources retournent le même résultat), la tâche est terminée. Sinon, elle est assignée à une nouvelle ressource et cette étape est répétée jusqu'à obtention du quorum recherché. Cependant, la tâche ne peut pas être envoyée à plus de l ressources. Si le quorum n'est toujours pas atteint avec l résultats, alors le résultat qui remporte le plus d'adhésions est certifié.

Un algorithme d'ordonnancement avec redondance des tâches est proposé dans [138]. Il se base sur un système de réputation qui estime la probabilité qu'une ressource retourne un résultat incorrect, ce qui est utilisé pour déterminer quelles ressources doivent calculer une tâche donnée.

Sarmenta présente un mécanisme basé sur la redondance, les tâches vérifiables et la liste noire dans [130]. Le résultat certifié est celui qui est majoritaire. Il décrit aussi un système de réputation basé sur la crédibilité pour rendre compte de la fiabilité des ressources.

5.2.2.2 Gestion de la collusion

Ces précédents travaux ne gèrent que les fautes individuelles. Les travaux décrits dans cette section sont conçus pour résister à la collusion des ressources.

Zhao et al. [159] présentent deux solutions basées sur le principe de redondance et de tâche vérifiable et les analysent de façon probabiliste. De plus, un système basé sur la confiance est décrit et étend les deux approches proposées.

Domingues et al. [54] proposent une méthode pour s'assurer du déroulement correct de l'exécution des tâches. Il s'agit de réaliser des vérifications intermédiaires en comparant l'état d'un calcul à une étape donnée à celui des autres calculs comparables. Ces vérifications se basent donc sur la redondance des calculs pour pouvoir les comparer et détecter le plus tôt possible les divergences.

Silaghi et al. [136] décrivent une technique pour lutter contre la collusion. Leur mécanisme est basé sur le principe de redondance et sur un système de réputation utilisant l'algorithme

EigenTrust. Les participants détectés comme fautifs sont mis sur une liste noire et les tâches pour lesquelles ils avaient retourné un résultat sont resoumises. Cependant, ce travail possède quelques désavantages : il suppose que l'algorithme de détection n'est pas connu par le participant et l'algorithme nécessite d'attendre que toutes les tâches soient finies avant de certifier les résultats.

Similairement, Yurkewych et al. [156] proposent une solution pour estimer le coût d'un mécanisme de redondance en présence de collusion grâce à la théorie des jeux. Leur travail vise une plateforme commerciale et se base sur la redondance et les tâches vérifiables.

Krings et al. [103, 104] étudient le problème de la certification des résultats lorsque les tâches sont soumises à des contraintes de dépendance. Leurs travaux permettent de détecter si une attaque massive a eu lieu et requiert des ressources sûres.

Certains de ces travaux se basent sur la possibilité de vérifier le résultat d'une tâche. Comme ce n'est pas toujours possible, nous ne considérons pas cette possibilité. Toutefois, ces approches sont complémentaires à nos travaux. D'autre part, la possibilité de faire des vérifications intermédiaires et tout mécanisme spécifique à une application précise sont écartés afin de rester générique dans notre approche du problème.

5.2.3 Système de réputation

Nous pouvons remarquer que de nombreuses solutions d'ordonnancement et de certification (voir section 5.2.2) reposent sur une caractérisation de la fiabilité des ressources calculatoires. Il s'agit alors d'estimer la probabilité qu'une ressource se compote de façon satisfaisante. Les systèmes de réputation rentrent dans ce cadre car leur objectif est d'associer un indice de réputation à chaque identité présente sur une plateforme donnée.

Dans [92], Jøsang et al. inventorient un ensemble de solutions existantes et donnent quelque critères pour évaluer la qualité d'un système de réputation (comme la précision et la robustesse).

L'algorithme EigenTrust [94] est sans doute l'un des systèmes les plus représentatifs de la littérature. Il associe à chaque identité une valeur qui agrège l'ensemble des valeurs de confiance qu'ont les autres identités envers celle-ci. Comme il fonctionne de façon itérative, à la deuxième itération, il prend aussi en compte la confiance qu'ont les identités envers les identités qui font confiance à l'identité en question. Ainsi, l'algorithme converge et produit une valeur pour chaque identité qui représente la confiance globale de l'ensemble du système pour cette identité. Il peut être appliqué dans tout système où des identités interagissent (pour le calcul, le partage de fichiers ou le transfert de données). Il présente enfin l'avantage de fonctionner de façon décentralisée.

Jøsang introduit dans [90] une logique subjective pour exprimer la confiance dans les réseaux. Chaque score est modélisé par une *opinion* : un triplet de croyance, d'incroyance et d'incertitude. Cette logique manipule les opinions pour produire des recommandations ou des consensus. Ce système peut donc aussi être utilisé pour l'ordonnancement et la certification des résultats.

Le système de réputation H-trust [158] agrège partiellement les confiances présentes dans un réseau. Il s'appuie sur un protocole que les participants doivent respecter et ne s'applique donc pas directement dans le contexte des plateformes de calcul distribué.

Les systèmes de réputation en général ne visent pas la détection des fautes collectives. Notre objectif n'est pas de produire une estimation de la fiabilité, mais de caractériser les propriétés fautives des participants (quantité et composition des groupes de ressources commettant des fautes collectives).

5.2.4 Diagnostic des fautes

La caractérisation des fautes collectives est liée à un dernier domaine : le diagnostic des fautes [23, 58, 102, 122] dont l'objectif est de trouver des processeurs fautifs en réalisant une série

Symbole	Définition
P	Ensemble des ressources calculatoires
n	Nombre de ressources ($n = \|P\|$)
$\mathcal{G}$	Ensemble des groupes réels
$\hat{\mathcal{G}}$	Ensemble des groupes observés
l	Nombre de groupes réels ($l = \|\mathcal{G}\|$)
$\hat{l}$	Nombre de groupes observés ($\hat{l} = \|\hat{\mathcal{G}}\|$)
C	Matrice de faute réelle (de taille l)
$\hat{C}$	Matrice de faute estimée (de taille $\hat{l}$)
A	Matrice d'affinité réelle (de taille l)
$\hat{A}$	Matrice d'affinité estimée (de taille $\hat{l}$)
c_{ij}	Probabilité de faute collective entre les groupes i et j
$\hat{c}_{ij}$	Probabilité estimée de faute collective entre les groupes observés i et j
a_{ij}	Probabilité d'affinité entre les groupes i et j
$\hat{a}_{ij}$	Probabilité estimée d'affinité entre les groupes observés i et j
$g(p)$	Groupe contenant la ressource p ($g(p) \in \mathcal{G}$)
$\hat{g}(p)$	Groupe observé contenant la ressource p ($\hat{g}(p) \in \hat{\mathcal{G}}$)
L	Sous-ensemble de ressources ($L \subseteq P$)
K_L	Ensemble des groupes couvrant chaque ressource appartenant à L ($K_L = \bigcup_{p \in L} \{g(p)\}$)
$\hat{K}_L$	Ensemble des groupes observés couvrant chaque ressource appartenant à L ($\hat{K}_L = \bigcup_{p \in L} \{\hat{g}(p)\}$)
E_L	Événement « toutes les ressources appartenant à L échouent collectivement pour une tâche donnée »
$P_{r,t}$	Ensemble des ressources ayant renvoyées le résultat r pour la tâche t
$\#\hat{c}_{ij}$	Nombre d'interactions prises en compte par l'estimation $\hat{c}_{ij}$
$I(\hat{c}_{ij})$	Intervalle de confiance associé à l'estimation $\hat{c}_{ij}$

TABLE 5.1 – Notations du chapitre

de requêtes entre les processeurs. Ces techniques tolèrent les fautes collectives et ne requièrent qu'une majorité de processeurs corrects. Dans notre cas, nous n'avons pas de contrôle sur les requêtes (i.e., la façon dont les tâches sont distribuées aux participants). Par ailleurs, ces travaux présupposent que les processeurs ne sont pas hostiles et n'essaient pas de tromper le mécanisme de détection.

5.3 Modèles et définitions

Les notations utilisées dans ce chapitre sont résumées dans le tableau 5.1.

5.3.1 Application et plateforme

Nous proposons le modèle suivant pour une plateforme de calcul distribué, modèle inspiré directement de BOINC [16] :

– Un ensemble de tâches est donné et chacune doit être exécutée sur la plateforme.

- Nous avons n ressources calculatoires. Chaque ressource $p \in P$ est capable de calculer le résultat de chaque tâche. Cependant, comme les participants peuvent se connecter et se déconnecter, celles-ci ne sont pas toujours disponibles. Cette disponibilité est définie par un ensemble d'intervalles. Si une ressource quitte le système lors d'un calcul, alors elle reprend le calcul là où elle l'avait laissé à son retour.
- Le serveur assigne une tâche à un ensemble de ressources et récupère ensuite les résultats. Pour chaque tâche, nous supposons qu'il n'y a qu'un seul résultat correct. D'autre part, l'espace des résultats est supposé suffisamment large pour que la probabilité que deux ressources renvoient le même résultat incorrect soit insignifiante (on supposera alors qu'il s'agit d'un type de fautes spécifique que nous détaillons dans la section 5.3.2).

5.3.2 Modèle de menace

Une ressource calculatoire commet une *faute individuelle* si elle renvoie un résultat incorrect indépendamment des autres ressources. Nous dirons alors que la ressource *échoue individuellement*. Cela peut se produire dans le cas d'une erreur d'entrée/sortie, d'une interférence cosmique ou encore d'une corruption mémoire.

Nous désignons par *faute collective* la génération du même résultat incorrect par deux ressources calculatoires distinctes suite aux calculs de la même tâche. Cette faute concerne alors les ressources ayant envoyé les résultats incorrects en question. Nous dirons aussi que les ressources *échouent collectivement*. On distingue deux types de fautes collectives :

- Il s'agit du premier lorsque des saboteurs coopèrent volontairement pour envoyer un même résultat incorrect pour une tâche donnée dans l'objectif de battre un algorithme par quorum. Dans ce cas, il y a *collusion* en les participants.
- Le second type de faute concerne des participants qui ne trichent pas délibérément comme dans le cas d'une infection par un virus ou lorsqu'il existe une erreur dans le code exécuté qui ne se révèle que sur certaines ressources.

Les ressources sont regroupées suivant leurs comportement de telle sorte que les ressources d'un même groupe renvoient le même résultat lorsqu'elles exécutent la même tâche (sauf s'il y a une faute individuelle). Il existe deux types de groupes :

Groupe non-fautif. Les ressources de cet unique groupe n'interviennent dans aucune faute collective mais peuvent échouer individuellement. Ce groupe est donc non-fautif en terme de faute collective.

Groupe fautif. Dans le cas de la collusion, les participants peuvent décider de renvoyer le résultat correct de temps à autre afin de réduire le risque d'être détectés. Les ressources d'un même groupe échouent donc collectivement avec une probabilité donnée.

Par soucis de concision, nous dirons qu'un groupe fautif *échoue* pour une tâche spécifique lorsque les ressources de ce groupe échouent collectivement. Si un autre groupe fautif échoue pour la même tâche, alors il y a deux cas :

- Les ressources des deux groupes renvoient le même résultat incorrect, auquel cas les deux groupes fautifs *échouent collectivement*. Ce comportement se produit si certaines ressources sont infectées par plusieurs virus qui agissent chacun différemment.
- Les deux résultats incorrects des deux groupes sont différents et nous dirons que deux groupes *échouent individuellement*. C'est la seule situation envisageable s'il n'y a qu'un seul groupe fautif. La probabilité qu'un groupe échoue collectivement avec un autre est inférieure à celle qu'il échoue individuellement.

Une ressource d'un groupe donné peut également échouer individuellement (une faute individuelle est prédominante sur une faute collective) et renvoyer un résultat incorrect qu'elle aura été la

seule a générer.

Soulignons que le modèle de menace proposé est fort : les groupes fautifs peuvent échouer collectivement (coopération entre les groupes par exemple), les groupes fautifs n'échouent pas systématiquement (ce qui peut permettre aux participants faisant preuve de collusion d'éviter d'être détectés), les ressources ne requièrent pas de synchronisation pour échouer collectivement et aucune de ces informations n'est connue par le système à priori.

Nous postulons en revanche que le groupe non-fautif est le groupe le plus large. Cela implique qu'il existe une majorité relative de ressources non-fautives.

5.3.3 Données du problème

Le problème que nous étudions porte sur la caractérisation des propriétés liées aux fautes collectives de chaque ressource calculatoire. Précisément, nous estimons le groupe auquel appartient chaque ressource, ainsi que les probabilités que chaque paire de groupes échoue collectivement. Aucune supposition n'est réalisée à propos de la façon dont sont assignées les tâches aux ressources sinon qu'elles sont dupliquées. Bien que la façon dont les tâches sont ordonnancées peut certainement aider à la détection et la caractérisation des fautes collectives, nous faisons des suppositions minimales sur l'étape d'ordonnancement. En effet, nous proposons une solution générique indépendamment de l'ordonnanceur utilisé. De nombreux systèmes (comme BOINC) reposent sur un mécanisme de redondance pour mieux résister aux pannes. L'hypothèse de duplication des tâches n'est donc pas une hypothèse forte. L'entrée de notre problème est une succession d'événements de deux types possibles :

- $< d, p, t, r >$à l'instant d, la ressource p envoie le résultat r pour la tâche t.
- $< d, t >$ à l'instant d, toutes les ressources assignées à la tâche t ont terminé leurs calculs et la tâche t est alors terminée.

Nous supposons que ces événements arrivent dans un ordre déterminé par la vitesse des participants.

5.3.4 Critères

Il existe un lien entre la caractérisation de la fiabilité des ressources et les systèmes de réputation. Jøsang et al. [92] ont proposé certains critères génériques pour évaluer un système de réputation qui peuvent se traduire ainsi dans notre contexte de caractérisation des fautes collectives :

Précis Capacité du système à donner une estimation des probabilités de faute collective proche de la réalité.

Robuste Capacité du système à observer une attaque tout en soutenant ses performances.

Introspectif Le système doit être en mesure de donner une indication sur la confiance que l'on peut accorder aux estimations produites.

Adaptatif Capacité à détecter (à la fois quantitativement et qualitativement) les changements subis par la plateforme.

Régulier Les changements entre les états successifs du système ne doivent pas être radicaux.

Nous ne donnerons pas la même importance à tous ces critères. Nous jugeons la précision comme le plus important et il sera par la suite le centre de notre attention. L'idée principale de notre mécanisme est de résister à des attaques et nous considérons donc que la robustesse de notre solution est implicitement assurée. Bien que l'introspection soit difficilement évaluable, nous donnerons un aperçu des performances obtenues à cet égard. Un système adaptatif peut facilement être obtenu à partir d'un système conçu pour caractériser une plateforme statique en insérant

un mécanisme d'amnésie. Il existe de nombreux cas d'étude où ce critère est important (comme la propagation de virus), mais nous limiterons notre étude au cas statique. Finalement, nous pensons que la régularité n'est pas une qualité importante pour un système de caractérisation et nous ignorons cet aspect par la suite.

5.3.5 Groupes observés et mesures

Pour modéliser la réalité, nous groupons les ressources qui partagent le même comportement. La sortie de notre système de caractérisation est un ensemble de groupes de ressources qui possèdent les même caractéristiques de faute collective. Ces groupes sont dits *observés* (i.e., lorsqu'une tâche est assignée à un sous-ensemble de ressources dans ce même groupe, le système a observé que les ressources échouaient de la même façon). Pour mesurer la précision de notre solution, nous comparons l'estimation avec les probabilités de faute collective réelles.

Soit $\mathcal{G}$ l'ensemble réel des groupes et $\hat{\mathcal{G}}$ l'estimation de l'ensemble $\mathcal{G}$ par notre système. Nous supposons que ces deux ensembles sont complets (chaque ressource est dans exactement un groupe). Soit $\hat{g}(p) \in \hat{\mathcal{G}}$ l'ensemble des ressources qui sont dans le même groupe observé que la ressource p. Nous voulons que les groupes observés $\hat{\mathcal{G}} = \bigcup_{p \in P} \{\hat{g}(p)\}$ concordent avec les groupes réels et que les probabilités de faute collective observées soient proches des véritables probabilités. Combiner ces critères qualitatif et quantitatif en une seule mesure précise est délicat. Soit $g \in \mathcal{G}$ et $g' \in \mathcal{G}$ deux groupes réels de ressources. Soit $e_{g \cup g'}$ la différence absolue entre la probabilité réelle et celle estimée que les groupes g et g' échouent collectivement. Nous agrégeons toutes les erreurs associées à chaque estimation en utilisant le RMSD (Root Mean Square Deviation).

$$\frac{1}{|\mathcal{G}|} \sqrt{\sum_{(g,g') \in \mathcal{G}^2} e^2_{g \cup g'}} \tag{5.3.1}$$

Une faible valeur de RMSD indique une bonne estimation des groupes. La façon de calculer $e_{g \cup g'}$ dépend de la représentation interne des interactions entre les groupes observés. Nous proposons deux représentations possibles et montrons comment borner $e_{g \cup g'}$ dans la section 5.4.1.4. Nous considérons par la suite ces deux aspects de la précision :

Temps de convergence Temps nécessaire pour obtenir une précision acceptable (définie par un seuil sur le RMSD).

Précision stabilisée Précision atteinte après un large nombre d'événements (médiane du RMSD pendant les 100 derniers événements d'une simulation).

5.4 Caractérisation des fautes collectives

La caractérisation et la représentation des fautes collectives nécessitent plusieurs éléments. Nous présentons d'abord une façon de considérer les interactions entre groupes de ressources. L'objectif est d'identifier efficacement et de mettre à jour des groupes observés. Nous présentons ensuite les algorithmes de groupement qui se basent sur deux représentations (faute et affinité) qui ont chacune des propriétés intéressantes. Ces algorithmes dynamiques fournissent de nouvelles estimations à chaque nouvel événement.

Les algorithmes présentés groupent les ressources qui partagent les mêmes caractéristiques en terme de faute collective. Les événements ne sont pas traités comme des observations sur les interactions entre des ressources mais entre des groupes de ressources. Cela permet de réduire la complexité de l'approche : au lieu de considérer chaque paire de ressources, on ne considère que chaque paire de groupes, factorisant ainsi la représentation interne. Comme nous travaillons au

niveau des groupes, certaines observations sont alors redondantes. Dans ces cas, les algorithmes ne mettront à jour leurs représentations internes qu'une seule fois. Par exemple, si toutes les ressources d'un même groupe envoient le même résultat pour une tâche donnée, une seule entrée sera mise à jour entre le groupe concerné et les autres. La première fois qu'une interaction entre des groupes est observée, nous la qualifierons de *nouvelle*.

L'objectif des algorithmes proposés en section 5.4.2 est de maintenir une structure de données qui représente les interactions entre les groupes de ressources calculatoires. Initialement, chaque structure de données ne correspond pas à la réalité et nous proposons des opérations qui visent à modifier ces structures jusqu'à une stabilisation autour d'un état donné. On dira qu'un algorithme *converge* si la structure de données utilisée se stabilise, indépendamment de la précision obtenue.

Avant de présenter les algorithmes en détail, nous décrivons la façon dont nous représentons les interactions entre les groupes de ressources calculatoires.

5.4.1 Modèle d'interaction

Dans cette section, nous proposons deux représentations possibles pour caractériser les interactions entre les groupes de ressources. La première est basée sur les fautes collectives constatées entre les ressources, la seconde se concentre sur leurs affinités compte tenu des résultats produits par les ressources.

5.4.1.1 Représentation par faute

La première représentation possible pour modéliser les interactions entre groupes de ressources consiste à se baser sur la probabilités que deux ressources réalisent des fautes collectives. Une matrice carrée $\hat{C}$ contient les estimations des probabilités de faute collective. La probabilité estimée que les ressources des groupes observés i and j échouent collectivement est notée $\hat{c}_{ij}$. Cette représentation possède un désavantage : quand deux ressources renvoient le même résultat, un mécanisme de certification est nécessaire pour déterminer si le résultat est incorrect, et donc s'il s'agit d'une faute collective. Puisque ce mécanisme se base sur les précédentes estimations du mécanisme de caractérisation, il y a un risque que le système amplifie les erreurs d'estimation et ne converge pas. Nous proposons une technique d'*adaptation* dans la section 5.4.2.1 qui permet de remédier à ce problème.

Soit L un sous-ensemble de ressources calculatoires ($L \subseteq P$). Soit E_L l'événement « toutes les ressources appartenant à L échouent collectivement pour une tâche donnée ». Les deux lemmes suivant sont utilisés pour produire des estimations sur les fautes collectives entre n'importe quel sous-ensemble de ressources L.

Lemme 5.1. *Soit K_L l'ensemble des groupes contenant toutes les ressources appartenant à un ensemble L (i.e., $K_L = \bigcup_{p \in L} \{g(p)\}$), alors :*

$$0 \leq \Pr[E_L] \leq \min_{(i,j) \in K_L^2} (c_{ij})$$

Démonstration. Ces bornes sont obtenues directement à partir des définitions : une probabilité est positive ou nulle ; la probabilité que toutes les ressources appartenant à L échouent collectivement ne peut pas excéder les probabilités de faute collective de chaque paire de groupes de ressources. □

Lemme 5.2. *Soit l'ensemble $L = i \cup j \cup k$ tel que $(i, j, k) \in \mathcal{G}^3$. Alors :*

$$\max\left(0, \frac{c_{ij} + c_{ik} + c_{jk} - 1}{2}\right) \leq \Pr[E_L]$$

Démonstration. Par définition, les groupes i et j échouent collectivement avec k, ou bien ils échouent collectivement sans k : $c_{ij} = \Pr[E_L] + \Pr[E_{i\cup j} \cap \overline{E_L}]$. Par analogie, $c_{ik} = \Pr[E_L] + \Pr[E_{i\cup k} \cap \overline{E_L}]$ et $c_{jk} = \Pr[E_L] + \Pr[E_{j\cup k} \cap \overline{E_L}]$. Nous avons également $\Pr[E_{i\cup j} \cap \overline{E_L}] + \Pr[E_{i\cup k} \cap \overline{E_L}] + \Pr[E_{j\cup k} \cap \overline{E_L}] + \Pr[E_L] \leq 1$ car les événements sont disjoints. Ainsi :

$$\frac{c_{ij} + c_{ik} + c_{jk} - 1}{2} \leq \Pr[E_L]$$

□

Le scénario qui éloigne le plus les bornes du lemme 5.2 est obtenu lorsque $c_{ij} = c_{ik} = c_{jk} = \frac{1}{3}$. Dans ce cas, l'intervalle entre les deux bornes est $\frac{1}{3}$.

5.4.1.2 Représentation par affinité

La seconde représentation est basée sur des observations absolues, i.e., les affinités entre les ressources de différents groupes relativement aux résultats renvoyés. La matrice carré $\hat{A}$ contient les estimations de probabilités que les groupes observés retournent un résultat similaire. La probabilité que les ressources du groupe i calculent le même résultat (correct ou non) que les ressources du groupe j est notée $\hat{a}_{ij}$. Avec cette représentation neutre, il n'y a pas besoin d'un mécanisme de certification externe pour déterminer si les résultats sont corrects ou non. Le système a donc plus de chance de converger.

Les deux lemmes suivant fournissent des bornes pour estimer la probabilité que les ressources appartenant à L échouent collectivement et retournent le même résultat incorrect (i.e., $\Pr[E_L]$).

Le groupe contenant le plus grand nombre de ressources est indicé à 1 et nous supposons qu'il s'agit du groupe non-fautif.

Lemme 5.3. *Soit $L = i \cup j$ tel que $(i, j) \in \mathcal{G}^2$. Alors :*

$$\max(0, a_{ij} - a_{1i}, a_{ij} - a_{1j}) \leq \Pr[E_L]$$

$$\Pr[E_L] \leq \min\left(a_{ij}, \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2}\right)$$

Démonstration. Nous séparons l'univers des possibles en plusieurs événements disjoints :

$\overline{E_i} \cap \overline{E_j}$	aucun des groupes i et j n'échoue
$\overline{E_i} \cap E_j$	seul le groupe j échoue
$E_i \cap \overline{E_j}$	seul le groupe i échoue
$\overline{E_L} \cap E_i \cap E_j$	les groupes i et j échouent individuellement
E_L	les groupes i et j échouent collectivement

Par construction, $a_{1i} = \Pr[\overline{E_i} \cap \overline{E_j}] + \Pr[\overline{E_i} \cap E_j]$, $a_{1j} = \Pr[\overline{E_i} \cap \overline{E_j}] + \Pr[E_i \cap \overline{E_j}]$ et $a_{ij} = \Pr[\overline{E_i} \cap \overline{E_j}] + \Pr[E_L]$. Comme l'ensemble de l'univers est couvert par ces événements, la somme de leurs probabilités est égale à un. Ainsi :

$$\Pr[E_L] = \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2} - \frac{\Pr[\overline{E_L} \cap E_i \cap E_j]}{2}$$

On a aussi $\Pr[\overline{E_L} \cap E_i \cap E_j] \leq 1 - a_{1i}$ car le groupe i échoue moins souvent qu'il s'accorde avec le groupe le plus large (supposé être le groupe non-fautif). Par analogie, $\Pr[\overline{E_L} \cap E_i \cap E_j] \leq 1 - a_{1j}$ et la borne inférieure peut en être déduite. La borne supérieure s'obtient en notant qu'une probabilité est positive ou nulle et que les probabilités de faute collective entre les groupes i et j ne peut excéder leurs probabilités d'affinité. □

L'intervalle le plus large pour les bornes données dans le lemme 5.3 est $\frac{1}{3}$ lorsque $a_{ij} = a_{1i} = a_{1j} = \frac{1}{3}$.

Lemme 5.4. *Soit K_L l'ensemble des groupes contenant toutes les ressources appartenant à un ensemble L, alors :*

$$0 \leq \Pr[E_L] \leq \min_{(i,j)\in K_L^2} \left(a_{ij}, \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2} \right)$$

Démonstration. Il s'agit d'une généralisation directe du lemme 5.3. □

5.4.1.3 Relations entre les deux représentations

Nous présentons désormais quelques relations mathématiques entre ces deux représentations.

Théorème 5.5. *Soit $C = (c_{ij})$ et $A = (a_{ij})$ les matrices exactes de faute et d'affinité. Le groupe le plus large a pour indice 1 et est supposé non-fautif. Il est possible de borner A à partir de C et C à partir de A avec les équations suivantes :*

$$a_{ij} \leq 1 + 2 \times c_{ij} - c_{ii} - c_{jj}$$

$$c_{ij} \leq \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2}$$

Démonstration. Les groupes i et j renvoient des résultats similaires en cas de succès ou s'ils échouent collectivement. Ainsi :

$$a_{ij} = \Pr[E_{i\cup j} \cup \overline{E_i \cup E_j}]$$

$$a_{ij} = \Pr[E_{i\cup j}] + 1 - \Pr[E_i \cup E_j]$$

$$a_{ij} = c_{ij} + 1 - \Pr[E_i] - \Pr[E_j] + \Pr[E_i \cap E_j]$$

$$a_{ij} = c_{ij} + 1 - c_{ii} - c_{jj} + \Pr[E_{i\cup j}] + \Pr[\overline{E_{i\cup j}} \cap E_i \cap E_j]$$

Comme il faut considérer le cas où les groupes i et j échouent individuellement, nous obtenons la borne suivante :

$$a_{ij} \leq 2 \times c_{ij} + 1 - c_{ii} - c_{jj}$$

Borner la matrice C à partir de la matrice A est une application du lemme 5.4 :

$$c_{ij} \leq \frac{1 + a_{ij} - a_{1i} - a_{1j}}{2}$$

□

5.4.1.4 Évaluation des erreurs dans le RMSD

À partir de ces résultats, nous pouvons désormais développer l'équation 5.3.1 qui définit le RMSD :

$$\frac{1}{|\mathcal{G}|}\sqrt{\sum_{(g,g')\in\mathcal{G}^2} e^2_{g\cup g'}}$$

Nous utilisons la borne supérieure pour estimer la probabilité de faute collective entre le groupe g et le groupe g', et donc pour calculer $e_{g\cup g'}$. Nous avons ainsi

$$e_{g\cup g'} = |c_{gg'} - \min_{(i,j)\in\hat{K}^2_{g\cup g'}} (\hat{c}_{ij})|$$

avec la première représentation et

$$e_{g\cup g'} = \left| c_{gg'} - \min_{(i,j)\in\hat{K}^2_{g\cup g'}} \left(\hat{a}_{ij}, \frac{1+\hat{a}_{ij}-\hat{a}_{1i}-\hat{a}_{1j}}{2}\right)\right|$$

avec la seconde.

5.4.1.5 Lois de probabilité bêtas

Suivant la représentation choisie (faute ou affinité), chaque interaction donne lieu a une observation. Il s'agit d'une observation soit positive (faute collective ou affinité), soit négative (pas de faute collective ou différence entre les résultats générés). Nous cherchons donc à estimer une valeur de la matrice C (probabilité de faute) ou de la matrice A (probabilité d'affinité) à partir d'une succession d'observations. L'estimation correspond à un élément de la matrice $\hat{C}$ ($\hat{c}_{ij}$) ou de la matrice $\hat{A}$ ($\hat{a}_{ij}$). Quelque soit la représentation, on désigne par X le nombre d'observations positives sur une succession de n observations. Comme nous supposons que le système est stationnaire, les probabilités de faute et d'affinité sont constantes. La variable aléatoire X suit donc une loi binomiale $\mathcal{B}(n;p)$. Le problème consiste à déterminer la probabilité p de cette loi à partir d'une réalisation de X.

L'inférence bayésienne permet d'utiliser les observations pour inférer la probabilité d'une hypothèse. Le paramètre p est alors considérée comme la réalisation d'une variable aléatoire π. Nous définissons deux distributions de probabilités liées à π :

- La loi de probabilité que suit π *à priori* est caractérisée par la fonction $f_\pi(p)$.
- La loi que suit π *à posteriori* est conditionnée par le nombre d'observations positives et est défini par la fonction $f_\pi(p|X=x)$.

Si toutes les valeurs de l'intervalle $[0;1]$ sont équiprobables, alors π suit *à priori* une loi uniforme dans l'intervalle $[0;1]$ et π suit *à posteriori* une loi bêta de paramètre $\alpha = x+1$ et $\beta = n-x+1$ où x est une réalisation de X. Nous rappelons ce résultat connu par Bayes [21] (voir [129, Section 13.7.3] pour une description moderne). La formule de Bayes donne la fonction de densité $f_\pi(p|X=x)$ de la variable aléatoire π sachant que X prend pour valeur x :

$$f_\pi(p|X=x) = \frac{\Pr[X=x|\pi=p]\times f_\pi(p)}{\Pr[X=x]} = \frac{\Pr[X=x|\pi=p]\times f_\pi(p)}{\int_0^1 \Pr[X=x|\pi=y]\times f_\pi(y)dy}$$

Pour une valeur donnée de p, la probabilité d'une distribution binomiale est $\Pr[X=x|\pi=p] = C_n^x p^x (1-p)^{n-x}$. D'autre part, nous supposons que π suit *à priori* une loi uniforme. On obtient

donc la fonction de densité suivante :

$$f_\pi(p|X=x) = \frac{C_n^x p^x (1-p)^{n-x}}{\frac{1}{n+1}} = \frac{\Gamma(n+2)}{\Gamma(x+1)\Gamma(n-x+1)} p^x (1-p)^{n-x}$$

Ce qui est la densité d'une loi bêta de paramètre $\alpha = x+1$ et $\beta = n-x+1$ dont l'espérance est $\frac{x+1}{n+2}$.

Ce résultat signifie qu'au départ, c'est-à-dire avant toute observation, $\alpha = \beta = 1$. À chaque nouvelle observation positive (resp., négative), le paramètre α (resp., β) est incrémenté. L'estimation de p est fournie par l'espérance de π, à savoir $\frac{x+1}{n+2}$. Considérer que π suit *à posteriori* une loi bêta (c'est-à-dire sachant $X = x$) permet de produire un intervalle de confiance sur les valeurs probables de p et donc d'avoir une idée de la précision de l'estimation. Il s'agit d'ailleurs de l'intérêt premier de l'inférence bayésienne. Notons que plus le nombre d'observations augmente, plus $\alpha+\beta$ augmente et l'intervalle de confiance diminue (la loi *à posteriori* de π est alors de plus en plus concentrée).

Nous considérons par la suite que chaque valeur $\hat{c}_{ij}$ et $\hat{a}_{ij}$ est associée à une loi bêta qui est mise à jour à chaque nouvelle observation. Cette approche est similaire à celle présentée dans [91] où des lois bêtas sont utilisées dans un système de réputation.

Des opérations arithmétiques (additions et maximums) sont réalisées sur les éléments des matrices $\hat{C}$ et $\hat{A}$. C'est donc sur les lois bêtas internes que se réalisent ces opérations. Quelque soit l'opération arithmétique, le problème se réduit à trouver les paramètres α et β tels que la loi bêta résultante possède une moyenne μ et une variance σ^2 données (voir chapitre 1). Comme les résultats existants se montrent insuffisants à cet égard, nous proposons des développements arithmétiques spécifiques à notre problème.

Rappelons que les couples $(\mu, \sigma^2) \in [0;1]\times\mathbb{R}^+$ ne sont pas tous admissibles pour une loi définie sur le support $[0;1]$. Pour une loi bêta, l'inégalité $\sigma^2 < \mu(1-\mu)$ est forcément vérifiée. Cette contrainte s'applique d'ailleurs à toute loi définie sur le support $[0;1]$. Dans notre cas, cependant, nous souhaitons que $\alpha \geq 1$ et $\beta \geq 1$ pour des raisons de cohérence (une valeur inférieure à 1 pour l'un de ces deux paramètres signifierait un nombre d'observations négatif). Dans ce cas, les contraintes d'admissibilité sont plus fortes : $\sigma^2 < \mu(1-\mu)\frac{\mu}{1+\mu}$ et $\sigma^2 < \mu(1-\mu)\frac{1-\mu}{2-\mu}$.

Si les valeurs μ et σ^2 respectent ces deux conditions, alors la loi bêta recherchée a pour paramètres $\alpha = \mu\left(\frac{\mu(1-\mu)}{\sigma^2} - 1\right)$ et $\beta = (1-\mu)\left(\frac{\mu(1-\mu)}{\sigma^2} - 1\right)$. Ces égalités résultent de dérivations triviales à partir de la définition de la loi bêta.

Sinon, il n'est pas possible de produire de loi bêta respectant toutes les contraintes. Nous donnons la priorité à la variance et ajustons la moyenne sur sa valeur admissible la plus proche. Cela minimise la perte de précision en terme d'introspection. Dans le cas où l'espérance est trop grande (i.e., proche de 1), nous fixons le paramètre β à sa plus petite valeur, (i.e., à 1) et cherchons la valeur de α telle que la variance soit σ^2 et telle que l'espérance soit maximale. À l'aide d'un logiciel de calcul formel, nous aboutissons à

$$\alpha = \frac{2}{3}\sqrt{\frac{\sigma^2+3}{\sigma^2}}\,\sin\left(\frac{\arctan\left(\frac{3}{(\sigma^2+18)}\sqrt{\frac{3\,(1-\sigma^4-11\,\sigma^2)}{\sigma^2}}\right)}{3} + \frac{\pi}{6}\right) - \frac{4}{3}$$

Dans le cas où l'espérance est trop petite, on obtient des équations analogues en inversant les valeurs des deux paramètres. Ces formules assurent d'obtenir une loi bêta dont la variance est égale à celle recherchée avec des valeurs de paramètres supérieures à 1.

Enfin, il se peut que la variance soit trop grande (il est impossible que la variance d'une loi bêta soit supérieure à $\frac{1}{12}$) auquel cas la loi bêta est réinitialisée, i.e., $\alpha = \beta = 1$.

5.4.2 Algorithme dynamique

Dans la section 5.4.1, nous avons décrit deux structures de données. Chacune donne lieu à un algorithme différent. Nous décrivons désormais la façon dont les groupes sont construits et mis à jour. Deux opérations sont pour cela nécessaires : la *fusion* des groupes de ressources et la *séparation* des ressources. Initialement, chaque ressource est mise dans un groupe singleton. Après quelques événements, les observations peuvent être suffisantes pour déterminer si deux groupes observés se comportent pareillement. Le cas échéant, les deux groupes sont *fusionnés*. Comme les conditions pour fusionner les groupes de ressources sont de nature statistique, des fusions inappropriées peuvent avoir lieu. Dans ce cas, une ressource sera *séparée* d'un groupe observé donné s'il s'avère qu'elle agit différemment du groupe en question. L'objectif est de créer une correspondance entre les groupes observés et les groupes réels. Les conditions de fusion et de séparation sont différentes pour chaque représentation (faute ou affinité). Notons que ces opérations modifient la taille de la matrice considérée ($\hat{C}$ ou $\hat{A}$) et influent donc le coût algorithmique de chaque manipulation des structures de données.

5.4.2.1 Groupement par faute

Nous proposons un algorithme reposant sur deux procédures et qui utilise la représentation par faute présentée dans la section 5.4.1.1. Tous les événements sont considérés de façon chronologique et le type d'événement détermine la procédure à utiliser : à chaque événement correspondant à la réception d'un résultat ($< d, p, t, r >$), l'algorithme 5.1 est utilisé ; lorsque l'événement correspond à la terminaison d'une tâche ($< d, t >$), c'est l'algorithme 5.2 qui est utilisé. Dans ces deux procédures, nous notons $P_{r,t}$ l'ensemble des ressources calculatoires qui ont renvoyé le résultat r pour la tâche t.

Nous décrivons désormais le déroulement de l'algorithme 5.1. À chaque fois qu'un résultat r est calculé par une ressource p pour la tâche t, nous considérons d'abord toutes les ressources qui ont obtenu un résultat r' différent de celui-ci pour la même tâche (ligne 1). Si le résultat r' n'a été généré que par une seule ressource, alors il est ignoré (on considère qu'on ne peut pas déterminer s'il s'agit d'une faute individuelle, collective, ou bien même s'il s'agit du résultat correct). Dans le cas contraire, on peut considérer l'interaction entre le groupe observé contenant p', $\hat{g}(p')$, et celui contenant p, $\hat{g}(p)$ (sauf si cette interaction n'est pas *nouvelle*, c'est-à-dire si elle a déjà été prise en compte). Ces deux conditions sont regroupées à la ligne 2. Si l'interaction est considérée, il existe deux possibilités : soit les ressources p' et p font partie du même groupe observé, auquel cas elles sont toutes les deux séparées de ce groupe (ligne 4) car elles ont produit des résultats différents ; sinon, l'estimation de la probabilité de faute collective entre $g(p')$ et $g(p)$ doit être diminuée (incrémentation de la valeur β de la distribution bêta correspondante), ce qui a lieu à la ligne 6.

Le deuxième type d'événement ($< d, t >$) correspond à la terminaison d'une tâche t. La première action consiste à invoquer un mécanisme de certification externe pour estimer lequel des résultats obtenus pour la tâche t est correct (ligne 1). Lorsque l'un des résultat est certifié, il est alors considéré comme correct. Notre algorithme ne dépend pas d'un mécanisme spécifique. Il est cependant nécessaire qu'il soit suffisamment pertinent pour assurer une convergence de notre structure de donnée. Tous les résultats sont ensuite considérés (ligne 2). Si un résultat est orphelin (une seule ressource l'a généré), alors on l'ignore comme dans l'algorithme 5.1. Sinon, toutes les paires de ressources ayant produit ce résultat sont considérées (ligne 6). Si l'interaction entre les groupes observés $\hat{g}(p')$ et $\hat{g}(p)$ n'a pas encore été prise en compte (ligne 7), alors nous augmentons ou diminuons l'estimation de la probabilité de faute collective entre ces deux groupes selon que le résultat r est certifié ou non (ligne 9 et 11). Enfin, si les deux groupes observés $\hat{g}(p')$

Algorithme 5.1 Mise à jour de la structure par faute lors de la réception d'un résultat (événement de type $< d, p, t, r >$)

1: **pour chaque** $p' \in P_{r',t}, r' \neq r$ **faire**
2: **si** $|P_{r',t}| \neq 1$ et l'interaction entre $\hat{g}(p')$ et $\hat{g}(p)$ est *nouvelle* pour t **alors**
3: **si** $\hat{g}(p') = \hat{g}(p)$ **alors**
4: séparer p et p' de $\hat{g}(p')$
5: **sinon**
6: incrémenter le paramètre β correspondant à l'estimation $\hat{c}_{\hat{g}(p')\hat{g}(p)}$ {diminue la probabilité de faute collective entre $\hat{g}(p')$ et $\hat{g}(p)$}
7: **fin si**
8: **fin si**
9: **fin pour**

et $\hat{g}(p)$ sont distincts mais présentent des affinités, alors ils sont fusionnés (ligne 13 et 14). À chaque fois qu'une estimation d'une probabilité de faute collective est modifiée (ligne 9 et 11), l'algorithme utilise une distribution bêta comme cela est décrit dans la section 5.4.1.

Nous postulons que le groupe le plus large est non-fautif et donc la probabilité que ce groupe échoue est nulle. Cependant, lors d'une fusion ou d'une séparation, il se peut que le groupe observé le plus large change et la probabilité de faute collective du nouveau groupe observé dominant n'est alors pas nécessairement nulle. Nous avons conçu une technique d'*adaptation* qui ajuste les estimations de façon à les rendre cohérentes. L'adaptation modifie la matrice $\hat{C}$ en procédant en deux étapes : la matrice est d'abord convertie en une matrice d'affinité ; la nouvelle matrice $\hat{C}$ est alors obtenue à partir de cette matrice d'affinité. Les conversions se font en utilisant les bornes fournies par le théorème 5.5. L'égalité suivante est finalement utilisée pour obtenir une nouvelle matrice de faute $\hat{C}' = (\hat{c}'_{ij})$ à partir de la matrice $\hat{C} = (\hat{c}_{ij})$ (où 1 est l'indice du nouveau groupe observé le plus large) :

$$\hat{c}'_{ij} = \hat{c}_{ij} - \hat{c}_{1i} - \hat{c}_{1j} + \hat{c}_{11}$$

Les opérations arithmétiques (additions et soustractions) sont en fait réalisées sur les distributions bêtas correspondant aux estimations des probabilités de faute collective. Comme les additions sur les variables aléatoires propagent les écart-types, les distributions produites sont plus dispersées et l'on perd en précision. Il est donc préférable que le système n'aboutisse pas en une configuration où l'adaptation devient nécessaire. Cela peut être obtenu par exemple en évitant de faire des fusions inappropriées qui pourraient changer le groupe le plus large. Cependant, il est aussi souhaitable que les fusions se produisent facilement (surtout à l'initialisation) afin de favoriser la convergence du système. Au final, nous avons à réaliser un compromis entre le risque d'adapter le système et la facilité avec laquelle les fusions se produisent (chaque aspect ayant une influence sur la vitesse de convergence).

Les conditions de fusion sont donc critiques pour le succès de l'algorithme. L'idée principale est de favoriser les fusions au début au risque d'avoir quelques groupes observés qui ne correspondent pas à la réalité. Cela réduit ainsi rapidement la complexité algorithmique de l'approche en réduisant le nombre de groupes observés à considérer. Par la suite, les opérations sur la structure doivent nécessiter plus de certitude. Deux groupes doivent être fusionnés s'ils échouent collectivement avec la même probabilité que celle avec laquelle ils échouent individuellement. Avant de décrire la condition précise de fusion entre les groupes i et j, certaines notations sont nécessaires. Soit $\hat{c}_{ii}$, $\hat{c}_{ij}$ et $\hat{c}_{jj}$ les estimations des probabilités de faute collective. $\#\hat{c}_{ij}$ est le nombre d'observations qui a permis d'aboutir à l'estimation $\hat{c}_{ij}$ et $I(\hat{c}_{ij})$ est son intervalle de

Algorithme 5.2 Mise à jour de la structure par faute lors de la terminaison d'une tâche (événement de type $< d, t >$)

```
1: invoque un mécanisme de certification pour les résultats de la tâche t
2: pour chaque résultat r renvoyé pour la tâche t faire
3:   si |P_{r,t}| = 1 alors
4:     l'observation est ignorée
5:   sinon
6:     pour chaque (p', p) ∈ P^2_{r,t} faire
7:       si l'interaction entre ĝ(p') et ĝ(p) est nouvelle pour t alors
8:         si r est le résultat certifié alors
9:           incrémenter le paramètre β correspondant à l'estimation ĉ_{ĝ(p')ĝ(p)}   {diminue la
             probabilité de faute collective entre ĝ(p') et ĝ(p)}
10:        sinon
11:          incrémenter le paramètre α correspondant à l'estimation ĉ_{ĝ(p')ĝ(p)}   {augmente la
             probabilité de faute collective entre ĝ(p') et ĝ(p)}
12:        fin si
13:        si ĝ(p') ≠ ĝ(p) et ĝ(p') est similaire à ĝ(p) alors
14:          fusionner ĝ(p') et ĝ(p)
15:        fin si
16:      fin si
17:    fin pour
18:  fin si
19: fin pour
```

confiance avec un degré de confiance de 95%. Soit $\hat{l}$ le nombre de groupes observés. Les groupes observés i et j sont fusionnés si :

$$|\hat{c}_{ii} - \hat{c}_{ij}| < \frac{I(\hat{c}_{ii}) + I(\hat{c}_{ij})}{2} \wedge |\hat{c}_{ii} - \hat{c}_{jj}| < \frac{I(\hat{c}_{ii}) + I(\hat{c}_{jj})}{2} \wedge |\hat{c}_{ij} - \hat{c}_{jj}| < \frac{I(\hat{c}_{ij}) + I(\hat{c}_{jj})}{2} \quad (5.4.1)$$

et si :

$$\min(\#\hat{c}_{ii}, \#\hat{c}_{ij}, \#\hat{c}_{jj}) > \frac{\gamma \times \max(|i \cup j|, \frac{n}{\hat{l}})}{(1 - |\hat{c}_{ii} - \hat{c}_{ij}|)(1 - |\hat{c}_{ii} - \hat{c}_{jj}|)(1 - |\hat{c}_{ij} - \hat{c}_{jj}|)} \quad (5.4.2)$$

où $\gamma = \begin{cases} 2,5 & \text{si le groupe observé le plus large change par cette fusion} \\ 1 & \text{sinon} \end{cases}$

La condition spécifiée dans l'équation 5.4.1 vérifie que les estimations $\hat{c}_{ii}$, $\hat{c}_{ij}$ et $\hat{c}_{jj}$ sont cohérentes. La seconde condition (équation 5.4.2) impose un nombre minimal d'observations qui est gouverné par quatre principes. Le paramètre γ, dont les valeurs sont fixées empiriquement, rend la fusion plus exigeante si le groupe observé le plus large est sur le point de changer (cela induirait alors une adaptation qui introduirait de l'incertitude dans les estimations). La taille $|i \cup j|$ spécifie que fusionner des groupes doit nécessiter d'autant plus d'observations que le groupe résultant est large. Ainsi, des groupes observés de tailles modérées apparaîtront rapidement. Toutefois, il n'est pas utile de fusionner des petits groupes facilement s'il existe déjà des groupes larges (c'est le sens du paramètre $\frac{n}{\hat{l}}$). Enfin, le dénominateur prend en considération les écarts numériques entre les estimations.

La valeur $2,5$ du paramètre γ est empirique. Elle doit être supérieure à 2 pour réduire le risque d'adapter la matrice. Toutefois, des valeurs supérieures à 3 handicapent le système et retardent

Algorithme 5.3 Mise à jour de la structure par affinité lors de la réception d'un résultat (événement de type $< d, p, t, r >$)

1: **si** $|P_{r,t}| \neq 1$ **alors**
2: **pour chaque** $p' \in P_{r,t}$ **faire**
3: **si** l'interaction entre $\hat{g}(p')$ et $\hat{g}(p)$ est *nouvelle* pour t **alors**
4: incrémenter le paramètre α correspondant à l'estimation $\hat{a}_{\hat{g}(p')\hat{g}(p)}$ {augmente la probabilité d'affinité entre $\hat{g}(p')$ et $\hat{g}(p)$}
5: **si** $\hat{g}(p') \neq \hat{g}(p)$ et $\hat{g}(p')$ est similaire à $\hat{g}(p)$ **alors**
6: fusionner $\hat{g}(p')$ et $\hat{g}(p)$
7: **fin si**
8: **fin si**
9: **fin pour**
10: **fin si**
11: **pour chaque** $p' \in P_{r',t}, r' \neq r$ **faire**
12: **si** $|P_{r',t}| \neq 1$ et l'interaction entre $\hat{g}(p')$ et $\hat{g}(p)$ est nouvelle pour t **alors**
13: **si** $\hat{g}(p') = \hat{g}(p)$ **alors**
14: séparer p' et p de $\hat{g}(p)$
15: **sinon**
16: incrémenter le paramètre β correspondant à l'estimation $\hat{a}_{\hat{g}(p')\hat{g}(p)}$ {diminue la probabilité d'affinité entre $\hat{g}(p')$ et $\hat{g}(p)$}
17: **fin si**
18: **fin si**
19: **fin pour**

la convergence de système. La valeur choisie a été expérimentalement validée et présente un bon compromis entre la précision et la vitesse de convergence.

Un mécanisme de séparation additionnel peut avoir lieu lors de la mise à jour d'une estimation. Si l'estimation de la probabilité de faute collective du groupe observés le plus large doit augmenter, alors la ressource responsable de cette augmentation est séparée du groupe observé.

5.4.2.2 Groupement par affinité

En utilisant la représentation par affinité, l'algorithme est plus simple (voir l'algorithme 5.3). D'abord, les affinités entre les groupes sont obtenues directement. De plus, les terminaisons des tâches ne sont pas utilisées. Enfin, il n'y a pas besoin d'un mécanisme de certification externe.

Les fusions et séparations sont plus simples. Les groupes observés i et j sont fusionnés seulement si aucun désaccord n'a été observé entre ces groupes (i.e., $\hat{a}_{ij} = 1$) et si le nombre d'observations est supérieur à $|i \cup j|$. Une séparation intervient si deux ressources calculatoires appartenant au même groupe observé renvoient des résultats distincts (faute individuelle mise à part) pour la même tâche.

5.5 Validation empirique

Les deux approches présentées dans la section 5.4 sont comparées empiriquement en utilisant des entrées de grande taille (de l'ordre du million d'événements $< d, p, t, r >$ ou $< d, t >$ pour une simulation donnée). Comme nous n'avons pas accès à des traces adaptées pour notre étude, nous avons agrégé différentes traces de plusieurs plateformes de calcul distribué en y adaptant notre modèle de menace. Les deux approches ont ensuite été étudiées avec chaque trace générée.

5.5.1 Description des données

La première source principale de traces est le projet *Failure Trace Archive* (FTA [101]) qui met à disposition des traces de disponibilité de systèmes parallèles et distribués. En particulier, nous utilisons les informations de disponibilité et de performance des ressources calculatoires du projet SETI@home [17]. D'autres traces sont aussi accessibles grâce au projet FTA : Overnet, un réseau pair-à-pair ; et Microsoft, un réseau d'entreprise.

Tandis que ces traces nous permettent d'assigner des tâches à des ressources dont les disponibilités et les performances sont connues, le coût des tâches doit être connu pour réaliser un véritable ordonnancement. Michela Taufer et al. [64] ont proposé une modélisation du temps de calcul nécessaire pour exécuter une tâche sur plusieurs plateformes de calcul distribué. Les durées des tâches obtenues ainsi dépassent rarement la centaine de secondes. Les durées se repartissent principalement en deux modalités, l'une centrée autour de 10 secondes, l'autre autour d'une minute. De plus, Michela Taufer nous a fourni des traces provenant du projet volontaire docking@home contenant la durée d'exécution d'un ensemble de tâches. Ces tâches durent plus longtemps (plus de 3 heures dans le cas médian).

Les tâches sont distribuées aux ressources calculatoires en utilisant un mécanisme par quorum (comme cela est fait par la plateforme BOINC). Chaque tâche est initialement distribuée à o ressources. Puis, la tâche est assignée à une nouvelle ressource jusqu'à l'obtention d'un quorum de q résultats ou jusqu'à l'utilisation de plus de l ressources.

L'hétérogénéité des performances est obtenue en normalisant la durée des tâches pour chaque ressource. Par exemple, nous considérons qu'une ressource deux fois plus rapide que la vitesse moyenne prendra deux fois moins de temps qu'une ressource moyenne pour calculer une tâche donnée. À chaque fois qu'une ressource devient indisponible, son calcul en cours est interrompu et repoussé jusqu'à ce qu'elle revienne sur le réseau. Enfin, une échéance est associée à chaque tâche (14 jours) et à chaque calcul (4 jours). Si une tâche n'est toujours pas finie 14 jours après le début de sa première exécution sur une ressource, alors elle est terminée et toutes ses instances en cours d'exécution sont annulées.

La façon dont les résultats sont générés dépend de notre modèle de menace. Une ressource renvoie par défaut un résultat correct. Nous considérons en plus les fautes individuelles et les fautes collectives. Ces premières fautes sont modélisées en séparant les ressources en deux sous-ensembles : une partie est fiable (fautes collectives mises à part) ; les autres échouent individuellement avec la même probabilité. Les groupes de faute collective sont basés sur le même principe : une ou plusieurs fractions sont spécifiées, définissant ainsi la quantité de ressources dans chaque groupe ; pour chaque groupe, une probabilité de faute collective est enfin précisée.

Le tableau 5.2 détaille les paramètres et les valeurs utilisés dans les simulations. Pour chaque paramètre, chaque valeur testée est utilisée (en considérant les valeurs par défaut pour tous les autres paramètres) pour générer une trace (i.e., une succession d'événements). On aboutit à quatre scénarios pour les fautes individuelles (en considérant toutes les combinaisons possibles de fractions et de probabilités). Pour les fautes collectives, il y a neuf scénarios. La valeur *Paire* signifie que 40% des ressources appartiennent à l'un des deux groupes de faute collective (chacun étant de taille égale). Dans le premier groupe, les fautes surviennent avec une probabilité égale à 0,2, alors que c'est toujours le cas dans le second. Cinq périodes espacées ont été considérées pour les traces provenant du projet SETI@home. En tout, nous avons 27 scénarios différents sans compter celui par défaut. Pour chacun, 20 racines différentes ont été utilisées (nécessaire pour le modèle de menace notamment), aboutissant à 560 traces sur lesquelles les deux approches ont été étudiées.

Enfin, l'algorithme 5.2 requiert un mécanisme de certification externe. Comme évoqué dans la section 5.4.2.1, ce mécanisme doit être suffisamment pertinent pour assurer une convergence

Paramètre	Valeur par défaut	Valeurs testées
Trace de disponibilité des ressources	Seti@home	Overnet, Microsoft, ...
Trace ou modèle des tâches	charmm	mfold, docking@home
Quorum (o, q, l)	$(4; 3; 10)$	$(2; 1; 2)$, $(19; 15; 19)$
Nombre de ressources (n)	100	30, 50, 70, 80, 200
Faute individuelle (fraction, probabilité)	$(0,7; 0,7)$	$(0,7; 0,99) \times (0,7; 0,99) \setminus (0,7; 0,7)$
Faute collective (fraction, probabilité)	$(0,2; 0,5)$	$(0,02; 0,2; 0,49) \times (0,01; 0,5; 1) \setminus (0,2; 0,5)$, Paire

TABLE 5.2 – Paramètres expérimentaux

de notre structure de données. Nous avons retenu une solution qui certifie le résultat correct s'il est présent parmi ceux générés et celui qui obtient la majorité dans le cas contraire.

5.5.2 Analyse des résultats

Le premier graphique (figure 5.5.1) illustre une exécution représentative et montre l'évolution de la précision de nos deux algorithmes en fonction du temps. En moins de 10 heures de temps simulé, le RMSD subit une baisse importante et se stabilise jusqu'à la fin de la simulation. À la fin de cette simulation, l'algorithme par affinité contient 3 groupes observés de tailles 79, 20 et un, ce qui correspond aux groupes réels si l'on ignore le groupe singleton. En plus, le RMSD est fortement impacté par une seule erreur qui concerne la probabilité d'affinité entre les deux groupes principaux (l'algorithme l'estime à 0,8 alors qu'elle est en réalité 0,5). Les autres erreurs agrégées dans le RMSD ne sont pas significatives. L'algorithme par faute produit les mêmes groupes observés de taille 79 et 1. En revanche, le groupe fautif de taille 20 n'est pas observé correctement : l'algorithme détecte trois groupes de tailles 10, 7 et 3. Leur probabilité d'échouer collectivement est estimée à 0,28 (0,5 en réalité) et c'est la principale source d'erreur qui influence le RMSD. Dans les deux cas, l'estimation de la structure des groupes de ressources ne possède pas de faux positif ni de faux négatif (ressource non-fautive mise dans un groupe fautif à tord, et réciproquement). Nous avons donc une estimation structurelle raisonnable bien que quantitativement inexacte.

Bien que les deux structures de données se soient stabilisées sur la figure 5.5.1, les estimations ne convergent pas de façon quantitative vers la réalité. Ces erreurs peuvent s'expliquer par l'ensemble des facteurs suivants : la probabilité de faute collective est différente de 1 ; la valeur de quorum est faible ; et, la fraction des ressources causant des fautes collectives est faible. Avec ces paramètres, il y a de nombreuses occurrences où une ressource peut être en situation de faute collective pour une tâche donnée sans qu'aucune autre ressource ne renvoie le même résultat incorrect. Cela arrive si la ressource est la seule de son groupe fautif à devoir calculer la tâche en question. Les algorithmes proposés considèrent alors ce comportement comme une faute individuelle (ligne 2 de l'algorithme 5.1). Cette incapacité à déterminer la cause de l'inexactitude d'un résultat orphelin implique que même si nos algorithmes convergent, ils produisent des estimations inexactes.

Les figures suivantes montrent les résultats obtenus sur des sous-ensembles de simulations. La majeure partie de nos simulations couvre environ trois jours de temps simulé. Le temps de convergence doit donc être inférieur à cette limite pour pouvoir statuer sur la rapidité de nos algorithmes. Nous avons observé empiriquement que lorsque le RMSD a une valeur inférieure

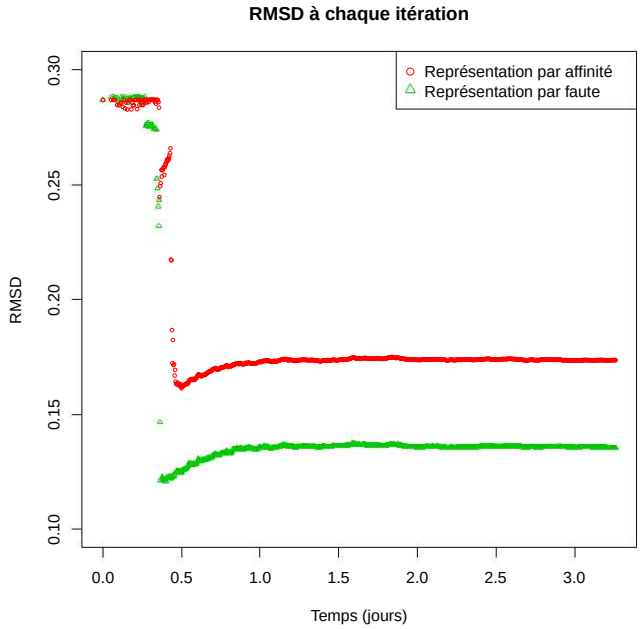


FIGURE 5.5.1 – Analyse d'une exécution avec des paramètres par défaut.

à 0,2, la structure des groupes de ressources estimées est proche de la réalité.

Les figures 5.5.2 à 5.5.6 représentent les temps de convergence et les précisions stabilisées mesurés pour les deux algorithmes. Chaque figure correspond à la variation d'un paramètre, les autres étant fixés à leur valeur par défaut. Les résultats sont représentés par l'intermédiaire de diagrammes en violon. Ces graphiques sont une combinaison de boîte à moustaches et de noyau de densité. Une boîte à moustache est constituée d'un segment et d'un rectangle (dans la partie colorée). Le rectangle s'éloigne de la médiane (la marque blanche) de chaque coté jusqu'au premier et troisième quartile. Le segment s'écarte de la médiane jusqu'à 1,5 fois l'intervalle interquartile. Intuitivement, la majeure partie des mesures se concentre dans la zone couverte par le segment et la moitié de ces mesures est dans le rectangle. Le noyau de densité est une estimation de la densité de probabilité des points mesurés. Il s'agit plus ou moins d'un histogramme lissé. Plus une zone est large, plus elle contient de points. Les diagrammes en violon sont particulièrement pratiques lorsque les mesures sont concentrées sur plusieurs modalités.

L'efficacité des algorithmes proposés est d'abord étudiée en utilisant plusieurs traces de disponibilité sur la figure 5.5.2. Nous constatons que les simulations se comportent similairement quelque soit les traces SETI@home. À savoir, les deux algorithmes convergent en moins d'une demi-journée dans la plupart des cas. De plus, la précision stabilisée est meilleure avec la représentation par faute. Avec les traces microsoft, le temps de convergence est plus faible. En effet, les ressources sont plus souvent présentes dans un réseau d'entreprise et les interactions sont donc plus fréquentes. Bien que les traces overnet révèlent des précisions stabilisées comparables, les temps de convergence sont étrangement distribués. Ce comportement reste inexpliqué.

L'effet du type de tâches utilisé est représenté sur la figure 5.5.3a. Nous pouvons tirer les mêmes conclusions en terme de précision stabilisée, c'est-à-dire que la représentation par faute est meilleure et que ces résultats ne dépendent pas de la charge de calcul. Les temps de convergence pour le projet docking@home dépassent la limite supérieure de l'axe des ordonnées car les tâches prennent plus de temps en comparaison des modèles charmm et mfold. Pour ces deux derniers

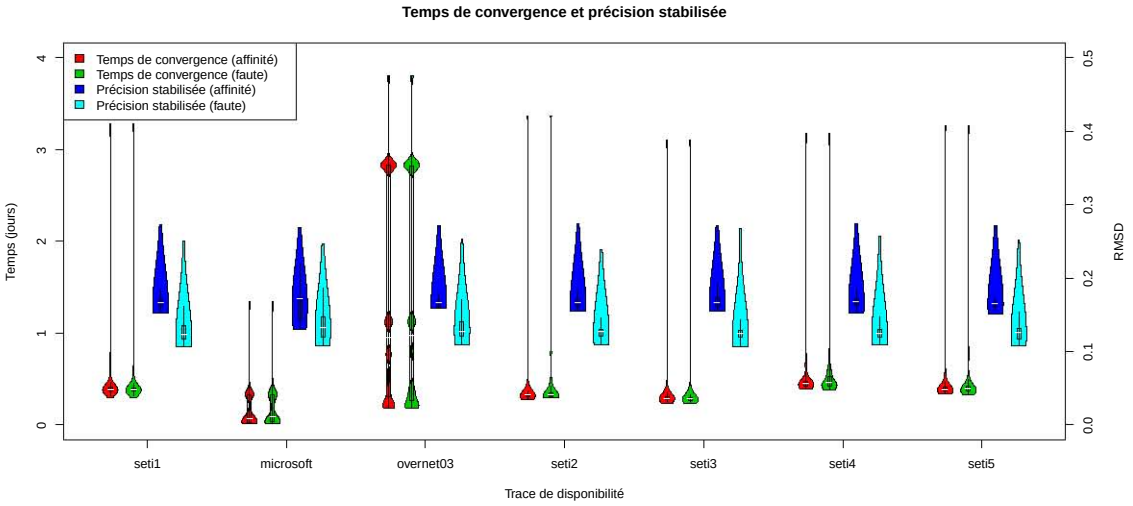


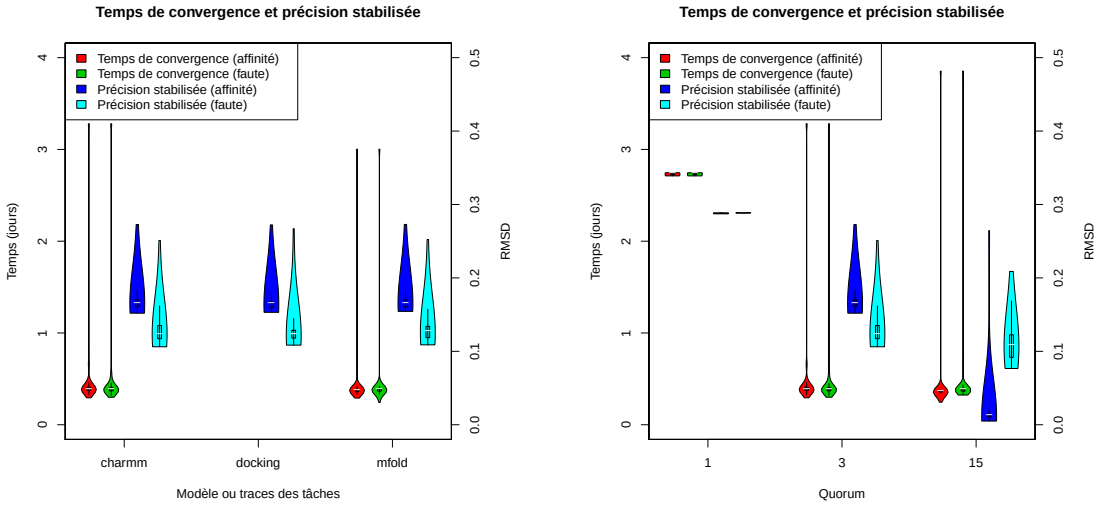
FIGURE 5.5.2 – Traces de disponibilité

modèles de tâche, les temps de convergence sont en revanche similaires.

Différentes valeurs de quorum ont été testées et les résultats sont présentés sur la figure 5.5.3b. Un quorum égal à 1 ne permet pas à nos algorithmes de fonctionner correctement. Cela s'explique par l'absence d'interaction entre les groupes de ressources (pour obtenir une observation complète, il faut au moins quatre ressources assignées à la même tâche et dont les résultats se divisent en deux groupes de même taille). Bien que les temps de convergence soient similaires lorsque le quorum vaut 15, la représentation par affinité permet d'obtenir une excellente précision stabilisée. Intuitivement, cela s'explique par l'augmentation du nombre d'observations réalisées.

La figure 5.5.4 montre l'évolution de l'efficacité de nos algorithmes lorsque le nombre de ressources calculatoires dans le système augmente. Outre une légère augmentation du temps de convergence, les résultats sont stables jusqu'à ce que ce nombre dépasse 100. Avec 200 ressources, la moitié des simulations ne converge pas, ce qui explique la dégradation observée en terme de précision stabilisée. D'autre part, on remarque que le temps de convergence maximal décroît. En effet, le temps simulé avance plus lentement lorsque davantage de ressources sont présentes. Avec un nombre fixé d'événements (un million), le temps simulé est donc plus court.

Sur les figures 5.5.5 et 5.5.6, nous varions la fraction des ressources échouant collectivement et leurs probabilités de faute collective. Par exemple, sur la première colonne de la figure 5.5.6, la probabilité de faute est 0,01, mais la fraction de ressources fautives est soit 0,02, 0,2 ou 0,49. Le scénario où la probabilité de faute est 0,5 et la fraction 0,02 apparaît comme le scénario le plus difficile. Aucune de nos approches ne détecte cette situation et la modalité supérieure présente sur les deux figures correspond à cette situation. Dans tous les cas, spécifier une probabilité égale à 0,01 aboutit à une bonne précision avec une convergence rapide. Cependant, pour une probabilité aussi faible, le RMSD n'est pas une mesure pertinente car il agrège des erreurs absolues et non pas relatives. Dans les faits, nos algorithmes convergent vers des estimations égales à 0 (à la place de 0,01), pour lesquelles la valeur du RMSD est faible. En fait, le groupe fautif est difficile à détecter avec ces paramètres et le faible temps de convergence indique que nos approches ne détectent pas de fautes collectives. Ces faibles valeurs expliquent la modalité inférieure présente sur la figure 5.5.5. Au contraire, une probabilité de faute collective égale à 1 rend la détection plus facile qu'avec une probabilité valant 0,5. Nous observons d'ailleurs que les algorithmes proposés ont une bonne précision dans de tels cas. Une autre raison pour l'expliquer



(a) Modèle ou traces des tâches (b) Quorum

FIGURE 5.5.3 – Temps de convergence et précision

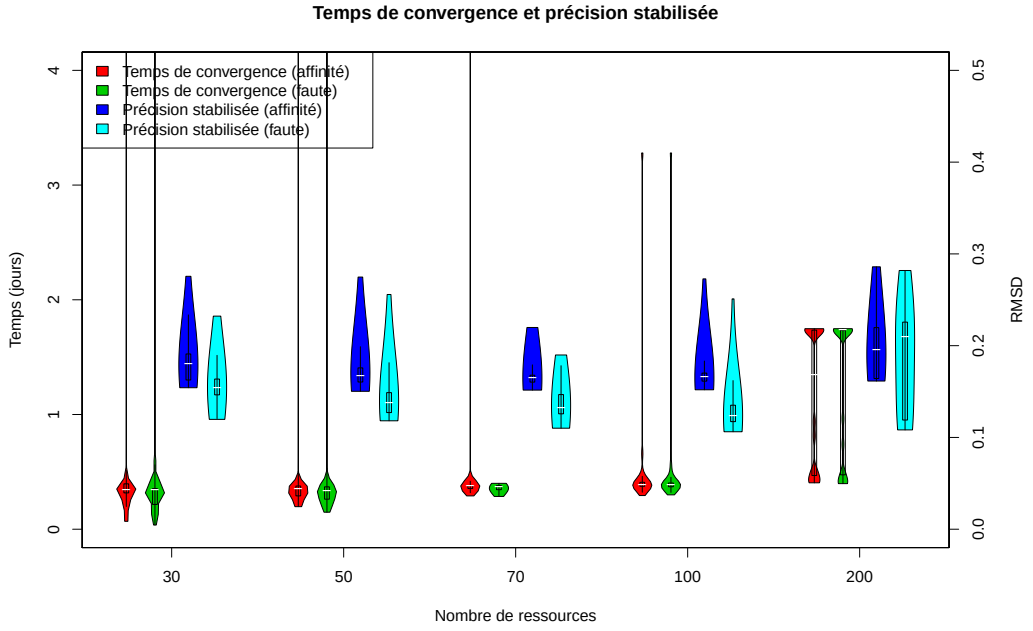


FIGURE 5.5.4 – Quantité de ressources

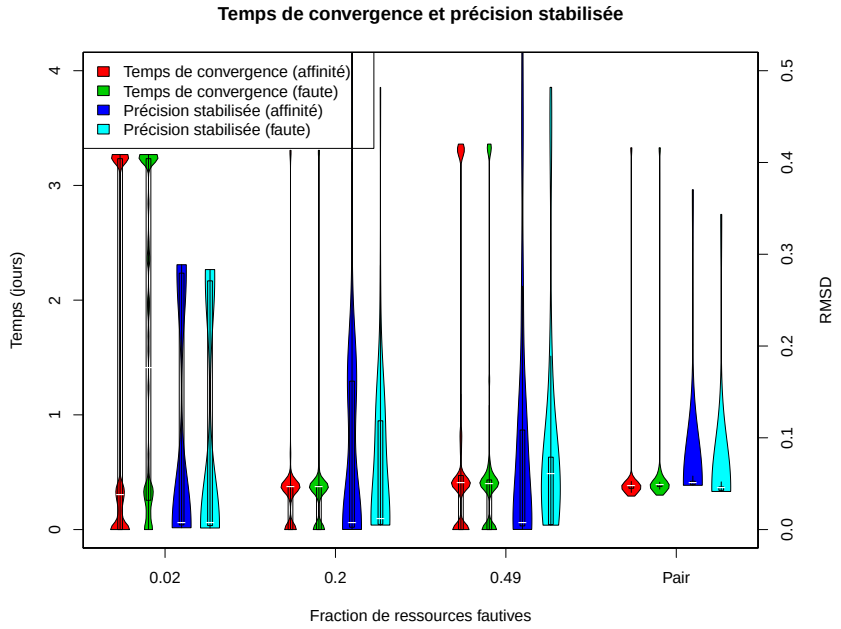


FIGURE 5.5.5 – Fraction de ressources fautives

est liée aux statistiques : estimer avec le même degré de confiance une probabilité égale à 1 ou à 0 requiert moins d'observations que lorsque la probabilité à estimer diffère de ces deux valeurs. Une quasi-majorité de ressources fautives (49%) semblait à priori plus difficile à détecter qu'un groupe fautif plus réduit. Bien qu'il existe certains cas où nos algorithmes ne convergent pas avec une fraction valant 0,49, la différence est faible par rapport à une fraction valant 0,2. Finalement, la présence de deux groupes fautifs ne représente pas une difficulté majeure pour nos algorithmes.

5.6 Conclusion

Les fautes collectives menacent les plateformes de calcul distribué qui ne sont pas munies de techniques adéquates pour y faire face. Notre modèle de menace est fort : il permet le recouvrement des groupes fautifs ; les fautes collectives surviennent aléatoirement ; aucune synchronisation entre les ressources n'est nécessaire ; et aucune information sur les ressources n'est connue à priori.

Nous proposons deux algorithmes, chacun basé sur une représentation des fautes collectives différente. Ils sont tous les deux efficaces et identifient correctement les groupes fautifs. L'algorithme par faute est meilleur en terme de précision numérique. En revanche, l'algorithme par affinité est plus simple et ne dépend pas d'un mécanisme de certification externe.

Il reste de nombreux aspects à développer concernant le problème traité. Le premier axe de recherche se concentre sur la gestion de la non-stationnarité des comportements fautifs. Dans le cas d'une infection par un virus ou d'une erreur logicielle qui est finalement corrigée, il convient de détecter les changements que subissent les ressources. Proposer un système adaptatif présente cependant des écueils. Il est notamment nécessaire de faire un compromis entre la précision du système et la rapidité avec laquelle il détecte les changements intervenus. Élaborer une stratégie d'ordonnancement se basant sur le mécanisme de caractérisation proposé dans ce chapitre constitue une seconde perspective de recherche. Cela implique de faire cohabiter optimalement les

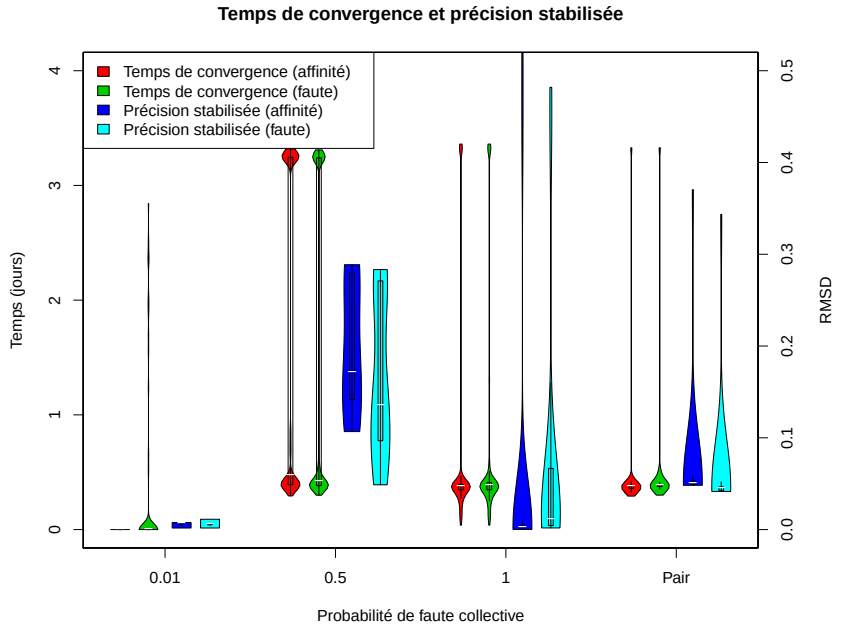


FIGURE 5.5.6 – Probabilité de faute collective

nécessités du mécanisme de caractérisation (pour qu'il puisse produire des estimations précises) et les critères d'efficacité et de précision de l'algorithme d'ordonnancement.

Conclusion

Nous revenons synthétiquement sur l'étude et les contributions apportées dans chaque partie de cette thèse et développons des conclusions générales sur l'ordonnancement en présence d'incertitude. Nous terminons sur les perspectives découlant de nos travaux et se rapportant à la thèse.

Résultats

Méthodes génériques

Examiner l'effet de l'incertitude sur les ordonnancements nous amène à considérer les aspects probabilistes et multicritères qui sont traités dans la première partie.

Dans le chapitre 1, le problème consiste à déterminer la distribution d'un graphe stochastique. Nous rappelons qu'il s'agit d'appliquer les opérations de somme et de maximum sur des variables aléatoires. Deux heuristiques sont proposées et présentent chacune un compromis en terme de précision et de temps qui se révèle avantageux. Les techniques proposées sont utilisées dans le chapitre 3 et pourraient être adaptées au problème du chapitre 4. Enfin, des opérations arithmétiques sont réalisées sur des lois bêtas indépendantes dans le chapitre 5, ce qui est un cas particulier du problème. L'évaluation d'un graphe stochastique est donc un problème fondamental.

Le chapitre 2 aborde l'optimisation multicritère et plus spécifiquement l'ordonnancement bicritère. Il s'agit principalement d'un état de l'art accompagné de quelques contributions concernant la conception d'heuristiques gloutonnes multicritères et la convergence d'un algorithme évolutionnaire. Les outils présentés ont une application directe dans le chapitre 3 (dans lequel elles sont d'ailleurs évaluées) et le chapitre 4. La dimension multicritère est également présente dans le chapitre 5. Dans ce dernier cas, cependant, les techniques multicritères en question ne sont pas adaptées puisque le problème d'ordonnancement sous-jacent est dynamique. Cela invalide par exemple l'usage d'une métaheuristique.

Problèmes d'ordonnancement

La seconde partie repose sur l'analyse de problèmes représentatifs de différentes modalités en terme d'ordonnancement et d'incertitude.

L'application à ordonnancer est un graphe de tâches dans le chapitre 3 et le chapitre 4, alors qu'il s'agit de tâches indépendantes dans le chapitre 5. De plus, nous nous positionnons dans un contexte statique dans ces deux premiers chapitres et dans un contexte dynamique dans le dernier chapitre. Rappelons enfin que l'incertitude se porte sur un élément différent dans chacun de ces chapitres : la durées des calculs dans le chapitre 3 ; le succès des calculs dans le chapitre 4 ; et, l'exactitude des résultats dans le chapitre 5.

Le chapitre 3 et le chapitre 4 s'appuient sur des résultats de complexité pour justifier l'emploi d'heuristiques que nous avons par ailleurs empiriquement validées. La complexité du problème traité dans le chapitre 5 n'est pas caractérisée car il se base sur des observations statistiques qui ne permettent pas de construire systématiquement une solution optimale. Nous proposons en revanche des représentations que nous avons analysées formellement et validées empiriquement.

L'analyse des trois problèmes étudiés dans la seconde partie permet d'affirmer que les problèmes d'ordonnancement en présence d'incertitude sont invariablement plus difficiles. Nous en déclinons trois conclusions méthodologiques. D'abord, de nouveaux critères en lien avec l'incertitude viennent s'ajouter aux critères d'efficacité rendant les problèmes multicritères. Ensuite, nous remarquons le passage d'une situation déterministe à une situation stochastique ce qui nécessite l'usage d'outils probabilistes. Enfin, il en découle que tout nouveau critère fait appel à des concepts probabilistes. Par exemple, il peut nécessiter l'évaluation d'une ou de plusieurs probabilités, son évaluation se retrouvant ainsi dans la classe #P'. Au vu des résultats de cette thèse, il est naturel d'en soupçonner la complétude dès lors qu'un graphe intervient dans l'évaluation du critère.

Perspectives

De nombreuses directions de recherche spécifiques à chaque problématique sont proposées dans les chapitres correspondants. Ainsi, la vitesse de convergence de l'algorithme génétique proposé dans la section 2.4 pourrait faire l'objet de recherche future. Notons aussi que la complexité d'un problème d'évaluation décrit dans le chapitre 4 reste ouvert. Enfin, l'étape d'ordonnancement dans le chapitre 4 et le chapitre 5 n'a pas été traitée.

Outre ces axes de recherche directs, trois perspectives générales émergent de nos travaux.

Ordonnancement statique et dynamique

Dans plusieurs chapitres, l'étude se focalise sur l'ordonnancement statique. Trois arguments sont avancés : une approche dynamique peut se baser sur un ordonnancement statique ; davantage de techniques d'optimisation sont disponibles en statique ; l'analyse d'ordonnancements statiques prépare l'étude des stratégies dynamiques. Plusieurs directions de recherche en découlent.

D'abord, il serait intéressant de proposer des techniques dynamiques pour adapter l'exécution d'un ordonnancement aux imprévus (durées dépassant un seuil spécifique ou pannes). Ce type de techniques s'appuie sur des prédictions à court ou moyen terme pour réaliser des décisions pertinentes. Bien qu'estimer le déroulement de l'exécution à court terme ne pose sans doute pas de difficultés, il n'est pas souhaitable d'être confronté à un problème d'évaluation #P'-Complet comme ceux rencontrés dans le chapitre 3 et le chapitre 4. C'est sur cet aspect notamment que l'étude d'ordonnancements statiques met en valeur les écueils possibles en ordonnancement dynamique. L'efficacité d'une telle stratégie dynamique pour une instance donnée semble d'autre part difficile à évaluer et nous conjecturons que cet aspect est au moins aussi dur qu'en statique.

Parmi les méthodes proposées, la métaheuristique décrite dans le chapitre 2 est la seule technique d'optimisation qui n'est pas gloutonne. Des méthodes sophistiquées et spécifiques à l'ordonnancement statique mériteraient de faire l'objet de recherches futures.

Techniques de décision multicritère

La littérature est très riche en terme de techniques d'optimisation multicritère. L'objectif principalement considéré dans cette thèse consiste à énumérer l'ensemble des ordonnancements Pareto-optimaux. En toute généralité, le nombre de tels ordonnancements peut dépasser une

quantité raisonnable. Il existe deux types d'approches pour formaliser les problèmes de manière à limiter le nombre de solutions qu'il faut générer :

- inclure un des indicateurs présentés dans la section 2.2.3 dans la formulation du problème ;
- préciser le lien entre les critères par un opérateur d'agrégation comme cela est décrit dans la section 2.3.1.

Une méthode classique consiste à fixer un seuil pour l'un des critères et à optimiser l'autre (il s'agit de la méthode ϵ-contrainte). Aborder l'aspect multicritère dans une telle optique faciliterait le développement d'approches garanties.

Modèles d'incertitude

Même en réduisant la taille du résultat recherché grâce aux approches multicritères mentionnées dans le paragraphe précédant, les problèmes d'ordonnancements décrits dans le chapitre 3 et le chapitre 4 resteraient tout autant difficiles. Ce qui rend l'étude de ces problèmes si compliquée tient notamment au coût de l'évaluation de chaque ordonnancement : l'évaluation de la robustesse ou de la fiabilité d'un ordonnancement est même inapproximable. Il serait ainsi étonnant qu'un algorithme d'approximation existe. Parallèlement à l'intégration des techniques d'optimisation multicritère dans les problématiques, utiliser des modèles d'incertitude plus simples permettant une évaluation en temps polynomial offrirait plus de chances d'aboutir à des algorithmes garantis.

Par exemple, nous pourrions considérer qu'une seule panne peut survenir pendant l'exécution d'un ordonnancement donné. Cela limiterait le nombre de cas possibles à une quantité polynomiale. Il s'agirait en plus d'un modèle réaliste lorsque les probabilités de pannes multiples sont faibles. D'autre part, l'insertion d'une ressource calculatoire sûre dans les plateformes volontaires permettrait d'offrir des garanties partielles. Cette ressource servirait alors à vérifier l'exactitude de certains résultats pour déterminer plus rapidement et plus sûrement quels sont les groupes fautifs.

Publications scientifiques

Revues internationnales

[1] Louis-Claude Canon, Olivier Dubuisson, Jens Gustedt et Emmanuel Jeannot : Defining and Controlling the Heterogeneity of a Cluster : the Wrekavoc Tool. *Journal of Systems and Software*, 83(5):786–802, mai 2010.

[2] Louis-Claude Canon et Emmanuel Jeannot : Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, avril 2010.

Conférences internationales avec publications des actes et comité de sélection

[3] Louis-Claude Canon et Emmanuel Jeannot : Wrekavoc a Tool for Hemulating Heterogeneity. *In 15th IEEE Heterogeneous Computing Workshop (HCW'06)*, Island of Rhodes, Greece, avril 2006.

[4] Louis-Claude Canon et Emmanuel Jeannot : A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. *In HeteroPar'07*, pages 568–567, Austin, Texas, USA, septembre 2007.

[5] Louis-Claude Canon et Emmanuel Jeannot : Scheduling Strategies for the Bicriteria Optimization of the Robustness and Makespan. *In 11th International Workshop on Nature Inspired Distributed Computing (NIDISC 2008)*, Miami, Floride, USA, avril 2008.

[6] Louis-Claude Canon, Emmanuel Jeannot, Rizos Sakellariou et Wei Zheng : Comparative Evaluation of the Robustness of DAG Scheduling Heuristics. *In Proceedings of CoreGRID Integration Workshop*, Heraklion-Crete, Greece, avril 2008.

[7] Louis-Claude Canon, Emmanuel Jeannot et Jon Weissman : A dynamic approach for characterizing collusion in desktop grids. *In 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Atlanta, Georgia, USA, avril 2010.

Conférences francophones avec publications des actes et comité de sélection

[8] Louis-Claude Canon : Évaluation et Comparaison de Métriques de Robustesse pour l'Ordonnancement de Graphes de Tâches sur des Systèmes Hétérogènes. *In Proceedings of RenPar'18*, Fribourg, Suisse, février 2008.

[9] Louis-Claude Canon et Emmanuel Jeannot : Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules. *In 10ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, pages 13–24, Nancy, France, février 2009.

Rapports de recherche

[10] Anne Benoit, Louis-Claude Canon, Emmanuel Jeannot et Yves Robert : On the complexity of task graph scheduling with transient and fail-stop failures. Rapport technique 2010-01, LIP, ENS Lyon, France, janvier 2010. Available at `graal.ens-lyon.fr/~yrobert`.

[11] Louis-Claude Canon et Emmanuel Jeannot : Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments. Research Report 6476, INRIA, mars 2008.

[12] Louis-Claude Canon et Emmanuel Jeannot : Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules. Research Report 6895, INRIA, avril 2009.

[13] Louis-Claude Canon, Emmanuel Jeannot et Jon Weissman : A Scheduling Algorithm for Defeating Collusion. Research Report 7403, INRIA, octobre 2010.

Bibliographie

[14] Shoukat Ali, Anthony A. Maciejewski, Howard Jay Siegel et Jong-Kook Kim : Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, juillet 2004.

[15] Myles Allen : Do it yourself climate prediction. *Nature*, 401:642, octobre 1999.

[16] David P. Anderson : Boinc : A system for public-resource computing and storage. *In 5th International Workshop on Grid Computing (GRID)*, pages 4–10, novembre 2004.

[17] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky et Dan Werthimer : SETI@home : An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, 2002.

[18] Baruch Awerbuch, Yossi Azar, Amos Fiat et Frank T. Leighton : Making commitments in the face of uncertainty : How to pick a winner almost every time. *In 28th ACM SToC*, pages 519–530, 1996.

[19] J. Bannister et K. S. Trivedi : Task allocation in fault-tolerant distributed systems. *Acta Informatica*, 20:261–281, 1983.

[20] Richard E. Barlow et Frank Proschan : *Mathematical theory of reliability*. John Wiley, New York, 1967.

[21] Thomas Bayes : An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

[22] Olivier Beaumont, Vincent Boudet et Yves Robert : A realistic model and an efficient heuristic for scheduling with heterogeneous processors. *In 11th IEEE Heterogeneous Computing Workshop (HCW'02)*, 2002.

[23] Richard Beigel, William Hurwood et Nabil Kahale : Fault diagnosis in a flash. *In FOCS*, pages 571–580, 1995.

[24] Wolfgang W. Bein, Jerzy Kamburowski et Matthias F. M. Stallmann : Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 21(6): 1112–1129, 1992.

[25] Charles H. Bennett et John Gill : Relative to a Random Oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq$ co-$\mathbf{NP}^A$ with Probability 1. *SIAM Journal on Computing*, 10(1):96–113, 1981.

[26] S. N. Bhatt, F. R. K. Chung, Frank T. Leighton et Arnold L. Rosenberg : On optimal strategies for cycle-stealing in networks of workstations. *IEEE Transactions on Computers*, 46(5):545–557, 1997.

[27] A. Bhattacharyya : On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35(4):99–109, 1943.

[28] David Blaauw, Kaviraj Chopra, Ashish Srivastava et Lou Scheffer : Statistical timing analysis : From basic principles to state of the art. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(4), avril 2008.

[29] Stefan Bleuler, Marco Laumanns, Lothar Thiele et Eckart Zitzler : PISA — a platform and programming language independent interface for search algorithms. *In First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, pages 494 – 508, Berlin, 2003.

[30] Hans L. Bodlaender et Thomas Wolle : A note on the complexity of network reliability problems. *IEEE Transactions on Information Theory*, 47:1971–1988, 2004.

[31] Ladislau Bölöni et Dan C. Marinescu : Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5):395–412, septembre 2002.

[32] George E. P. Box et Norman R. Draper : *Empirical Model-Building and Response Surfaces*. Wiley, 1987.

[33] B. Bream : Reliability Block Diagrams and Reliability Modeling. Rapport technique, Office of Safety and Mission Assurance, NASA Lewis Research Center, 1995.

[34] Franc Brglez, David Bryan et Krzysztof Koźmiński : Combinational profiles of sequential benchmark circuits. *In IEEE International Symposium on Circuits and Systems*, 1989.

[35] Peter Brucker : *Scheduling Algorithms*. Springer-Verlag, 2004.

[36] John M. Burt et Mark B. Garman : Conditional Monte Carlo : A Simulation Technique for Stochastic Network Analysis. *Management Science*, 18(3):207–217, novembre 1971.

[37] Sébastien Cahon, Nordine Melab et El-Ghazali Talbi : ParadisEO : A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[38] K. M. Chandy et P. F. Reynolds : Scheduling partially ordered tasks with probabilistic execution times. *In SOSP '75 : Proceedings of the fifth ACM symposium on Operating systems principles*, pages 169–177, New York, NY, USA, 1975.

[39] W.Y. Chiang et M.S. Fox : Protection against uncertainty in a deterministic schedule. *In Procedings of the Fourth International Conference on Expert Systems and Leading Edge in Production and Operations Management*, pages 184–197, 1990.

[40] SungJin Choi, HongSoo Kim, EunJoung Byun, MaengSoon Baik, SungSuk Kim, ChanYeol Park et ChongSun Hwang : Characterizing and Classifying Desktop Grid. *In 7th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'07)*, pages 743–748, Rio de Janeiro, Brazil, mai 2007.

[41] Gustave Choquet : Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295, 1953.

[42] Charles E. Clark : The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, mars/avril 1961.

[43] Carlos A. Coello Coello, Gary B. Lamont et David A. Van Veldhuizen : *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer-Verlag New York Inc, 2007.

[44] Richard Cole et Ramesh Hariharan : Dynamic LCA Queries on Trees. *In Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 235–244. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1999.

[45] David W. Corne, Nick R. Jerram, Joshua D. Knowles et Martin J. Oates : PESA-II : Region-based Selection in Evolutionary Multiobjective Optimization. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 283–290, San Francisco, California, USA, août 2001.

[46] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian et Thorsten von Eicken : Logp : towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1–12, 1993.

[47] Richard L. Daniels et Janice E. Carrillo : β-Robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29(11):977–985, novembre 1997.

[48] Andrew J. Davenport et J. Christopher Beck : A survey of techniques for scheduling with uncertainty. Unpublished, 2002.

[49] Andrew J. Davenport, Christophe Gefflot et J. Christopher Beck : Slack-based Techniques for Robust Schedules. *In Proceedings of the Sixth European Conference on Planning (ECP-2001)*, pages 7–18, Toledo, Spain, septembre 2001.

[50] Kalyanmoy Deb, Amrit Pratab, Samir Agrawal et T. Meyarivan : A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization : NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, avril 2002.

[51] Kalyanmoy Deb et Santosh Tiwari : Omni-optimizer : A generic evolutionary algorithm for single and multi-objective optimization. *European Journal of Operational Research*, 185(3):1062–1087, 2008.

[52] Josep Díaz, Jordi Petit et Maria J. Serna : A Survey of Graph Layout Problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[53] Bajis Dodin : Bounding the project completion time distribution in PERT networks. *Operations Research*, 33(4):862–881, juillet 1985.

[54] Patrício Domingues, Bruno Sousa et Luís Moura Silva : Sabotage-Tolerance and Trust Management in Desktop Grid Computing. *Future Generation Computer Systems*, 23(7): 904–912, août 2007.

[55] Jack J. Dongarra, Emmanuel Jeannot, Erik Saule et Zhiao Shi : Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. *In 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'07)*, San Diego, CA, USA, juin 2007.

[56] John R. Douceur : The sybil attack. *In* Druschel *et al.* [57], pages 251–260.

[57] Peter Druschel, M. Frans Kaashoek et Antony I. T. Rowstron, éditeurs. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 de *Lecture Notes in Computer Science*. Springer, 2002.

[58] Elias Procópio Duarte Jr. et Takashi Nanya : A hierarchical adaptive distributed system-level diagnosis algorithm. *IEEE Transactions on Computers*, 47(1):34–45, 1998.

[59] Didier Dubois et Henri Prade : *Possibility Theory : An Approach to Computerized Processing of Uncertainty*. Plenum Press, New York, 1988.

[60] Matthias Ehrgott : *Multicriteria Optimization*. Springer, 2005.

[61] Matthias Ehrgott et Margaret Wiecek : Multiobjective programming. *In* J. Figueira, S. Greco et Matthias Ehrgott, éditeurs : *Multicriteria Decision Analysis : State of the Art Surveys*, pages 667–722. Springer Science, 2005.

[62] Hesham El-Rewini, Theodore G. Lewis et Hesham H. Ali : *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, 1994.

[63] Darin England, Jon Weissman et Jayashree Sadagopan : A New Metric for Robustness with Application to Job Scheduling. *In 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, pages 135–143, juillet 2005.

[64] Trilce Estrada, Michela Taufer et Kevin Reed : Modeling job lifespan delays in volunteer computing projects. *In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pages 331–338, Washington, DC, USA, 2009.

[65] Helene Fargier, Philippe Fortemps et Didier Dubois : Fuzzy scheduling : Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147(2):231–252, 2003.

[66] Stefan Felsner et Klaus Reuter : The Linear Extension Diameter of a Poset. *SIAM Journal on Discrete Mathematics*, 12(3):360–373, 1999.

[67] Michael J. Flynn : Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 21(C):948–960, 1972.

[68] D. R. Fulkerson : Expected Critical Path Lengths in PERT Networks. *Operations Research*, 10(6):808–817, 1962.

[69] Harold N. Gabow : Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. *In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1990.

[70] H. Gao : Building robust schedules using temporal protection–an empirical study of constraint based scheduling under machine failure uncertainty. Mémoire de D.E.A., Departement of Industrial Engineering, University of Toronto, 1995.

[71] Apostolos Gerasoulis, Jia Jiao et Tao Yang : Experience with Graph Scheduling for Mapping Irregular Scientific Computation. *In First IPPS workshop on Solving Irregular Problems on Distributed Memory Machines*, pages 1–8, Santa Barbara, CA, USA, avril 1995.

[72] Apostolos Gerasoulis et Tao Yang : A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, 1992.

[73] Alain Girault et Hamoudi Kalla : A novel bicriteria scheduling heuristic providing a guaranteed global system failure rate. *IEEE Transactions on Dependable and Secure Computing*, 6(4):241–254, 2009.

[74] Alain Girault, Erik Saule et Denis Trystram : Reliability versus performance for critical applications. *Journal of Parallel and Distributed Computing*, 69(3):326–336, mars 2009.

[75] D. E. Goldberg : *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[76] R. L. Graham, E. L. Lawler, J. K. Lenstra et A. H. G. Rinnooy Kan : Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[77] Jane N. Hagstrom : Computational complexity of PERT problems. *Networks*, 18(2):139–147, 1988.

[78] Yacov Y. Haimes : *Risk Modeling, Assessment, and Management*. Wiley Series in Systems Engineering and Management, 2009.

[79] Yacov Y. Haimes, L. Ladson et D. Wismer : On a bicriteria formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:296–297, 1971.

[80] Nicholas G. Hall et Marc E. Posner : Sensitivity analysis for scheduling problems. *Journal of Scheduling*, 7(1):49–83, janvier 2004.

[81] Mark C. Hansen, Hakan Yalcin et John P. Hayes : Unveiling the iscas-85 benchmarks : A case study in reverse engineering. *IEEE Design & Test*, 16(3):72–80, 1999.

[82] Michael Pilegaard Hansen et Andrzej Jaszkiewicz : Evaluating the quality of approximations to the non-dominated set. Rapport technique IMM-REP-1998-7, Technical University of Denmark, mars 1998.

[83] Jun He et Xin Yao : Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145:59–97, 2003.

[84] Willy Herroelen et Roel Leus : Robust and reactive project scheduling : a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, avril 2004.

[85] Willy Herroelen et Roel Leus : Project scheduling under uncertainty : Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, septembre 2005.

[86] Edwin Hou, Nirwan Ansari et Hong Ren : A genetic algorithm for multiprocessor scheduling. *Journal on Parallel and Distributed Systems*, 5(2), février 1994.

[87] G. Igelmund et F. J. Radermacher : Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13(1):1–28, 1983.

[88] Emmanuel Jeannot, Erik Saule et Denis Trystram : Bi-Objective Approximation Scheme for Makespan and Reliability Optimization on Uniform Parallel Machines. *In 14th International Euro-Par Conference on Parallel and Distributed Computing*, pages 877–886, Spain, août 2008.

[89] Yaochu Jin et Jürgen Branke : Evolutionary optimization in uncertain environments–a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–3017, juin 2005.

[90] Audun Jøsang : An algebra for assessing trust in certification chains. *In Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*. The Internet Society, 1999.

[91] Audun Jøsang : The beta reputation system. *In Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.

[92] Audun Jøsang, Roslan Ismail et Colin Boyd : A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, mars 2007.

[93] Jerzy Kamburowski : Normally Distributed Activity Durations in PERT Networks. *The Journal of the Operational Research Society*, 36(11):1051–11057, novembre 1985.

[94] Sepandar D. Kamvar, Mario T. Schlosser et Hector Garcia-Molina : The EigenTrust Algorithm for Reputation Management in P2P Networks. *In WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003.

[95] S. Kartik et C. Siva Ram Murthy : Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Transactions on Computers*, 46(6):719–724, 1997.

[96] Seung-Jean Kim, Stephen P. Boyd, Sunghee Yun, Dinesh D. Patil et Mark A. Horowitz : A Heuristic for Optimizing Stochastic Activity Networks with Applications to Statistical Digital Circuit Sizing. *Optimization and Engineering*, 8(4):397–430, décembre 2007.

[97] George B. Kleindorfer : Bounding Distributions for a Stochastic Acyclic Network. *Operations Research*, 19(7):1586–1601, novembre 1971.

[98] Joshua D. Knowles et David W. Corne : On Metrics for Comparing Nondominated Sets. *In 4th IEEE Congress on Evolutionary Computation (CEC 2002)*, volume 71, pages 1–716, 2002.

[99] R. Kolisch et A. Sprecher : Psplib - a project scheduling library. *European Journal of Operational Research*, 96:205–216, 1996.

[100] Derrick Kondo, Filipe Araujo, Paul Malecot, Patrício Domingues, Luís Moura Silva, Gilles Fedak et Franck Cappello : Characterizing result errors in internet desktop grids. *In 13th International Euro-Par Conference on Parallel and Distributed Computing*, pages 361–371, Rennes, France, août 2007.

[101] Derrick Kondo, Bahman Javadi, Alex Iosup et Dick Epema : The failure trace archive : Enabling comparative analysis of failures in diverse distributed systems. *In 10th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'10)*, mai 2010.

[102] Evangelos Kranakis et Andrzej Pelc : Better adaptive diagnosis of hypercubes. *IEEE Transactions on Computers*, 49(10):1013–1020, 2000.

[103] Axel Krings, Jean-Louis Roch et Samir Jafar : Certification of large distributed computations with task dependencies in hostile environments. *In* IEEE, éditeur : *IEEE Electro/Information Technology Conference (EIT 2005)*, Lincoln, Nebraska, mai 2005.

[104] Axel Krings, Jean-Louis Roch, Samir Jafar et Sébastien Varrette : A probabilistic approach for task and result certification of large-scale distributed applications in hostile environments. *In Proceedings of the European Grid Conference (EGC2005)*, Amsterdam, Netherlands, février 2005.

[105] S. M. Larson, C. D. Snow, M. Shirts et V. S. Pande : Folding@Home and Genome@Home : Using distributed computing to tackle previously intractable problems in computational biology. *ArXiv e-prints*, janvier 2009.

[106] Erik Learned-Miller et Joseph DeStefano : A probabilstic upper bound on differential entropy. *IEEE Transactions on Information Theory*, 54(11):5223–5230, 2008.

[107] Joseph Y-T. Leung, éditeur. *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. Chapman & Hall/CCR, 2004.

[108] David A. Levin, Yuval Peres et Elizabeth L. Wilmer : *Markov Chains and Mixing Times*. AMS Bookstore, 2009.

[109] Brian Neil Levine, C. Shields et N. B. Margolin : Detecting the Sybil Attack in Mobile Ad hoc Networks. Rapport technique 2006-052, University of Massachusetts Amherst, Amherst, MA, USA, octobre 2006.

[110] Richard J. Lipton et Jonathan S. Sandberg : PRAM : A scalable shared memory. Rapport technique CS-TD-180-88, Princeton University, septembre 1988.

[111] L. Lovász : Random walks on graphs : A survey. *Combinatorics*, 2:1–46, 1993.

[112] Arfst Ludwig, Rolf H. Mohring et Frederik Stork : A Computational Study on Bounding the Makespan Distribution in Stochastic Project Networks. *Annals of Operations Research*, 102(1–4):49–64, février 2001.

[113] D. G. Malcolm, J. H. Roseboom, Charles E. Clark et W. Fazar : Application of a Technique for Research and Development Program Evaluation. *Operations Research*, 7(5):646–669, septembre 1959.

[114] J. J. Martin : Distribution of the Time through a Directed, Acyclic Network. *Operations Research*, 13(1):46–66, janvier 1965.

[115] Makoto Matsumoto et Takuji Nishimura : Mersenne twister : A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[116] Sanjay Mehta et Reha Uzsoy : Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1):15–38, 1999.

[117] A. Pascoletti et P. Serafini : Scalarizing vector optimization problems. *Journal of Optimization Theory and Applications*, 42(4):499–524, 1984.

[118] Michael L. Pinedo : Stochastic Scheduling with Release Dates and Due Dates. *Operations Research*, 31(3):559–572, 1983.

[119] Michael L. Pinedo : *Scheduling : Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008.

[120] J. Scott Provan et Michael O. Ball : The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[121] K. Pruhs, J. Sgall et E. Torng : Online Scheduling. *In* Joseph Y-T. Leung, éditeur : *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*, chapitre 15. Chapman & Hall/CCR, 2004.

[122] Irina Rish, Mark Brodie, Sheng Ma, Natalia Odintsova, Alina Beygelzimer, Genady Grabarnik et Karina Hernandez : Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks*, 16(5):1088–109, septembre 2005.

[123] Pierre Robillard et Michel Trahan : Expected Completion Time in PERT Networks. *Operations Research*, 24(1):177–182, 1976.

[124] Arnold L. Rosenberg : Optimal schedules for cycle-stealing in a network of workstations with a bag-of-tasks workload. *IEEE Transactions on Parallel and Distributed Systems*, 13(2):179–191, 2002.

[125] Arnie Rosenthal : Computing the reliability of complex networks. *SIAM Journal on Applied Mathematics*, 32(2):384–393, mars 1977.

[126] Günter Rudolph et Alexandru Agapie : Convergence Properties of Some Multi-Objective Evolutionary Algorithms. *In Second IEEE Congress on Evolutionary Computation (CEC 2000)*, pages 1010–1016, La Jolla, California, USA, juillet 2000.

[127] Johnatan Eliabeth Pecero Sanchez : *Local-Global scheduling interactions*. Thèse de doctorat, Institut National Polytechnique de Grenoble, 2008.

[128] Sachin S. Sapatnekar et Hongliang Chang : Statistical Timing Analysis Under Spatial Correlations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(9):1467–1482, septembre 2005.

[129] Gilbert Saporta : *Probabilités, analyse des données et statistiques*. Editions Technip, 2006.

[130] Luis F. G. Sarmenta : Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[131] D. Sculli : The Completion Time of PERT Networks. *The Journal of the Operational Research Society*, 34(2):155–158, février 1983.

[132] Sol M. Shatz et Jia-Ping Wang : Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38(1):16–26, 1989.

[133] Sol M. Shatz, Jia-Ping Wang et Masanori Goto : Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, 41(9):1156–1168, 1992.

[134] Vladimir Shestak, Jay Smith, Howard Jay Siegel et Anthony A. Maciejewski : A Stochastic Approach to Measuring the Robustness of Resource Allocations in Distributed Systems. *In Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06)*, pages 459–470, Columbus, Ohio, USA, août 2006.

[135] Zhiao Shi, Emmanuel Jeannot et Jack J. Dongarra : Robust Task Scheduling in Non-Deterministic Heterogeneous Systems. *In Proceedings of IEEE International Conference on Cluster Computing*, Barcelona, Spain, octobre 2006.

[136] Gheorghe Cosmin Silaghi, Filipe Araujo, Luís Moura Silva, Patrício Domingues et Alvaro E. Arenas : Defeating colluding nodes in desktop grid computing platforms. *Journal of Grid Computing*, 7(4):555–573, 2009.

[137] Stephen F. Smith : Reactive scheduling systems. *In Expert Systems and Intelligent Manufacturing*, pages 43–56. Elsevier Science Publishing Co., Inc, 1994.

[138] Jason D. Sonnek, Abhishek Chandra et Jon Weissman : Adaptive reputation-based scheduling on unreliable distributed infrastructures. *IEEE Transactions on Parallel and Distributed Systems*, 18(11):1551–1564, 2007.

[139] M. A. Stephens : Tests based on edf statistics. *In* R. B. D'Agostino et M. A. Stephens, éditeurs : *Goodness-of-Fit Techniques*, pages 97–193. Marcel Dekker, New York, 1986.

[140] Thomas G. Stockham : High-speed convolution and correlation. *In Proceedings of the AFIPS Sprinp Joint Computer Conference*, 1966.

[141] Vincent T'kindt et Jean-Charles Billaut : *Multicriteria Scheduling*. Springer, 2006.

[142] Takao Tobita et Hironori Kasahara : A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[143] Haluk Topcuoglu, Salim Hariri et Min-You Wu : Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, mars 2002.

[144] Vicenç Torra et Yasuo Narukawa : *Modeling Decisions : Information Fusion and Aggregation Operators*. Springer-Verlag New York Inc, 2007.

[145] Leslie G. Valiant : The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[146] Richard M. van Slyke : Monte Carlo Methods and the PERT Problem. *Operations Research*, 11(5):839–860, septembre/octobre 1963.

[147] Chandu Visweswariah, Kaushik Ravindran, Kerim Kalafala, Steven G. Walker, Sambasivan Narayan, Daniel K. Beece, Jeff Piaget, Natesan Venkateswaran et Jeffrey G. Hemmett : First-order incremental block-based statistical timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2170–2180, octobre 2006.

[148] Tobias Wagner, Nicola Beume et Boris Naujoks : Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization. *Lecture Notes in Computer Science*, 4403:742, 2007.

[149] Lee Wang, Howard Jay Siegel, Vwani R. Roychowdhury et Anthony A. Maciejewski : Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, novembre 1997.

[150] Richard R. Weber : Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime. *Journal of Applied Probability*, 19(1):167–182, 1982.

[151] Annie Wu, Han Jin, Kuo-Chi Lin et Guy Schiavone : An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), septembre 2004.

[152] S. David Wu et Robert H. Storer : Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, septembre 1994.

[153] S. David Wu, Robert H. Storer et Pei-Chann Chang : One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20(1):1–14, janvier 1993.

[154] Ronald R. Yager : On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18:183–190, 1988.

[155] Nihal Yazici-Pekergin et Jean-Marc Vincent : Stochastic Bounds on Execution Times of Parallel Programs. *IEEE Transactions on Software Engineering*, 17(10), octobre 1991.

[156] Matthew Yurkewych, Brian Neil Levine et Arnold L. Rosenberg : On the cost-ineffectiveness of redundancy in commercial p2p computing. *In Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 280–288, 2005.

[157] Lizheng Zhang, Weijen Chen, Yuhen Hu et Charlie Chung-ping Chen : Statistical static timing analysis with conditional linear max/min approximation and extended canonical timing model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1183–1191, juin 2006.

[158] Huanyu Zhao et Xiaolin Li : H-trust : A robust and lightweight group reputation system for peer-to-peer desktop grid. *In ICDCSW '08 : Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops*, pages 235–240, Washington, DC, USA, 2008.

[159] Shanyu Zhao, Virginia Mary Lo et Chris Gauthier-Dickey : Result verification and trust-based scheduling in peer-to-peer grids. *In Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), 31 August - 2 September 2005, Konstanz, Germany*, pages 31–38, 2005.

[160] D. Zhu, R. Melhem et D. Mossé : The effects of energy management on reliability in real-time embedded systems. *In International Conference on Computer Aided Design, ICCAD'04*, pages 35–40, San Jose (CA), USA, novembre 2004.

[161] Eckart Zitzler et Simon Künzli : Indicator-Based Selection in Multiobjective Search. *In Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, Birmingham, UK, septembre 2004.

[162] Eckart Zitzler, Marco Laumanns et Lothar Thiele : SPEA2 : Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. *In Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, septembre 2001.

[163] Eckart Zitzler et Lothar Thiele : Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. *Lecture Notes in Computer Science*, pages 292–304, 1998.

[164] Eckart Zitzler et Lothar Thiele : Multiobjective Evolutionary Algorithms : A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[165] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca et Viviane Grunert da Fonseca : Performance Assessment of Multiobjective Optimizers : An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, avril 2003.

[166] Albert Zomaya, Chris Ward et Ben Macey : Genetic scheduling for parallel processor systems : Comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, 10(8), août 1999.

Résumé

Cette thèse traite de l'ordonnancement dans les systèmes distribués. L'objectif est d'étudier l'impact de l'incertitude sur les ordonnancements et de proposer des techniques pour en réduire les effets sur les critères à optimiser.

Nous distinguons plusieurs aspects de l'incertitude en considérant celle liée aux limites des méthodes employées (e.g., modèle imparfait) et celle concernant la variabilité aléatoire qui est inhérente aux phénomènes physiques (e.g., panne matérielle). Nous considérons aussi les incertitudes qui se rapportent à l'ignorance portée sur les mécanismes en jeu dans un système donné (e.g., soumission de tâches en ligne dans une machine parallèle).

En toute généralité, l'ordonnancement est l'étape qui réalise une association ordonnée entre des requêtes (dans notre cas, des tâches) et des ressources (dans notre cas, des processeurs). L'objectif est de réaliser cette association de manière à optimiser des critères d'efficacité (e.g., temps total consacré à l'exécution d'un application) tout en respectant les contraintes définies.

Examiner l'effet de l'incertitude sur les ordonnancements nous amène à considérer les aspects probabilistes et multicritères qui sont traités dans la première partie. La seconde partie repose sur l'analyse de problèmes représentatifs de différentes modalités en terme d'ordonnancement et d'incertitude (comme l'étude de la robustesse ou de la fiabilité des ordonnancements).

Mots-clés : système distribué ; incertitude ; ordonnancement ; stochastique ; probabilité ; multicritère ; optimisation ; heuristique ; robustesse ; fiabilité ; collusion.

Abstract

This thesis consists in revisiting traditional scheduling problematics in computational environments, and considering the adjunction of uncertainty in the models.

We adopt here a wide definition of uncertainty that encompasses the intrinsic stochastic nature of some phenomena (e.g., processor failures that follow a Poissonian distribution) and the imperfection of model characteristics (e.g., inaccuracy of the costs in a model due to a bias in measurements). We also consider uncertainties that stem from indeterminations such as the user behaviors that are uncontrolled although being deterministic.

Scheduling, in its general form, is the operation that assigns requests to resources in some specific way. In distributed environments, we are concerned by a workload (i.e., a set of tasks) that needs to be executed onto a computational platform (i.e., a set of processors). Therefore, our objective is to specify how tasks are mapped onto processors. Produced schedules can be evaluated through many different metrics (e.g., processing time of the workload, resource usage, etc) and finding an optimal schedule relatively to some metric constitutes a challenging issue.

Probabilistic tools and multi-objectives optimization techniques are first proposed for tackling new metrics that arise from the uncertainty. In a second part, we study several uncertainty-related criteria such as the robustness (stability in presence of input variations) or the reliability (probability of success) of a schedule.

Keywords: distributed system; uncertainty; scheduling; stochastic; probability; multicriteria; optimization; heuristic; robustness; reliability; collusion.